A Framework for Interaction and Task Decomposition for Objects Emulating Agency Behavior

Safwan Aly

Dissertation
Submitted to the School of Architecture of
Carnegie Mellon University in fulfillment of the requirements
for the degree of Doctor of Philosophy

School of Architecture
Carnegie Mellon University

Advisory Committee

Ramesh Krishnamurti [Chair]
Professor
School of Architecture
Carnegie Mellon University

Ömer Akin
Professor
School of Architecture
Carnegie Mellon University

Jens Pohl
Professor
College of Architecture
California Polytechnic State University (San Luis Obispo)

Len Myers
Professor
School of Computer Science
California Polytechnic State University (San Luis Obispo)

I hereby declare that I am the author of this dissertation.

I authorize Carnegie Mellon University to lend this dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Carnegie Mellon University to reproduce this dissertation by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Safwan Aly

ABSTRACT

Computational systems for decision support are typically stand-alone tools. These are often designed to provide assistance with respect to a single aspect of the decision making. In a design process, where decision making is integral to the activity, designers use such computational tools to generate alternative solutions, to model and simulate the behavior of the artifact being designed, and to produce design documents. Stand-alone tools provide design assistance, but not without pitfalls:

- Each tool requires designers to commit to a schema of representation.
  In order to examine various aspects of a design, the same artifact would be represented differently according to the schema of the tool employed.
- Interdependency among the various design aspects is rarely examined.
  A lack of a unified representation leaves such examination to the judgement of the designer.
- Designers are often required to provide vast amounts of information even for the smallest task.
- Designers seldom have access to the mechanism by means of which a tool internally decomposes a design problem. Designers are, thus, deprived of opportunities to make decisions that may, incrementally, impact on the evaluation process.

In an attempt to improve the efficiency of these tools, research groups undertook the task of developing models for comprehensive design environments where multiple design tools share the same representation schema. Designers use modeling and generative tools to produce a model of the artifact being designed, where other tools simulate, according to some domain expertise, the behavior of the model within the same shared representation. Such design environments are often described as multi-agent decision making environments. Agents are the designer(s) and/or the computational applications each of which encompasses a specific domain expertise. Agents interact and execute tasks to manipulate the design objects until the collective state of these objects are deemed acceptable by the designer(s).

In this thesis I introduce an enhancement to the design of computational assistant tools, mainly geared toward multi-agent design environments that use shared representation schemes. I propose an expansion of the notion of agency to include design objects in which agents may interact with other agents in the execution of design tasks related to the objects. I call this the *objects as agents* approach, where objects are selectively activated to participate in design decision making sessions to execute tasks regarding their immediate design states. In this sense, an object-agent is a design object that is activated to perform tasks. I provide a framework for an object-agent-based design environment, in which domain applications are global problem solving nodes, object-agents are local coordination and management nodes, and, collectively, the designer(s) act as coordinator and final judge. In this sense, the designer orchestrates this fine grained agent environment through incremental interactions until the model arrives at an acceptable design state. Within this framework, I address various issues pertaining to the notion of agency in design such as autonomy, planning and interaction. Vital to the success of such an object-agent-based design environment is the ability of an object-agent to manage assigned or self-initiated tasks. Managing tasks relies on the ability to decompose and delegate such tasks. I provide a decomposition/aggregation mechanism to enable object-agents to manage their tasks. Such a mechanism provides the designer with access to a wealth of local decision making information.

ACKNOWLEDGMENT

I believe that developing a Ph.D. thesis is a small part of a larger experience, during which I interacted with many people; advisors that helped making an idea an exciting yet a valid research topic; students who made me strive to become a decent teacher; colleagues who made tough times seems enjoyable; administrative personal who eased the way through; and certainly family who made a Ph.D. degree worthwhile. I am not sure I can give each and everyone of them their dues in just a few lines, but at least I can try:

Without an open minded and a sharp principal advisor and a friend such as Ramesh Krishnamurti I would not have been able to develop such an unconventional thesis. Many of the ideas introduced through out the thesis are direct results of fruitful discussions with him. In addition, while co-teaching with him I learned how to prepare and prefect course material, an essential part of making a decent teacher. I was also fortunate to have an advisor such as Jens Pohl during the Masters and the Ph.D. His dreams and eagerness to explore new endeavors truly helped making the idea of this thesis a research reality. His support during the proposal time was fundamental. During the proposal and the development of this thesis Len Myers broad knowledge of the AI field provided a needed credibility and validity and helped me define and tune the idea. Last but certainly not least, Ömer Akin, his insightful comments in research ensures the quality and the integrity of the process and the final product. Ömer's support throughout the duration of my stay at CMU is another key element to the success of the whole experience.

Friends, colleagues, teammates and roommates such as Magd Donia and Georg Suter makes a Ph.D. worth while repeating (sure not!). Especially, with loads of Swiss chocolate shipped monthly from Switzerland by Georg's mom and a daily cup of tea prepared by Georg himself. I certainly owe the entire Suter family! I also cannot skip Judy Kampert, an administrator who keeps the wheels in motion for the entire school. She managed to always solve most of my administrative problems and to make my teaching career very successful and pleasant. Many thanks Judy.

During the Ph.D. I lost my two parents one after the other. A wonderful father, dean, professor, and a world class historian whose support morally and financially was unlimited during his life and beyond. And the best teacher and kind mother that ever existed! God bless both Abdullatif and Zeinab. I don't know if my prayers can pay them back part of what they invested in me. I am certainly very proud of being their son.

After my mother I could not resist having another Zeinab in my life, my daughter is currently carrying the same Z and the same beautiful spirit. Finally, this all goes to my friend, my doctor and my wife Amal. I don't know how did she manage to put up with me during such time (and even afterwards!!), I love you Amal!

Before and after, thanks and praises be all to Allah, most knowledgeable most compassionate and most merciful.

Safwan Aly

Table of Contents

List of Figures

List of Tables

1 Problem Statement

1.1 Belief

The context of this dissertation is architectural design. Within this context, I view the process of design as being characterized by the following distinguishing features:

- Design is an intelligent activity involving complex forms of decision making.
- Design problems, in general, can be decomposed to smaller problems that are easier to handle.
- Designing is a collaborative effort of many individuals or *agents* all of whom may act independently in a self-regulating manner, with the proviso that all work towards an agreed eventual goal state.
- *Agents* work cooperatively to change the current design state.
- The flow of relevant information with respect to any design state is always considered as significant to the designer.
- Certain design values are best decided by judgement of the designer(s).

Design problems are multi-faceted, involving many aspects that contribute, in varying degrees, to the final solution. In architecture (and, also, engineering), the complexity of a design problem is a product of the number of its aspects. In most cases, design tasks are sufficiently complex that the scope of a problem and its solution is beyond the capability of a single contributor. Such problems can be solved in a reasonable time only by decomposing them into more manageable sub-tasks. This requires a team effort, in which the sub-tasks are delegated to members. Team members have pluralistic backgrounds, interests and agendas, yet, they typically agree on a common design solution [Schon 88]. The success

of the collective team effort depends on the organization of team members and resources.

1.2 Computational design environments

Computational design environments are computer systems that are meant to provide designers with assistance in a wide range of design activities which includes, but is not limited to, formulating design specifications and architectural programing, generation of preliminary design alternatives, configuration of details, simulation analysis of performance in respect to various design criteria, and modeling of complete artifact for presentation and, possibly, production.

The development of computational design tools has been mostly oriented along a single tool approach. Each tool simulates the intelligence, knowledge and expertise of a single member of the design team, such as lighting or structural experts in an architectural design team. This notion of stand-alone tools has proven insufficient due to the inevitable need for interaction between the many members of the design team in real world situations. Members plan their activities while keeping in mind the actions of other members. This suggests that the development of a computational design environment should be concerned with the representation of a community of agents that interact by cooperation, coexistence and/or competition.

The notion of multi-agent design environments is an attractive proposition for the following three main reasons. These environments:

- accommodate the diversity of design activities and knowledge, based on geographic or functional criteria.
- provide rich environments based on contributions from multiple agents, where the designer can seek better opportunities of handling the design tasks.
- provide opportunities for reducing complexity by breaking the knowledge down into different cooperative entities.

A number of design environments have been investigated by various research labs. Some have reached closure, e.g., IBDE [Fenves 89], ICADS [Pohl 92 and Myers 93] and EDM [Eastman 92]; others are still under development, e.g., SEED [Flemming 95], ICODES [Pohl 97], SEMPER [Mahdavi 96], and FDCA [Khedro 93][1]. At present, there is still no design environment as such that is

available for commercial use for a variety of reasons, chief of these are the following two:

- For commercial purposes, design environments require vast investments for development. Moreover, they pose difficult marketing issues.
- For technical reasons, design environments are difficult to develop owing to the diversity of the bodies of knowledge involved and the complexity of integration. Additionally, the lack of unified representations makes the use of off-the-shelf stand alone applications inefficient.

There are stand alone architectural design applications that are widely available commercially, e.g., DOE2, CALPAS (energy simulation); Premavera, Quicknet (scheduling and cost analysis); LightScape, LumenMicro, Radius (daylighting simulation), BOSE (acoustic simulation), AutoCAD, MicroStation, ARRIS, FormZ, ProEngineer (geometric modeling). However, designers using such tools have to recreate a model of the building according to the representational needs of each tool.

A computational representation of a design environment relies on individual domain applications (which in this dissertation are considered as agents), domain objects of artifacts being designed, and the flow of information amongst these applications and objects. Typically, domain applications represent domain expertise. These exist in the form of procedural programs, expert systems, or at the very least, as sets of macros. Domain objects encapsulate information about the real world objects they represent. Domain objects exist in the form of sets, libraries or prototypical databases. In general, a computational design environment is a collection of domain expert applications and libraries of prototypical objects. The infrastructure of communication (i.e, local and global massage passing system), translators between representations (such as mapping and language bindings), interfaces (for designer/applications/domain objects/databases), process and configuration management mechanisms (for resource allocation and synchronization) all facilitate the coexistence of the environment players and the interactions amongst them.

1. "Federation of Collaborative Design Agent", a system that facilitates communication among designers in A/E/C. Although this is not a comprehensive design environment I elect to list it here because it is an effort to provide a structure for a comprehensive design environment and it addresses some of the issues investigated in this thesis.

1.3 Problem and Proposal

The problem addressed in this dissertation is focused on the representation of design objects and how applications can be modified to take advantage of the proposed representation in a design environment. The key to the solution of this problem is, I believe and one which I propose and describe in this dissertation, lies in the augmentation of the representation of design objects through agency behavior. It may seem, at first glance, that the development of computational design environments is still at a stage where further enhancement of design object representations is inappropriate or ahead of the game. However, I believe that such augmented representation of design objects opens new needed directions in the development of design environments and, therefore, this research has been focused on the representation of design objects.

Typically, an expert application which represents domain knowledge of a real world expert is an active player, while a design object which represents a real world object is a passive player. Active players possess pertinent knowledge to manipulate the passive players. In other words, agents manipulate objects. Decision making is thus a characteristic of agents.

However, design environments made up of active and passive players typically suffer from some of the following problems though headway has been claimed towards their resolution, e.g., in the IMMACCS system [Pohl 99].

- elimination of rich sources of design information from local nodes[2];
- difficulties to identify problem sources in their immediate settings;
- loss of capability to handle problems at the local level;

2. The IMMACCS (A Multi-Agent Decision-Support System), among other features in the system, relies on an extensive shared representation of objects and their relations, agents with reasoning capabilities, an object browser, and UI with customized views. Modifications to this shared representation is continually updated and accordingly can be viewed across the entire environment through the browser or in the customized views. The browser and views, in this sense, provide direct access to the information of local nodes (objects with their current attribute values and relations). However, when an object own/related information changes, the object is not able to manage the evaluation of its new state in respect to a single or multiple domains. It is the agents (of each domain) who receive/gather the information (using the shared representation and the extensive viewing mechanism) and then evaluate the new state of the object and may accordingly initiate or recommend certain actions.

- inability to handle design problems with a high level of abstraction, or the need for relatively excessive information (which is mostly irrelevant) while dealing with relatively smaller design problems.

In this dissertation, I propose to expand, in a specific way, the notion of representing design knowledge to include the design object level. Passive players (design objects) can be given active roles through agency. This does not necessarily imply fragmentation of domain knowledge, but rather the proportional distribution of knowledge among all members of a decision making environment. In this sense, design objects can acquire agent behavior and, therefore, can manage initiated and assigned tasks during a decision making session. Mainly, these would interact with other environment agents to assess their current situation and continually (or upon request) provide valuable information to other agents of the environment. To make use of this stream of local information, the main concern then becomes the coordination of the acts of potentially large number of distributed decision makers.

As a short hand, I call a design object which is capable of performing such activities an *object-agent*. What object-agents can do and when these can participate in the decision making process are questions that are explored through out this dissertation.

In a computational design environment, design objects represent the artifact being designed at various levels of abstraction. To act as agents, these should be endowed with task management and problem solving knowledge. A potential benefit of this design object agentification approach is the enrichment of the design environment with adequate information about the state of each design object in respect to its performance requirements (see Chapter 5). In addition, design tasks can be broken down into smaller self-regulating sub-tasks that are easier to understand and manage. Such sub-tasks are distributed to the applicable object-agents.

To illustrate the basic premise of the object-agent approach, in an architectural design setting, consider a room, a domain object, that can find its required or prototypical daylighting level from the architectural program (or a prototypical database), and then, can interact with a daylighting application to evaluate its current daylighting level. During the course of executing this daylighting evaluation task the room decomposes the task to its openings (e.g., to determine

the amount of daylighting coming through each opening) and room surfaces (e.g., for reflection). A window may be found to admit less daylighting than anticipated due to its glazing type. Such detailed information can be communicated to the designer to modify the glazing area or take another action to increase the daylighting level of the room (e.g., resizing an existing window or adding another). This type of domain object representation draws in a number of related issues which must be addressed, for instance, the degree of agent autonomy granted to an object-agent in such an environment. In the previous example, one would have to address how, with reference to the room, is a daylighting evaluation task initiated. By the room itself (as an object-agent)? or be assigned to the room by another environment agent? Other issues such as the planning of agent acts, the handling of agent conflicts, dealing with domain object constraints, are each addressed in this dissertation.

1.4 Objectives and Method

The objective of this dissertation is to explore the potential benefits and disadvantages, from a designer stand point, of adopting an object-agent approach in a computational design environment. A consideration of this objective is to develop a model for an object-agent design environment; another is to explore the implications of engineering such a design environment.

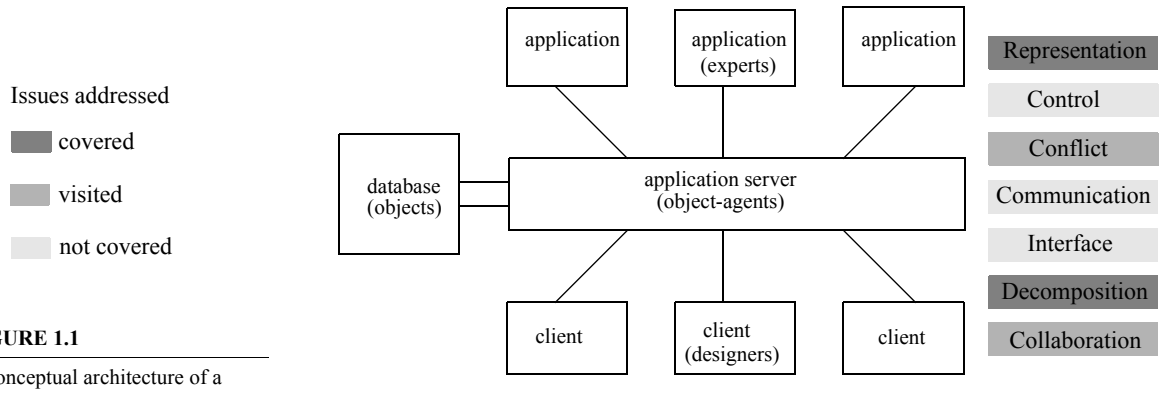


FIGURE 1.1

A conceptual architecture of a multi-agent environment and issues of focus in this dissertation.

Figure 1.1 illustrates a conceptual architecture for a multi-agent design environment without further consideration to the specifics of each component.

The database stores objects, designers use client applications/interface to handle objects and other applications. The application server deals with both objects as passive players and object-agents as active players in dealing with client and expert application requests.

The model has to accommodate an initial set of domain-objects and applications and is, at the same time, expansive to new entities within the same framework. This is achieved through multiple layers of development. To develop such model for an object-agent based environment I use the following method:

- Developing a **framework** for a design object-agent based environment. The framework is comprised of interaction categories. Each category is identified by a set of members (e.g., domain expert agents, tools, databases etc.) which interacts in a context. The players and the context of interaction within each category is identified and discussed in Chapter 3.
- Identifying the **patterns of interactions** among the modules of the framework. This is achieved through the development of a series of general and domain specific scenarios of the object-agent interactions with other environment agents (including the designer). In Chapter 4 scenarios of interaction are captured in event-trace charts [Rumbaugh 91]. The use of charts is enhanced to ease the process of algorithm development for the reusable patterns of interaction.
- Developing a set of **task handling algorithms** that enable an object-agent to manage the execution of its tasks in respect to the developed scenarios. Algorithms for general object-agent tasks are described in Chapter 5.
- Engineering a **detailed implementation design** of such an environment using object models and state diagrams following a rigorous object oriented software methodology [Pree 95] and [Gamma 95].

1.5 Thesis Structure

The outline of the dissertation is as follow:

- Chapter 2 provides additional motivation for adopting an object-agent approach in a computational decision making environment. Chapter 2 reviews the literature of agent environment developments and research efforts, in particular those which attempted to enhance the representation of objects to adopt more intelligent behavior. It also touches upon a

related field namely distributed artificial intelligence (DAI) since the object-agent approach depends on the notion of distribution of activities and decomposition of tasks.

- Chapter 3 outlines a framework of an object-agent based environment and discusses the traditional role of design objects vs. the agentification of design objects, which is a major focus of this proposal. It also presents the theoretical view of the design problem solving activity that involves multiple agents with various capabilities.
- Chapter 4 presents a series of general and domain specific scenarios of agent interactions in an elaborate event-trace form. The event-trace charts are followed by a step by step explanation which provides the bases for the development of a set of interaction and task handling algorithms in the following Chapter.
- Chapter 5 introduces a set of task handling algorithms that are fundamental to object-agents. It also presents a dynamic mechanism for task decomposition that can be used by the object-agents.
- Chapter 6 presents an implementation design of an object-agent design environment using an object oriented software engineering methodology. However, a full scale implementation of a design environment with the object-agents requires team effort and is not part of the scope of this dissertation.
- Chapter 7 identifies the research contributions and the research issues raised by the object-agent approach that need further investigation.

2 Review of Related Work

2.1 Background

The subject matter of this dissertation was proposed and presented in February 1993. At the time distributed artificial intelligence (DAI) and, in particular, the notion of computational agency were emerging research fields. The proposal included a literature review that was geared toward the application of DAI in computer science and engineering design. DAI research has since expanded and the notion of agency is widely adopted. There are multiple applications in use for both research and commercial purposes. Therefore, it is neither important to justify (or validate) the notion of agency in computational decision making environments, nor it is necessary in this dissertation to review and enumerate multi-agent research and application. Instead, design objects with agency behavior are the main focus of this thesis. Accordingly, I provide, in this chapter, a review of agency properties as defined in the literature, as these relate to objects acquiring agency behavior. In addition, I provide a review of related research that is concerned with the enhancement of object behavior, and how such object roles relate and differ from other agents in multi-agent environments.

2.2 Agent and Agency

The Latin word ‘agans’ means ‘to act’. Accordingly, the word ‘agent’ is defined as the producer of an effect, an active substance, a person or thing that performs an action, or a representative. Tokoro [Tokoro 94] considers the later two meanings of the word to best describe the word use in multi-agent research, where an agent is “an individual that performs an action” and a multi-agent system is “a system composed of multiple individuals which perform actions.” Tokoro suggests that multi-agent research (with its two main fields, Distributed

Problem Solving 'DPS' and Agent-Oriented Programming) mainly investigates whether higher level tasks can be achieved by cooperation between multiple sub-systems, each of which has lower ability.

There are no rigorous principles about what constitutes an agent or how an agent should behave. Most dictionary definitions embody three senses of agency: as an actor with the power to act, as an assistant with the power to represent, or as an instrument with the power to effect. Most computational agent definitions require an agent to be 'situated' in an environment from which it receives perceptual input and which it affects by acting autonomously to achieve goals. To cause an effect, agents takes action. Wobcke makes this clear:

> Actions are distinguished from mere random behavior in that action is goal directed whereas not all behavior need be. The clearest cases of action involve deliberation, choice intention, and subsequent execution of an intention, but not all agents have intentions [Wobcke 97].

Computational Agents

Daniel Rasmus suggests that, in a network of agents, an agent must include reasoning capability, embracing beliefs, goals and commitments [Rasmus 95]. The main characteristic of agency (and one which is emphasized throughout this dissertation) is the ability to accomplish self-initiated or assigned tasks. Such assigned tasks are often the result of a decomposition of a larger task and distributed among agents:

> Within its limited domain, an agent will try to accomplish a task. It may be a sub-task of a larger task that a knowledge broker distributed among its brethren, or it may be a simple task like scheduling a meeting [Rasmus 95].

Agent environments should be so designed that the collective efforts of agents toward executing their tasks lead to achieving higher goals. Again, Rasmus makes this clear by the following paragraphs:

> Agents don't do very much as individuals. They know how to schedule a meeting, buy a ticket, or cut a deal for a conference room. They may be the part of some larger conceptual system, eventually aggregating into a material requirements planing system or electronic data exchange system, but as individual bits of knowledge, agents live restricted existences.
>
> Agents need more than an operating system for survival. They require cooperating partners, information sources, and end users. A given set of agents performing a given set of tasks in a given company, end up working in a digital biome [Rasmus 95].

Steiner et al.[Steiner 93] discuss the definition of agent in IMAGINE (an Integrated Multi-AGent INteractive Environment), where an agent may represent software, human or a combination of both. Their definition relies on three characteristics: rationality, cooperation and reactivity:

> Rational Agent: An agent should structure its behavior in a way that, as it reasons, will optimally satisfy its goals. The exact definition of optimality is dependent upon the type of goals of the agent and its ability to reason about achieving them.
>
> Generic Cooperation: When several agents cooperate, they should do so in ways that are, in important respects, independent of particular domain.
>
> Reactivity: The architecture of an agent should be such that it can react in timely fashion to changes in the environment [Steiner 93].

Steiner et al. further draw a distinction between agent tasks and goals, and assumes that an agent will carry out a task, only if it thinks it leads towards a goal (defined as a description of a future state of the world). In fact the authors tie the existence of an agent to the existence of a goal:

> In the simplest model an agent comes into existence with one goal; it derives a course of action, that is a plan, to achieve that goal; it executes the plan; it terminates [Steiner 93].

For an agent to carry out more than one goal Steiner suggest that a more complex taxonomy is needed.[1] According to the rational agent characteristic, the agent may have various ways to select how to reach to a goal. In a multi-agent environment goal decomposition and task delegation may be the only way an agent may reach a goal:

> Agents in multi-agent systems may find that optimal means (from their point of view) to reach their goal is to get other agents to carry out certain actions. The process of several agents working out a future course of action together and carrying it out is how we define cooperation [Steiner 93].

Goals are activated either internally, when an agent react to events in the environment, or externally, when an agent responds to requests. The Steiner et al. model uses goals to direct the firing of actions in a more deliberative planning

1. Since such goals can be conjunctive (when goals conflict with each other), negative (when goals depend on properties that is no longer valid), temporal (when goals depends on agents ability to reason about time), sub-goals (when goals describe a subset of another goal), or predecessor (when goals represent an intermediate state to the ultimate goal).

process. Accordingly, the agents must be provided with planning capabilities to allow them to build plans for given goals. Such planning capabilities vary in its degree of sophistication where "reactive agents may have predefined plans linking their goals immediately to the corresponding tasks." Such predefined plans are based on certain key events in the world and an appropriate reaction to them.

Agent models

Agency can be defined by linking the notion of goals to the ability to perform actions. Wobcke proposes that:

> Agency is best understood as self-controlled goal-directed activity, where the notion of action being under the control of an agent is intimately tied to the agent's ability to perform that action successfully under normal conditions [Wobcke 97].

The 'normal' conditions referred to in Wobcke's proposal is determined by the context of the action taken (or the situatedness). Wobcke does not subscribe entirely to the view that agent ability rests on the repeatability and reliability of the agent to perform an action, nor that the action and its outcome must be completely under the agent's control. Instead, Wobcke requires that the action:

- normally succeeds when it is attempted by the agent;
- is only under the control of the agent if the agent can influence the outcome of the attempt;
- be within the control of the agent in the sense that it is within the agent's power not to do the action (or at least it is possible that the agent cannot do the action).

Wagner proposes a model of an agent based on agent actions. He lists five basic transitions of what he calls 'vivid agent system', namely, perception, reaction, planning, action and replanning:

> A vivid agent is a software-controlled entity whose state is represented by a knowledge-base and whose behavior is represented by means of action and reaction rules [Wagner 96].

Wagner further emphasizes the difference between action and reaction where agent actions are deliberatively planned in order to solve a task or to achieve a goal, while agent reactions are triggered by perception and communication events. An agent needs to react when environmental circumstances demand action. Werner argues that traditional planning approaches are not suitable for

agents since planning needs time, instead reactive strategies are more appropriate to enable an agent to deal with given circumstances [Werner 94]. The complexity of such strategies should correspond to the complexity of the problems that the environment agents are required to deal with. In other words, the complexity of the environment mirrors the complexity of its agent reactions (which in turn represents a higher level reasoning).

In an elaborate research effort to identify the characteristics of an agent, Foner reviews the behavior of a prototypical 'Mud' agent (a multi-person text-based virtual reality agent) known as 'Julia'[2] [Foner 93]. Julia was found to satisfy most of the agency properties identified by Foner. I summarize these properties here:

- Autonomy: where periodic action, spontaneous execution and initiative enable an agent to independently pursue an agenda of tasks.
- Personality: where learning and memory enable an agent to improve its ability to handle tasks across time.
- Discourse: where an agent shares the user's agenda about what and how a task should be executed (resembling a contract about what is to be done).
- Cooperation: where an agent collaborates with the user rather than receiving commands.
- Risk and trust: where a balance between trust and risk is necessary since the notion of task delegation implies both believing that an agent can perform a job but in the same time involves relinguishing control where mistakes can be costly.
- Domain: where the seriousness of the delegated task requires a relative degree of trust (risky domains require more robust agency behavior).
- Graceful degradation: where an agent should strive to execute a task or a subset of the task in case of communication or domain mismatch.
- Expectations: where user expectations from an agent should not exceed agent ability to perform especially in dynamic domains where goals, means and tasks are constantly changing (and therefore the balance between risk and trust is harder to achieve).

2. Developed by Michael Mauldin at the Center for Machine Translation at Carnegie Mellon University, Pittsburgh PA.

- Anthropomorphism: where an agent depicts human behavior. Though agency does not imply a need for anthropomorphism and not all entities that claims such behavior are considered agents.

It should be noted that not of all of these properties are pertinent to agent-based decision making environments. In this dissertation, I consider autonomy, cooperation, domain and expectation.

Commitment

The notions of discourse, risk and trust, domain, graceful degradation and expectations overlap with a widely accepted notion in the multi-agent world known as 'commitment' from an agent to another and is held with respect to some goal. Wooldridge and Jenning [Wooldridge 95b] draw distinction between two commitment related concepts, 'commitment' and 'convention'. They define commitment as a pledge or promise, while convention is a means of monitoring a commitments. A convention specifies both the conditions under which a commitment can be abandoned. This is identified as an important property of commitment called 'commitments persist', that is:

> Having adopted a commitment, we do not expect an agent to drop it until, for some reason, it becomes redundant. The conditions under which a commitment can become redundant are specified in the associated convention- examples include the motivation for goal no longer present, the goal being achieved, and the realization that the goal will never be achieved [Wooldridge 95b].

Durfee et al. [Durfee 92] add another dimension to the notion of commitment, that is, an agent should not only be concerned of how it models other agents but also should consider how it is modeled by other agents. It is of interest to an agent to influence how other agents model it (since an agent takes actions based on its models of the others). During agent interactions, if other agents have a better model of an agent (closer belief about its capabilities and interest) the expected commitment of that agent (or the type of assigned tasks) is more likely to be fulfilled. Models that survive for longer periods of time are typically more abstract:

> By propagating more abstract models of itself, an agent commits itself to fewer specifics, and thus retains more flexibility in the face of its dynamic environment.
>
> Of course, being overly abstract will sometimes make coordination inefficient.
>
> The degree to which the models provide enough information to lead to

> effective collective interactions determines the coordination performance [Durfee 92].

Durfee et al. suggest that an agent can dynamically influence how it is modeled by other agents through control of its communications and observable actions. By engaging in a 'flurry of communication/observation,' agents may modify their models of each other. The rate by which such modifications take place is variable. Durfee et al. describe four different rates of dynamic modification; reactive planning, rescheduling, replanning, and legislation. The four modification rates represent a common hierarchical space of agent behaviors. In reactive planning, agents formulate new action to take based on constant observations; in rescheduling, agents react to a change in schedule; and in replanning agents develop new joint plans in response to unexpected states. Legislation requires new rules of interaction. Durfee et al. conclude that an agent must be able to commit to a range of interactions without necessarily specifying details. Accordingly, an important goal in modeling agents is to allow them to decide dynamically how to best coordinate their actions.

In a later research effort, Singh discusses the notion of commitment in information-rich environments and shows that commitment is acquired by agents as a consequence of adopting a role [Singh 97]. His research is focused on developing abstractions for building flexible cooperative information systems (CISs) to the standard robustness of traditional systems. He presents the pros and cons of using interacting agents in open information systems vs. traditional database transactions and extended (or open) database transactions. The problem of structuring computations in open information systems leads Singh to argue that commitment is essential to help coordinate and structure multi-agent systems achieve coherence in their actions.

> The main problem is to structure activities in a manner that can respect the autonomy of the information resources. The database approach are restrictive. The agent approaches are flexible, but there is need for tools and formal approaches for designing them. In particular, there is need for a notion of commitment that flexibility reflects the organizational structure of how agents interact [Singh 97].

Singh presents an approach called 'Spheres of Commitment' $(SoCom)^3$. In SoCom, agents interact by forming commitments toward one another. Commitments are formed in a context, which is given by the enclosing CIS. A commitment comes with specification of how it may be satisfactorily discharged

and a specification of how it may be canceled (i.e., concomitant commitments). The act of joining a CIS (by an agent) corresponds to creating commitments. Agents must make sure that they have the capabilities and resources required to take any additional role and its concomitant commitments.

The above views of commitment as a property of agency lead to the belief that a representation of an agent must include, at its minimum, the notion of a 'task' as a form of agent commitment. And, in turn, a representation of a task must include representations of both a 'goal' and a 'result' to provide the means to achieve commitment monitoring. Such representation issues are considered in Chapter 6 which relates to implementation design.

Planning and reactiveness

Planning is another property of agency that depends on agent ability to reason about other agents. Russel and Norvig [Russel 95] explain how an agent may reason using first-order logic. They identify five capabilities which must exist in such an agent; reaction, abstraction of states, maintenance of internal models of relevant aspects of the world, sorting and relating actions to circumstances, and using goals in conjunction with knowledge about actions to construct plans. Russel and Norvig share the belief that in multi-agent domains, it becomes important for an agent to reason about the mental process of the other agents.

> In effect, we want the model of the mental objects that are in someone's head (or knowledgebase) and of the mental processes that manipulate those mental objects. The model should be faithful but it does not have to be detailed [Russel 95].

Russel and Norvig then discuss alternative representations of mental objects. They introduce the term 'propositional attitude' that associates with agents behavior such as 'believes', 'knows', and 'wants' in relation to other agents. They present a model based on an approach called 'syntactic theory' where strings written in a representation language represent such mental objects. The argument considers three different directions to enhance the representation of

3. SoCom relies on an older notion used by the database community known as 'Spheres of Control' (SoC) presented in [Davies 78] and re-visited in [Gray 93]. SoC attempts to contain the effects of an action as long as there may be a necessity to undo them. The entire execution history is maintained and its rolled back to undo the effect of committed activities then rolled forward to redo the necessary computations. Singh [Singh 97] argues that a draw back in SoC, as a data-centric approach, is that commitment depends solely on the computation that commits, not on the interplay between the two ends of the commitment, a drawback which favors the notion of multi-agent environments.

mental objects. First, it is unrealistic to expect that there will be any real logical agents and accordingly define limited rational agents which make limited number of deductions in a limited time. Second, axioms for other propositional attitudes (based on a common definition that 'knowledge is justified true belief'), in which case a representation of 'knows what' would include an agent, a term and a predicate that must be true to answer. Finally, propositional attitude changes over time and accordingly introduce a representation of time.

The purpose of such models is to make an agent useful by helping an agent to do some actions it could not have done before or to chose better actions when executing tasks. Actions have 'knowledge preconditions' and 'knowledge effects'. Russel and Norvig draw an important deduction, which supports an important notion presented in this thesis: each action has its own requirements on the form of the knowledge.

Chaib-Draa and Levesque categorize the types of interaction among agents based on the situation; routine situations, familiar situations and unfamiliar situations [Chaib-Draa 94]. They argue that reasoning about other agents is mainly needed when agents face unfamiliar situations. In such events, agents rely on intensive communications when they do not succeed in making decisions about what to do next with other agents. Such arguments assume that coordination is more important in routine situations (where agent behavior is governed by 'stored patterns of predefined procedures, that map directly from perception to an action', known as skill-based level[4]), or in familiar situations (where agent behavior is governed by 'a set of heuristics, that is a set of stacked rules', known as rule-based level), while communication is more important in unfamiliar situation (where agent behavior is governed by 'goal and utility and more generally by reasoning about others', known as knowledge-based level). Accordingly, the authors suggest that:

> Agents should prefer low levels (i.e., routine and familiar situations) than high level (i.e., unfamiliar situations). The reason is that low levels are fast, effortless and are propitious for coordinated activities between agents, whereas the high level is slow laborious and can lead to conflicts between agents [Chaib-Draa 94].

4. As described by Rasmussen [Rasmussen 86] in his skill-rule-knowledge levels which define the hierarchical models of human behavior and reasoning techniques.

In multi-agent environments autonomy leads to uncoordinated activities due to the uncertainty (or ignorance) of each agent's actions. Therefore, to support harmless autonomy, the notion of 'known about the others' should be considered as an integral part of agency in multi-agent environments (since the ability of an agent to initiate actions can be hampered by not knowing about the other agents abilities and actions). However, the Chiab-Draa and Levesque argument suggests that the importance of the notion of 'knowing about others' can be scaled down in environments where unfamiliar situations are minimized or designed not to be handled solely by the computational agents (e.g., in systems designed to allow human intervention in unfamiliar situations such as conflict).

Wooldridge and Jennings support the notion that agents as 'intelligent reactive systems' need to know about the abilities, skills and interests of the other agents of the environment, especially when they have goals that depends on the existing of such community members [Wooldridge 95b]. In a later publication Jennings, Sycara and Wooldridge present an overview in agents and multi-agents systems in which they present an adopted definition of an agent [Jennings 98]. They consider three main concepts: situatedness, autonomy and flexibility; and, accordingly, define an agent as:

> A computer system, situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives [Jennings 98].

According to them, 'Situatedness' means that the agent receives sensory input from an environment that it is situated in (such as the internet) and that it can perform actions which change the environment in some way. 'Autonomy' means that the agent should be able to act without direct intervention of humans (or other agents), and that it should have control over its own actions and internal state. Learning from experience is a stronger sense of autonomy that is acknowledged by the authors but not considered necessary. 'Flexible' means that an agent is first; responsive in a timely fashion to changes that occur in its environment, second; pro-active, meaning to be able exhibit opportunistic, goal directed behavior and take initiative where appropriate, third; social, meaning to be able to interact with other humans and agents to complete their own problem solving and help others with their activities. The authors do acknowledge other aspects such as mobility and adaptability that are considered by other researchers as properties of agency, but believe that the essence of agency is captured by these three key concepts: situatedness, autonomy and flexibility.

An object-agent, as proposed in this thesis, should exhibit a wide range of social and responsive behavior as described above, however, pro-activeness (or initiating actions on its own) depends to a large degree on the ability to interpret sensed information from its environment. In a design environment, with thousands of design objects, many changes can be interpreted as closely related information which may require a large number of agents to initiate actions (such as re-evaluation of own state in respect to the environment changes). Therefore, I consider that filtered or controlled agent pro-activeness are more appropriate in design environments. Filtered or controlled pro-activeness, in turn, reduces the degree of autonomy an agent enjoy (see discussion on object-agent autonomy in design environment in Chapter 4).

Patterns of interaction

Jennings distinguishes multi-agent systems from other software paradigms (such as object-oriented systems, distributed systems, and expert systems) by the complex patterns of interaction that may take place among agents of such characteristics:

> Multi-agent systems are ideally suited to representing problems that have multiple problem solving methods, multiple perspectives and/or multiple problem solving entities. Such systems have the traditional advantage of distributed and concurrent problem solving, but have the additional advantage of sophisticated patterns of interactions. Examples of common types of interactions include: cooperation (working together towards a common aim); coordination (organizing problem solving activity so that harmful interactions are avoided or beneficial interactions are exploited); negotiation (coming to an agreement which is acceptable to all parties involved). It is the flexibility and high-level nature of these interactions which distinguishes multi-agent systems from other forms of software and which provides the underlying power of the paradigm [Jennings 98].

Lyons and Hendriks [Lyons 95] discuss the importance of extracting the inherent patterns of interaction among the environment agents for re-use by the agents to achieve their objectives. They present an approach which allows an agent to dynamically 'exploit' such interaction patterns to achieve reactive behavior. The notion of reusing interaction patterns is emphasized later in this thesis for the purpose of developing the interaction protocols to be used by objects acquiring agency behavior (see Chapter 5).

2.3 Objects vs. Agents

Luck and d'Inverno presented a three-tiered hierarchy of entities comprising objects, agents and autonomous-agents [d'Inverno 96]. In this hierarchy, an 'action' is a discrete event which changes the state of the environment; an attribute is a perceivable feature; a 'goal' is a set of attributes that describe the state of affairs in the world; a 'motivation' is any desire or preference that can lead to the generation of and adoption of goals and which affects the outcome of reasoning or behavioral task intended to satisfy those goals. Accordingly, the three-tiered hierarchy defines an 'object' as an entity with a set of attributes and capabilities to take actions; an 'agent' as an object with a set of goals and finally an 'autonomous agent' as an agent with a set of motivations. In particular, an autonomous agent is any agent which has its own set of motivations. Motivations are non-derivative and governed by internal inaccessible rules, while goals are derivative. In this sense, performing an assigned task is adopting goals of other entities of the environment. This hierarchy narrows the gap between the notion of objects and the notion of agents, and assumes that an agent is an object with goals. However, it stops short from formalizing the dynamic transformation of an object to an agent status as proposed in this thesis.

Jennings, Sycara and Wooldridge [Jennings 98] describe objects (in object-oriented programming) as "entities that encapsulate some state, are able to perform actions, or methods on this state and communicate by message passing." Accordingly they provide three arguments to differentiate between objects and agents. The first argument is around the degree of autonomy and in particular around the self-control over its own behavior. The argument suggests that even though encapsulation provides an object with a degree of control over its own state the notion of public methods (or public instance variables), where other objects can invoke, limits the object control over its own behavior. Normally, in an object oriented system this drawback can be remedied if the objects in the system are designed so that their methods can only be accessed by objects that share common goals with them. However, in a multi-agent system, where agents may be designed by different developers such common goals may not exists, and therefore, agents must have more control over their own behavior "they do not invoke methods upon one-another, but rather request actions to be performed." In other words, the decision of what to perform is a property of the agent receiving the request while in an object case it is a property of the external object invoking the method. The authors acknowledge that a multi-agent system can be implemented using object oriented techniques where a layer of control can

be added to the object methods to provide the objects with more control over its own behavior and thus a stronger degree of autonomy but autonomy is still "not a component in the basic object oriented model."

The second argument is drawn around the agency notion of being 'flexible' with its three elements of autonomous behavior (reactive, pro-active, social). Standard object models are not designed to accommodate such behavior (even though it can be implemented to emulate such behavior). The third argument is that each agent has its own thread of control, so a multi-agent system may contain multiple threads of control executed concurrently, while a standard object model has one thread of control (again multi threaded programming for object models are available in some languages, such as Java, to support concurrency, but that still does not capture the idea of agents as autonomous entities).

Rasmus describes agents as a form of objects with the ability to utilize the resources of environments' they live in:

> Agents turn out to be specialized objects running in a common information environment. Because they are likely to consume and redistribute information, exist in communities, and become subject to a form of natural selection, it would be valuable to introduce some organic metaphors that help define agents and their environments [Rasmus 95].

Accordingly, Rasmus draws similarities between the concept of agent environments and bacteria, where agents interact with the host without being completely part of it, and 'exploit their hosts' native capabilities to compensate for internal deficiencies. For instance, access to databases might be provided by Information Builders EDA/SQL, the agent, then, would not require constructs for SQL, but would use whatever SQL the host employs.

As a second distinction from objects Rasmus suggests that not all agents are necessarily fully formed upon creation. Agents, in some cases, will have the capability to learn new rules that apply to their tasks, or will have the ability to exchange or search for such rules in order to satisfy their changing goals or circumstances. Such a characteristic, in the first glance, seems to suite multi-agent environments where constant goal changing is a common behavior, such as design environments (see Chapter 3 for further discussion about design environments characteristics).

The last example I review is taken from an agent related project from a School of Architecture. Based on the notion of an intelligent object, Schmitt [Schmitt 94] and Smith et al. [Smith 96] present a design system called 'Interactive Design using Intelligent Objects and Models' (IDIOM). The intelligent object in IDIOM is defined as:

> An intelligent object is a part of real case which can be interpreted for each new design task using models of their function, behavior and structure. Models provide explicit representations of physical principles, thereby avoiding the brittleness associated with traditional rule based systems [Smith 96].

The development of IDIOM depends on a prototype for an interactive multi-agent interface named 'Sculptor' [Engeli 96] where design objects contain elementary forms of agency. Conceptually, the objects in Sculptor (essentially simple polyhedra) provide reactive, autonomous, and interactive behavior. Reactive behavior is demonstrated by the objects in the form of falling because of gravity or in collision avoidance with other object. Autonomous behavior is represented in motion and transformation where objects can change their position in three dimensional space over time. The interactive behavior is represented in the communication capabilities of the objects among themselves.

In principal, Sculptor objects did enjoy little of the agency behavior discussed in the earlier sections of this chapter, however, they stop short from being agents due to the lack of object knowledge and autonomy to initiate or decompose and delegate tasks. In addition, they only enjoyed limited ability to conduct complex interactions with the designer or the rest of the environment agents. This was certainly realized by the authors who conclude that the next step for Sculptor is to turn its objects into design agents that can be guided by the designer.

3 Framework of an OA-Based Environment

3.1 Functions of an OA-Based Design Environment

In a computer-based design environment, design objects are treated as information entities without ability to initiate actions external to their domain.[1] The designer (DA[2]) models the design state by manipulating design objects whereas expert-agents (EAs[3]) (e.g., cost-agents, structural-agents, acoustic-agents) evaluate the individual or collective performance of the design objects, based on the design information represented in the model and the prototypical information available in the databases or provided by the designer. Accordingly, EAs may warn about performance paucities. Advanced EAs may be capable of recommending and even carrying out the implementation of recommended changes to the design objects (i.e., their attributes or relations).

The proposed approach based on object-agents (OAs) supports design through interactions among designers and a group of design agents in the course of developing a model. The approach suggests that the design objects (referred to in this thesis as data-object or DOs) themselves can be made responsible to perform and manage various design tasks to assist the DA in making various design decisions.

1. The object methods in an object oriented implementation environment are typically geared toward the manipulation of the object's internal data (or linking internal data with internal data of other objects).
2. The designers are treated as distinct agents and are collectively referred to as the DA (see also footnote 3).
3. The definitions and terminology used in this chapter is given in the glossary (see Appendix A).

The single most important question that must be addressed is: *what can be achieved in such an OA-based design environment ?*

> *"a CAD tool, AI-based or not, should always be seen as a complement to human designers that assists them in tasks where they perform less well, but does not compete in areas where they are doing just fine (as many recognition tasks) or where automation is hard to defend for reasons of principle (as in matters of judgment)"*
>
> Flemming [93]

The proposed OA-based design environment is not intended to automate the design process, instead it is to support the DA by providing adequate information about the current design state, its DOs and their relations and how they satisfy the design requirements. The concept of objects endowed with agency provides the DA with adequate information. An environment where OAs are able to incrementally obtain information (through the interaction with other agents) that is most relevant to their immediate tasks provides the DA with rich facilities for the incremental development of the design state.

Changing states vs. producing solutions

The continuous change of the state of the design objects is required until the current state is considered to be acceptable by the DA. The intention of such an environment is not to provide the DA with an optimum solution for a given design problem, instead, it is to incrementally change the current state of the design through the interaction among the various agents within the environment.

The OA representation is intended to provide the DOs with properties of agency to allow them (when required) to interact and manage other environment agents to carry fundamental design tasks such as evaluation, recommendation, generation, conflict handling, and implementation concerning its own attributes in respect to expected performance.

Evaluation

The DOs can be activated to provide various evaluations of their current state upon DA request. Evaluation tasks may be limited to the collection of the DOs factual information or could be extended to the assessment of the expected performance of the DOs in respect to the specified design goals and requirements. The later evaluation task requires a search for the DOs' performance requirements (prototypical or DA specified) and computations of the current performance values. In this sense, an OA interacts with the environment agents (e.g., DA, SAs) to obtain performance requirements (prototypical, DA specified or represented in the model in various forms such as

constraints networks). The OA then interacts with the EAs which assess the OA performance based on the information provided by the OA and in respect to specified requirements. This dissertation focuses on the evaluation tasks as its means to illustrate the approach advocated.

Recommendation Upon DA request, the OA may extend the evaluation session to obtain recommendations from the EAs (i.e., to suggest necessary changes to the OA state in order to meet the performance requirements). If all recommendations have been, exhaustively, considered and the OA still does not meet its performance requirements, the OA may request a DA interference to either relax the performance criteria or suggest changes that may assist the EAs involved to produce acceptable recommendations.

Generation If there are generative agents, the DA may interact with such generative-EAs to generate new alternatives. Recommendation can be viewed as a limited form of generation. However, recommendation, in a general sense, is the generation of instructions of how the current state can be modified toward a goal state while generation is the complete production of alternative states.[4]

Conflict handling Recommended modifications of attribute values should be checked for potential conflicts before they are implemented. The OA should provide the DA with a list of DO attributes and EAs with interest in the attribute values subject to modification. Each attribute of a DO has a list of other DO attributes and EAs that are interested in the value of such attribute. The interest in any attribute is either registered by the EAs or specified in the DO class (during the creation of the class), or by the DA in the task dependent hierarchy.[5] For instance, a daylighting-EA may be linked to a window glazing area; for any change in the value of this glazing area, the DA should be provided with a list that includes, but not limited to, the daylighting-EA (for potential conflict over the new glazing area). The DA may make use of such information by either modifying the

4. The generated states can be searched, optimized or tested against a goal state. If the generative-EA adopts a constraint generation mechanism the generated states are valid states (in respect to the requirements of the goal state which is implicitly represented by the constraints).
5. See A.5.12. See also Chapter 5 for more details on how the interest of a DO attribute in another DO attribute is registered. Basically, attributes of DO types are linked to attributes of other DO types by default or by DA specifications (who can also link attributes of selected instances of DOs). Attributes of the same DO class may also be linked.

recommendations to avoid potential conflicts with interested DOs and EAs,[6] or by ignoring the list and request the OA to implement the current recommendations. To deal with potential conflicts, the DA will have to request further information about such conflicts, in which case the interested DOs[7] may be activated to evaluate their performance in respect to the recommended change. **Conflict detection** is therefore the beginning of a conflict handling session.

After the potential conflicts are identified, upon DA request, a **conflict resolution** session is run. That is, a conflict resolution is a series of local bilateral evaluation sessions involving the interested DOs and EAs where the decision maker examines various DO attribute values to either resolve the conflict or reach an acceptable state of all the parties involved. Each evaluation session involves the decision maker and one of the interested parties. Evaluation results are communicated to the decision maker, no direct communication amongst the interested parties regarding the conflict is permitted. Validating the conflict resolution results is the sole responsibility of the decision maker.

The DA may reduce the potential conflicts through the control of tasks being executed, DO relations and hierarchies and DOs' lists of interested DOs and EAs (referred hereafter as $interest_{list}$). This is **conflict control or prevention**. The DA should be in control of such conflict handling session (detection and resolution) to eliminate the exponential number of DOs that can be brought into the conflict handling session. Each change in a DO attribute may invoke a number of interested DOs, each of which may accordingly result in changing other attributes values to balance its own performance in respect to other domain requirements. This in turn, may trigger other conflicts and may require the involvement of more DOs and EAs. A large number of DO activations may occur in response to an initial conflict which can cause dependency locks and infinite loops of task assignments or conflict handling sessions. The DA can avoid such situations by limiting the:

- evaluation domains (e.g. daylighting, cost) involved in each session;

6. Detection of conflict is implementation dependent. For instance, if the representation of the DO attributes and relations maintains a constraint network, conflicts can be detected as soon as a propagated constraint violates an existing one. The notion of interested DOs and EAs is specific to the framework presented in this dissertation.
7. Including the DO of the OA that is currently executing the task (only if an attribute of the DO are cross registered in the list of the attribute being modified).

- involved DOs (or OAs);
- the depth of layers of interested DOs in respect to the conflict in hand (i.e., how many layers of interested DOs to be involved in the current session).

In fact, successful bilateral resolution sessions are not always sufficient for resolving conflicts, especially when an indirect conflict is triggered by a bilateral conflict session. For instance, resolving a conflict between a cost and a lighting agents over a window glazing area (e.g., enlarging the glazing area, within the total budget, to admit more lighting) does not insure that the total cost is still within the budget. A thermal agent (not involved in the original bilateral conflict session) may find that enlarging the glazing area increases the heating and air-conditioning load which, in turn, requires a more expensive mechanical system that causes the total cost to be above the budget. The thermal agent may accordingly try to reduce the glazing area of the same window erasing the result of the original bilateral conflict resolution session. Such chain effects can be automatically detected by the system if the history of the conflict session is stored. However, conflict handling requires OAs and EAs to have certain capabilities, a rigorous treatment of which is beyond the scope of this thesis. These capabilities include the continuous monitoring of certain events in the environment or the continuous communication with interested agents and DOs.

Conflict detection does not necessarily require the agents to be aware of the tasks and capabilities of the other parties (i.e., agents involved in the same conflict); on the other hand, direct negotiation does. Agents involved in a direct negotiation process should be aware of the other parties needs and goals. OAs in this sense should obtain planning capabilities that are external to their local coordination knowledge. Conceptually, the OA approach does not impose any restrictions on direct negotiations between the agents. However, within the framework of this thesis, negotiation is only conducted through the DA. That is, complex negotiation between OAs is prohibited; however, OA communication is encouraged. OAs may detect conflicts on their current (or recommended) attribute values and may report it to the DA (only if the DA elects to be informed of the conflicts resulting from performing the assigned tasks). To resolve a conflict the DA may change conflict parameters by modifying the evaluation criteria or the current state of DOs (modifying attribute values, removing or introducing new DOs). After various local evaluation sessions changing the above parameters the DA may choose to temporarily or permanently adopt the current DO state with its unresolved conflicts.

Implementation Upon DA validation of any recommendation or the successful termination of a conflict handling session, the DA may implement or ask each OA involved to carry the implementation of any recommendation concerning its own attributes. To implement a recommendation (e.g., change a window dimension) the OA interacts with the CAD-agent and provides the new attribute values (e.g., new window height) or the attribute values of the new related DOs to be placed (e.g., select a shading device from the DO library for the window).

3.2 Agent Interactions

The creation of OAs to perform various task types depends to a large degree on the ability of the OAs to interact with the different agents of the environment. I consider five functional categories of interaction: activation, query, decision support, interface, database. The players involved in the different categories are illustrated in Figure 3.1. Each functional category is shaded differently and reflects possible agent types involved in its operations and their possible relationships. Three types of relationships are identified. The first two types represent direct interactions.

- *Two-way relations* between agents, where an agent may 'inform' or 'request' information or 'assign' tasks to the other.
- *One-way relations* between an agent and an entity (e.g., DO, database), where an agent may 'request' information, 'update' or 'manipulate' the entity information.
- *Secondary relations* which represent the relation 'has access' to information. These are referred to as secondary since they do not necessarily represent interactions. For instance, the relationship between a CAD-agent and a DO database is an example of a secondary relation.

It is instructive to note that not all agents are involved in all categories of interaction. For example in activation the data-base and query-agents do not have role. Likewise, in decision support the OAs play a role whereas the corresponding DOs do not. Only in query and interface interactions might all types of entities concurrently play a part.

I look at two categories: activation and decision support; and one aspect of the interface category: communication.

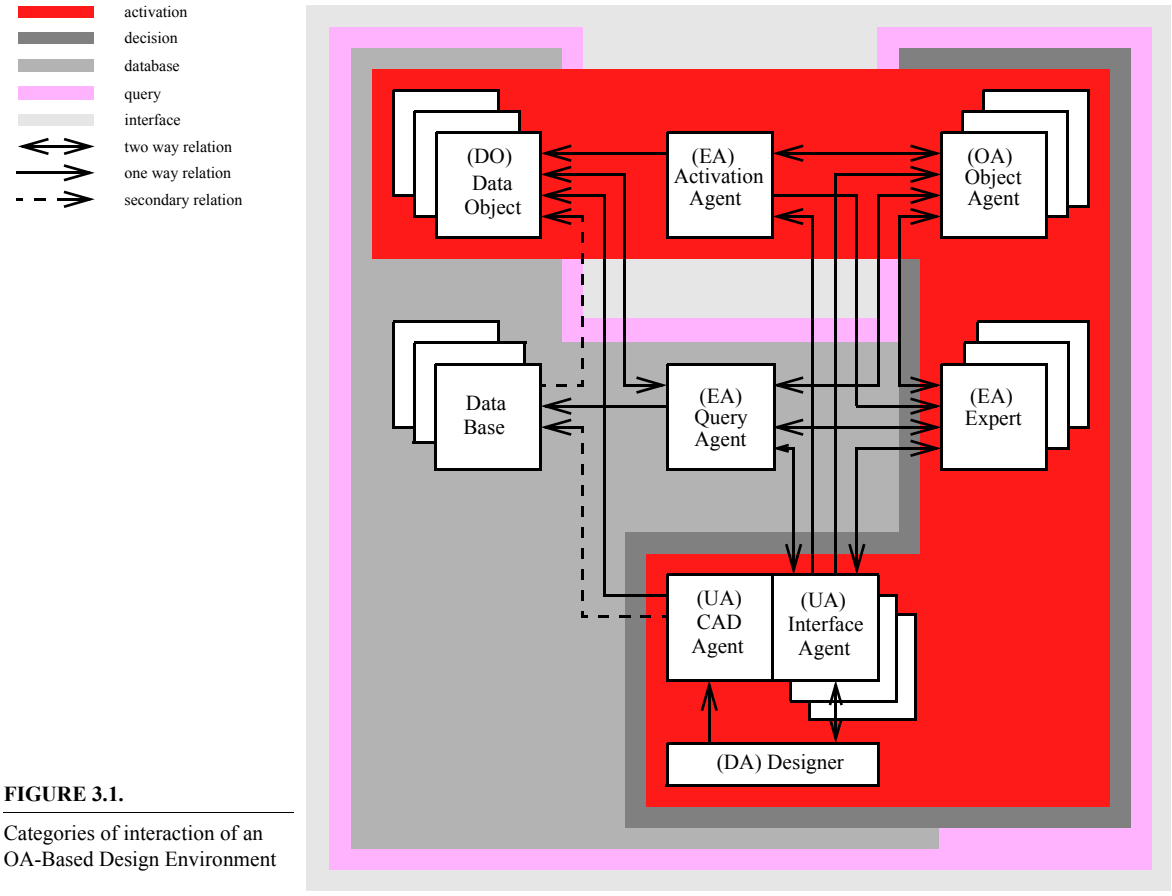


FIGURE 3.1.

Categories of interaction of an OA-Based Design Environment

3.2.1 Activation

Activation is essential to any OA-based environment. There are four main functions: activating a DO, deactivating an OA, loading an EA, and unloading an EA.

Activation

The activation of a DO is the creation of an OA which represents the DO in any interaction that requires agency behavior. The OA contains a copy of the DO attribute values, relations and a copy of the behavior expected from this DO type (e.g., problem solving protocols), and all the properties of agency planted in an OA (Chapter 6 contains detailed description). The created OA acts on behalf of the DO performing tasks assigned to the DO and reporting to the DA and other interested agents when required.

Deactivation The deactivation of an OA is the termination of the OA upon the completion of all assigned tasks. Updating of the DO (i.e., the DO attribute values and relations), informing interested parties with any updates, and informing the DA of any potential or detected conflicts in respect to the updated information, must all be completed prior to termination.

Loading Loading an EA is to invoke the EA in the current session. In an architectural design session the DA may need to temporarily work with set of selected EAs or a single EA, say the lighting-EA to evaluate the current lighting performance of a room, the DA may then load the lighting-EA and unload[8] all other EAs.

Unloading Unloading is then revoking an EA from the current session. An EA cannot be unloaded if it is involved in any task currently being executed.[9]

Activation operates in two modes:

- In the *DA mode*, the DA requests the activation or the deactivation of an OA or EA. The request takes place through the CAD-agent or an interface-agent and are executed by an activation-agent.[10]
- In the *OA mode*, an OA requests the activation of another DO. This request is sent directly to the activation-agent without involvement of the DA.

8. The DA should have the liberty of selecting the appropriate agents for the current session. According to the assigned task other agents (not loaded by the DA) may be brought into play to perform related tasks to the task being executed.
9. Unloading in this sense is one way of disabling an EA from participating in the current session. There may exist other ways of implementing a "disable" function, however, unload is conceptually transparent and helps reduce overhead
10. The activation-agent is a conceptual agent; that is, its existence as a separate entity is implementation dependent. In an object oriented language implementation, as in C++, activation and deactivation methods can reside in the DO class, and the loading and unloading methods can reside in the agent class. In this case there is no need for an independent activation-agent. The activation can be done directly between the requester and the DO and likewise for loading an EA. In a an expert system language implementation, say in CLIPS, an activation-agent can contain all four functions of activation. In such a case any of the four activation requests must be directed to and carried out by the activation agent. While the first approach is direct since only two parties are involved in the activation process, the second is closer to activation, as described in this dissertation.

3.2.2 Decision support

Decision support is the core of an OA-based environment. In this dissertation, this function is to **allow the DA to orchestrate the efforts of the local and global nodes** during the course of performing tasks. The EAs are domain specific problem solving nodes, the OAs are local coordination nodes while the DA is global coordinator, principal planner and evaluator. The DA is the only participant who is aware of the motives behind the group effort led by him/her. Crucial to decision support is DA interaction with agents. The interactions of the DA with the DOs, OAs, and EAs, are facilitated by both CAD-and interface agents.

Upon DA request, the agents provide information (continually or temporally) about the individual or collective states of the DOs. The DA orchestrates the efforts of the agents involved in acquiring information about the current state of the DOs or in changing the current DOs' state toward a goal state (which is either known solely to the DA or represented explicitly in the model).

Decision support involves a variety of task types (see Appendix B): evaluation, recommendation, generation, conflict handling, implementation. Tasks can be *local* as evaluating the current state of a single or multiple DOs with respect to a single domain (e.g., cost), or *global* as evaluating the current state of single or multiple DOs with respect to multiple domains. The state of a DO with respect to a single domain may be unsatisfactory although it may be satisfactory with respect to the collective state of DOs and vice versa.

3.2.3 Communication

How agents communicate ? and *what is being communicated ?* are two fundamental questions that need to be addressed. Communication among agents are facilitated, through interface- agent(s). Conceptually, interface-agents provide a common language to communicate among the various agents, and a message handling system to facilitate their interactions. The communication is available to agents locally, within the same environment, or globally, across remote environments. The interface-agents, common language and message system are implementation dependent.

Interface agents provide the elements of communication; a **message system** to pass information among agents locally within the same environment and globally

across multiple environments; and a common **language** which at the minimum contains a set of terms that triggers agents actions. Such terms adhere to the agents abilities to execute actions, this institutes that new terms are added when agents with new capabilities are introduced to the environment.

Interface-agents may use local and global message passing that is most appropriate to the nature of the environment. For instance, in an object oriented implementation local message system may be replaced by the direct use of object methods of the other environment agent. In other words, agents use object methods to communicate.

The process by which the agents know about the existence of other agents in the environment is important. Conceptually, two approaches are considered:

- The **global communication approach** assumes that an OA does not necessarily know about the other agents of the environment. To assign a task an agent may globally broadcast the task and the appropriate EA responds to the message. This approach emphasizes the role of the common language used by the agents and requires the EA to be continually observing the broadcasted messages within the environment (e.g., through a bulletin board-like system designed to facilitate this type of global communication). This approach complies more with the notion of agency since it does not require the agents to know about the existence or the abilities of the other agents of the environment. On the other hand this approach relies heavily on either the agents' ability to interpret and filter the global messages observed (which can be a very large number considering the number of agents interacting simultaneously).[11]
- The **direct communication approach** assumes that the agents are knowledgeable of the other agents' abilities (in particular, the OA protocols are designed to identify the agent most suited to the task on hand). To insure a response for sent messages the agents must be continually informed as to which agents are currently loaded or unloaded in the environment. In this sense, an agent knows about the existence of other agents and their expertise. Both Lesser and Gmytrasiewicz support this approach and claim that in a cooperative problem solving environment it is more efficient if the agents have detailed models about

11. The number of related messages to be filtered can be reduced using a message classification mechanism provided by the message system.

the other agents in the environment [Lesser 92] and [Gmytrasiewicz 93]. Within the scope of the thesis, this direct communication approach is adopted.

3.3 The modeling process

Computer-based design environments offer various approaches to modeling a design state. Environments where generative mechanisms are at the core of modeling process provide detailed degrees of design alternatives in accordance to a set of defined constraints. For example, in the SEED project the generation of alternative layouts is interactively controlled by the DA within the SEED Layout module [Flemming 95]. The DA selects a layout where another set of alternative 3D projections and building skins can be generated within the SEED-Config module [Woodbury 95]. A set of alternative structural schemes can also be generated within the SEED-Struct module [Fenves 95]. Preceding the generation of layouts, the design requirements and constraints defined by the client and the DA can be formulated as an architectural program within the SEED-Pro module [Akin 95] and [Donia 98]. The main advantage of such interactive and dynamic design environments is to insure the integrity of the design model throughout the different generation process. In addition, no model is generated unless it satisfies the design requirements and constraints as represented throughout the process. This is a 'constrained generation' modeling process [Baykan 92].

With the absence of a generative mechanism, the DA takes a more involved role in modeling the geometric and non-geometric information of the building (or artifact being designed). The DA incrementally introduces the requirements and constraints using his/her design knowledge.[12] In some previous environments, such as ICADS, the model is incrementally evaluated as every component of the building is interactively being added or modified [Pohl 92]. The evaluation process is automated and tied to the construction of the model. This is an 'incremental generate and test' modeling process.

12. The design requirements and constrains are implicitly considered in the models generated by the DA. Firstly since the DA knowledge encompasses such information. Secondly since the utilization of prototypical databases with reusable parts also encompasses chunks of such information. However, the DA take a bigger role in maintaining the integrity of model information.

In other environments, such as SEMPER, the evaluation process does not take place until the model representation is completed [Mahdavi 96]. The DA runs the model through various simulation sessions involving domain expert applications to assess the performance of the current model in respect to the design requirements and constraints. The DA iteratively modifies the current and subsequent models to improve their expected performance until the model is deemed satisfactory. This is a 'generate and test' modeling process.

In order to utilize the OA the DA must incrementally interact with OAs to develop an acceptable model. Therefore, an OA-based design environment lends itself to the 'incremental generate and test' modeling process.

The DA may use a bottom-up approach to develop a building model from an aggregation of rooms, zones, building floors, and building blocks. Alternatively, the DA may wish to follow a top-down approach and first model the building mass, then develop the blocks, floors, zones and rooms within that mass. Both approaches, or a combination of both approaches, are supported in an OA-based design environment. The DA may also use generative agents (if these exist in the environment) to suggest alternative layouts or building blocks. Using a top-down or a bottom-up approach the DA interacts with the environment agents including the OAs to incrementally arrive at an acceptable design model.

To better understand the modeling process by which a DA may interact with a an OA-based environment the following scenario assumes a DA using a bottom-up approach to generate a design model of an office building. Through out this scenario various design issues concerning the architecture of an OA-based environment are discussed.

DO Relations and Hierarchies

The DA selects rooms from a pool of predefined Room-DO types (or defines a new Room-DO type).[13] The DA may then choose to add Wall-DOs, Opening-DOs, Floor-DOs, Ceiling-DOs, etc. and link them to the Room-DO. Linking two DOs is to define the nature relation between them. Three main relation status are currently identified;

- no-relation (the default status).
- constituent-of/contains;

13. The DA needs to follow the implementation specific procedures provided by the environment for defining a new DO type.

- associated-with;

The relation “constituent-of/contains” has two parties involved; a sub-DO and a super-DO (as defined in Chapter 3), where the super-DO contains the sub-DO and the sub-DO is a constituent-of the super-DO. A Wall-DO is logically a sub-DO of a Room-DO. However, it should be permitted that the same Wall-DO be a super-DO of the same Room-DO if needed. The DA should be able to assign any type of relation between any DO types, the logic behind any relation is solely dependent on the DA’s views of how the DOs should be linked. There is no reason why a Wall-DO should not have a constituent-of/contains or associated-with relation with a Room-DO even if it is not geometrically located within the volume of that Room-DO. Visually, thermally and acoustically (for instance) this Wall-DO can still be associated with the Room-DO even if it is located on the volume of an adjacent Room-DO. On the other hand, there may not be a need to establish a relation between a door-DO and a BFloor-DO (building floor) if no agent in the environment can utilize such relation.

A DO can be a sub-DO of more than one DO simultaneously. For instance, a Window-DO can be a sub-DO of a Wall-DO and a Facade-DO in the same time. A Wall-DO can also be a sub-DO of two Room-DOs at the same time in which case the Wall-DO is a joint-DO (see definition in Appendix A).

The relation “associated-with” involves two DOs where a non-hierarchal functional or semantic link is needed in the model. Two DOs may be associated by one or more of their attributes. For instance, A Wall-DO thickness attribute may be associated with a Room-DO thermal and acoustic attributes. The relation “associated-with” is temporarily assigned during a design session (e.g., during the execution of a task as explained below). This relation can be used to register a DO or and EA in the list of interested DOs and EAs of a DO attribute.

A relation between two DOs is task dependent. For instance, an interior Wall-DO that is perpendicular to the facade can be linked to the Facade-DO when the later is performing a task to modify its proportions. The location of interior Wall-DO may consequently be changed if the proportions of the facade is modified (even though the interior Wall-DO is not a constituent-of the Facade-DO). The hierarchy of the DOs is specified by their relations according to the task in hand. That is, the DOs should only have task dependent hierarchies.

The DOs should not have a hierarchy as they reside in the database (i.e., the database should contain a flat set of DOs). Any hierarchy should only be established during the progress of the design session. The DA assigns the relations which, in turn, establish a hierarchy. The DA may also establish multiple hierarchies of the same objects. A Column-DO can exist in a structural hierarchy of a Building-DO and, at the same time, in an enclosure hierarchy of a Room-DO.

The questions are then: *how does the DA assign relations between DOs ?* and *whether it is necessary that the environment provide means to assist the DA in assigning relations and establishing the task depend hierarchies among the DOs ?*.

It can be argued that if the DA is to assign each single relation among DOs, modeling a large building with thousands of DOs becomes a tedious task. The answer to such an argument is that, in most cases, the design state advances in stages by making decisions at any given time. Therefore, the DA need only to assign those relations that are needed for the current tasks on hand. On the contrary, if the environment provides DOs with predefined hierarchies a large number of unnecessary relations are produced and may need to be disabled in order to perform certain tasks. That can be a more tedious task than simply assigning the needed relations. It is necessary that the environment adopts the DA's mental model of the design state and not force the DA to adopt a predefined model imposed by a set of default DO relations and hierarchies.

The environment may provide support to the DA in assigning relations amongst DOs in various ways:

- Through interface-agents which should provide the DA with multiple techniques of assigning relations amongst singular DOs as well as groups of DOs (e.g., establish a relation with all Wall-DOs of a Room-DO, a Floor-DO, or an entire Building-DO). The interface-agents should also allow the DA to disable or eliminate relations and inform the DA of any dependencies which may be effected by such elimination. The existence of DOs in a flat set is independent from its relations, therefore, side effects caused by modifying other DOs should be minimal. In addition, any hierarchy established (to perform a task) during a design session can be saved and reused while performing similar tasks within the same session or in later sessions.

- Predefined hierarchies of DOs may only be used to provide the DA with an experimental test beds. The environment can support the DA by providing exemplary task dependent hierarchies. The DA should be able to import complete or subsets of such hierarchies. The use of such hierarchies is, therefore, dependent on DA preference, and on the task in hand.
- The environment should provide domain specific agents that are geared toward establishing hierarchies amongst the DOs of the model. The DA can interactively (or graphically) use such agents to establish the task dependent hierarchy needed for the task in hand. At any modeling state such agent can be loaded into the current session to suggest and/or assign relations among the DOs, based on the agent knowledge of the task domain and based on the agent's ability to interpret the model (e.g., semantically, or geometrically).

To summarize the discussion about the DO relations and hierarchies:

- DOs have no relations to other DOs unless specified by the DA or other supporting agents according to the DA preference;
- relations between DOs are temporal;
- hierarchies established between DOs are task dependent.

3.4 Decision making with OAs

The OA-based approach suggests that a DO is activated (as an OA) to perform a task regarding its own design state. An OA may perform the task directly or activates other related DO (as sub-OAs) and decompose the task to sub-tasks amongst the sub-OAs. The decomposition is dependent on the relations and hierarchies established between the OA and its sub-OAs.

Decomposition

When performing a task two types of decompositions can be identified as illustrated in Figure 3.2

- Flat/Simple decomposition.
- Complex decomposition.

A *flat decomposition* is performed whenever

- the result of the task assigned to the OA is the aggregation of all the results of the sub-tasks assigned to its sub-OAs.

- each DOs can only be a constituent of one DO (no joint-DO in the hierarchy).

A cost estimate task for a building block materials is an example of a flat hierarchy (Figure 3.3A). The material cost of a building block is the cost of all its material components represented in the a hierarchy as the leaf nodes of the hierarchy tree. The aggregated cost of all leaf nodes regardless of its DO type or its spatial location in the block adds up to the total cost of the building block. The hierarchy needed to perform such a task is established around the constituent-of relations. In such a hierarchy each DO can be a **direct constituent** of the Block-DO.

A cost estimate task may require an aggregation of multiple levels of flat decompositions. Such a hierarchy is needed for a cost estimate task of a building block carpeting classified per room (e.g. the carpeting cost of each room in addition to the total cost of the building block carpeting) (Figure 3.3B). The carpet area need not to be represented in the hierarchy since the area of the room floor can be sufficient for calculating the carpeting area. Various flat decompositions can be established around different classifications of the same task.

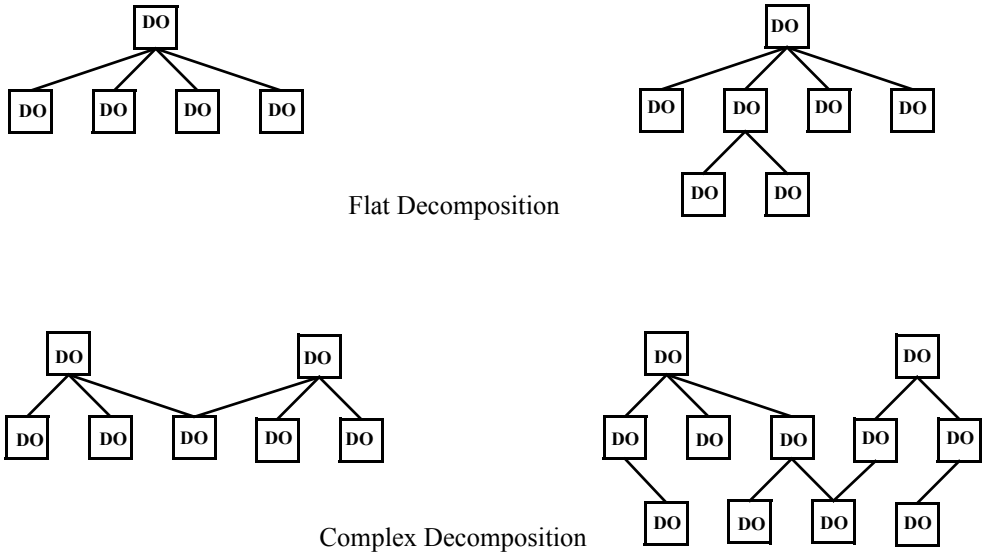
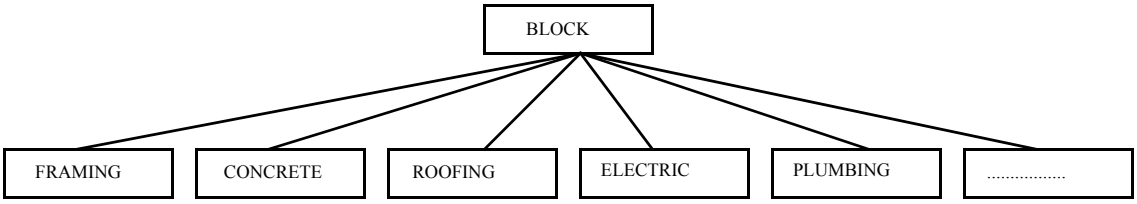
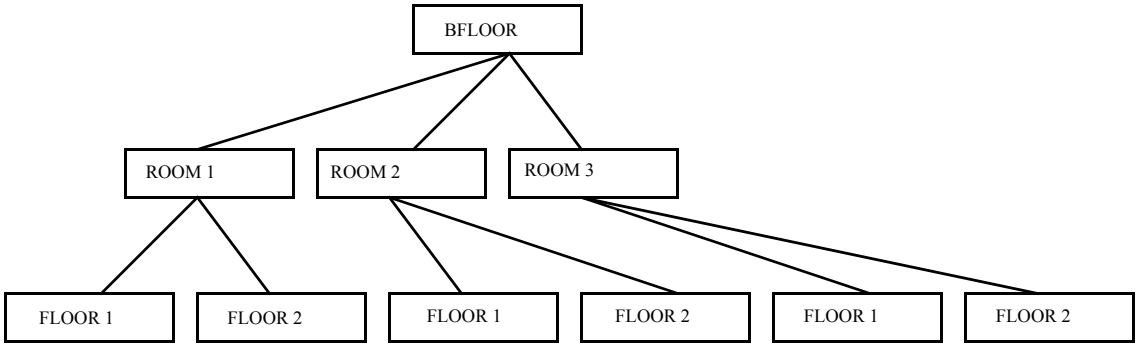


FIGURE 3.2.

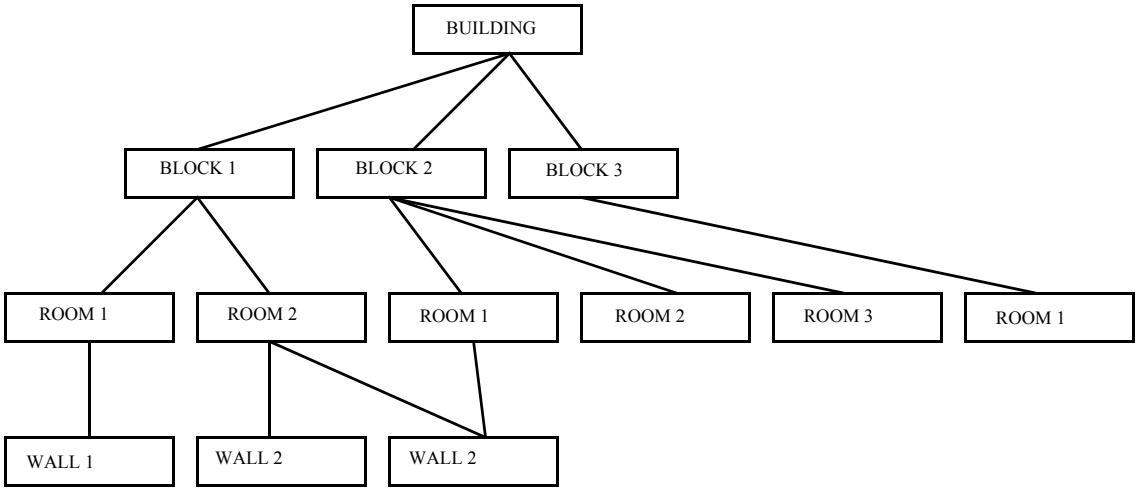
Decomposition types.



(A) Flat Decomposition
cost estimate of building block material (detached blocks)



(B) Flat Decomposition
cost estimate of a block carpeting classified per room



(C) Complex Decomposition
cost estimate of a building frame classified per block

FIGURE 3.3.

Task and decomposition.

A *complex decomposition* is performed whenever

- the result of the task assigned to the OA is not necessarily the aggregation of all the results of the sub-tasks assigned to its sub-OAs.
- at least one DO is a constituent of more than one DO (a joint-DO).

Performing a framing cost estimate for a building classified per building blocks requires a complex decomposition. The frame cost of each building block is the cost of all its frame components (and labor). Some components may be shared (joint-DOs) by other building blocks. A joint-DO such as a shared wall requires an additional layer of computation to determine the exact share of each shared wall (Figure 3.3C). For any classified quantity tack-off task for materials such as paint, dry walls, insulations, pluming, electrical installations all of which may be represented as a constituent of a wall, requires complex decompositions if shared walls are involved. .

Tasks which do not depend entirely on aggregation of sub-results may require complex decompositions as well. For instance, the structure analysis of a building floor is not necessarily the analysis of each individual room of that building floor. The building floor may be divided in structural zones each of which may contain more than one room (some of which may not contain any structural elements). The structural analysis of a zone may not be independent from other zones as well. Columns can be shared among rooms and loads may be distributed along continuous beams or frames which runs across multiple zones. In addition, different structure systems may exist in the same floor which require separate or different structural analysis method. In a multi-story building of identical floors the aggregation of the individual structural analysis of each building floor (top down) may be a valid decomposition.

The hierarchy (or hierarchies) established for a structural analysis task is completely dependent primarily on the type of suggested structure and on the DO type (e.g., Building-DO, BFloor-DO, Room-DO).

Comparing alternative structural systems the DA may need to find the total cost of a structural system of the building. The aggregation of the cost of all structural elements such as foundation, columns, beams, trusses constitutes a valid decomposition (considering the labor cost). On the other hand, if classification according to individual rooms is requested, the same decomposition may not be sufficient. As discussed previously, a Room-DO may

contain no structural elements such as columns and beam; however, this does not imply that there is no structural cost to this Room-DO. The structural cost of a space is related to its floor area, the distribution of the total cost of the structural system of the building among the total floor area provides the structural cost per square foot. This is an average cost vs. actual cost. Average cost is more appropriate in buildings with homogeneous structures[14] (Figure 3.4).

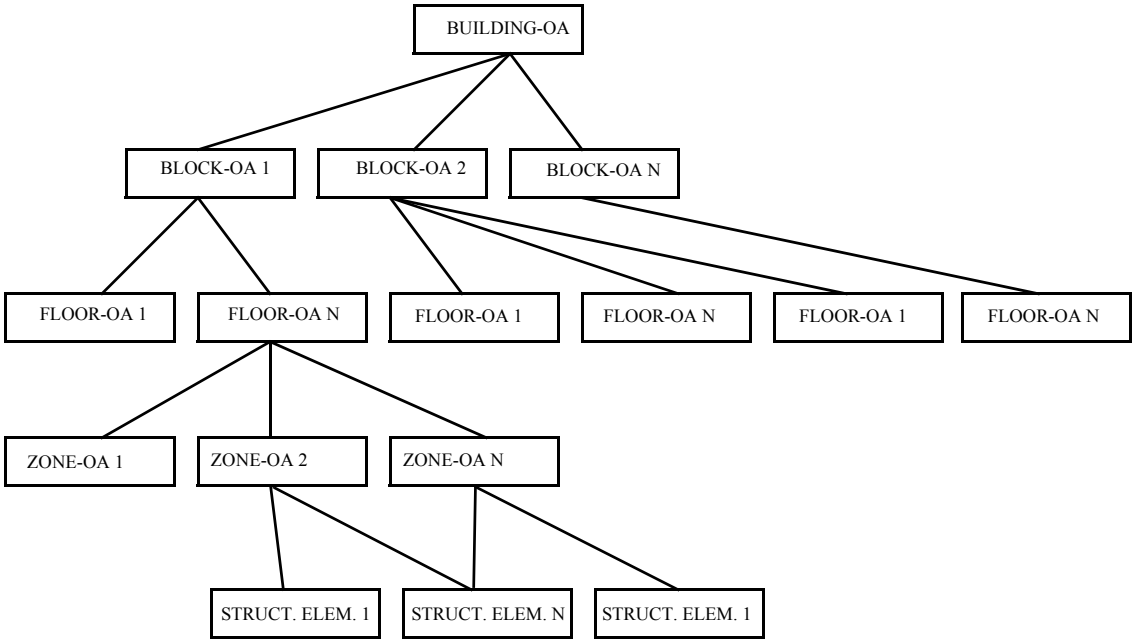
There are many ways by which a building can be decomposed, according to its spatial components such as blocks, floors, zones, rooms, or according to its internal subsystems such as structural, electrical, thermal etc., or according to its functional use of spaces such as management zones, working zones etc. To perform an evaluation of building using an OA-based environment the appropriate decomposition must be applied. Four main factors affect the required decomposition. Three of which are task related: DO type, task domain, task type, and task focus.

It is, therefore, more appropriate to allow the DA to establish the hierarchies according to the nature of task in hand. In such case, task decomposition among sub-DOs can be a direct reflection of the established hierarchy. The question is then how does an OA decompose a task among it is sub-DOs?

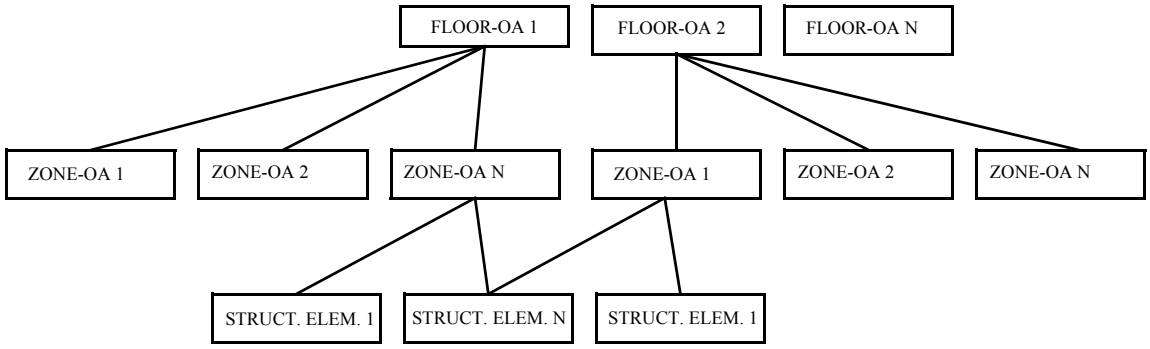
How do the OAs decompose a task?

An OAs knowledge of how to handle any task in hand is embedded within its problem solving protocols. These protocols are general guidelines of how a task can be decomposed when necessary and of how sub-tasks are delegated to the sub-OAs, or executed by the OA directly when no decomposition is necessary. The protocols are therefore specific to DO-type and task domain, type and focus (see Chapter 6 for detailed examples of decomposition protocols). For instance, a Wall-DO would have an evaluation protocol that is specific to cost tasks. Using such protocol a Wall-DO may return its total cost based on average costs of such a wall type as provided in the prototypical database. If classification is requested the DA may need to establish a constituent-of relations to its components (e.g.,

14. A multi-story office building with the same structure in each floor is a homogeneous structure. A multipurpose building may contain multiple structures such as a concrete frame for a theater space next to a skeleton for an office space. Such building structure is a non-homogeneous. In which case the building may have an average cost for each structural system and general average cost for entire building (including all structural systems).



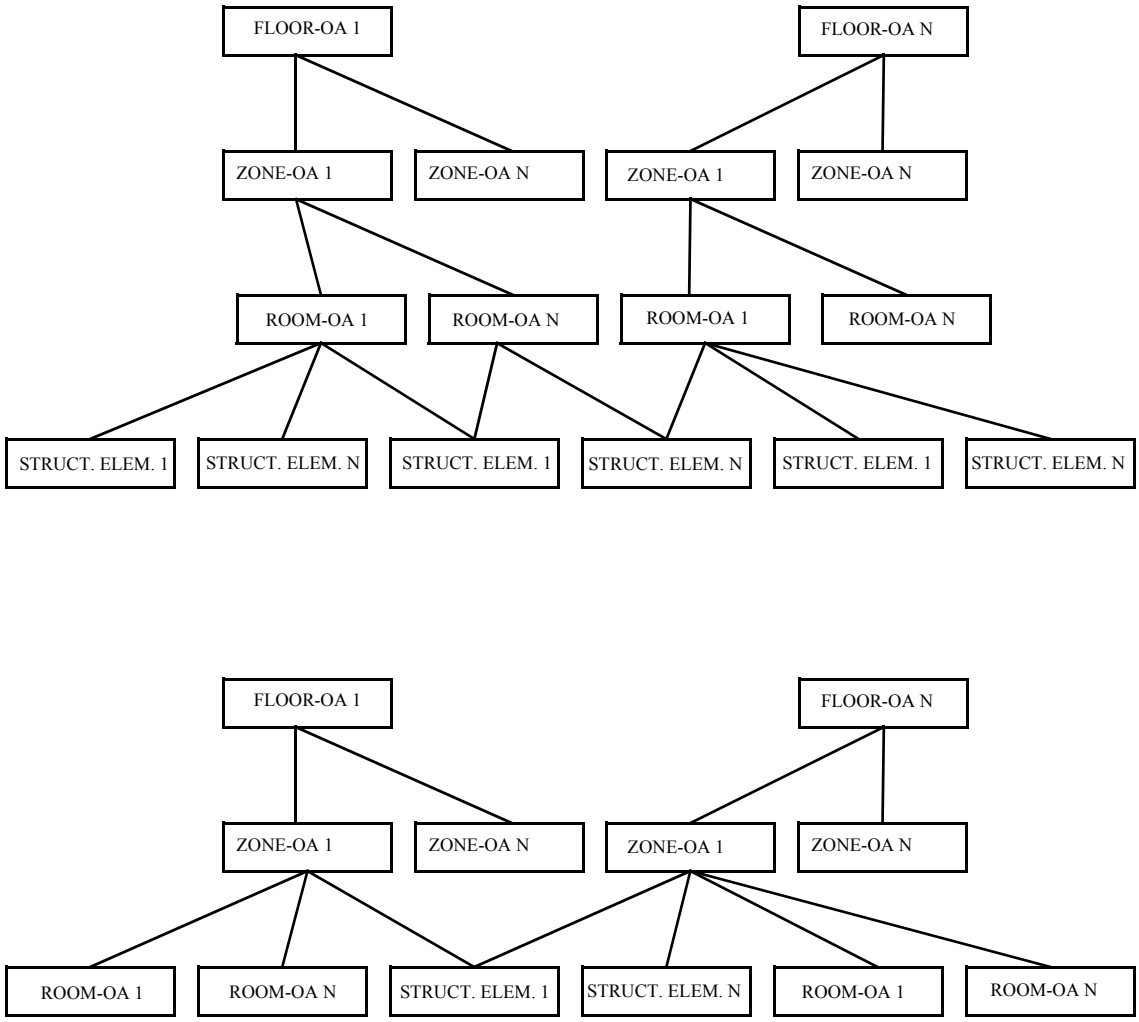
Structural analysis of a building



Cost estimate of structural elements (classified per zone)

FIGURE 3.4.

Hierarchy and decomposition 1.



(A) & (B) Two examples of cost estimate of a bfloor structural elements (classified per room)

FIGURE 3.5.

Hierarchy and decomposition 2.

wood frame, dry walls). The protocol would use this hierarchy to activate the sub-DO and delegate the tasks to the sub-OAs.

If a building bfloor-OA is not linked in a hierarchy with its structural elements, the OA would (when assigned a structural analysis task) interact with the structural-agent to provide the DA with a structural analysis based solely on the

bfloor own geometry and attributes. If the DA establishes a hierarchy between the bfloor-OA and structural elements, the bfloor-OA would be able to provide the structural-agent with more information about its sub-structural elements and relations. The structure-agent, in turn, would be able to provide more specific analysis of the building floor structural performance (see the section on The Level of Abstraction on page 46). In another words, the OA needs a hierarchy to apply the more sophisticated problem solving protocols. The OA may assist the DA in establishing the hierarchies needed. Upon request by the DA the OA provides the DA with the information embedded in the protocols of how a task should be decomposed. Accordingly, the DA should be able to interactively establish the relations needed among the related DOs. Since the hierarchies are used locally (i.e. task specific), there is a strong argument to allow the OAs to establish the required hierarchies needed by themselves. Interacting with agents that are capable of interpreting the functional or the spatial relation among the candidate DOs an OA may be able to acquire the needed hierarchy for the task in hand. In such case the use of the established hierarchy for task decomposition is better monitored or validated by the DA.

Each problem solving protocol is primarily intended to enable the OA to locate and interact with the appropriate EAs to accomplish the task in hand, and when necessary to decompose the assigned task to a set of sub-tasks, delegate the sub-tasks to other OAs (namely its sub-OAs), and manage the sub-OAs while executing the sub-tasks. It does not pertain any domain specific knowledge of how to execute the task (e.g., how to calculate the cost or how to analyze or recommend a structural system). In a short, the problem solving protocols provide the OA with management and planning knowledge regarding the task types to be performed.

When a new DO-types is added to the environment a set of protocols applicable to such type must be made available to its OAs. If multiple DOs of different DO-types is to be activated as one composite-OA (e.g., a corner-OA which may be a composition of walls floors and ceilings) a new set of protocols need also to be made available to such composite-OA. The aggregation of the protocols of the DO types involved in the composite-OA does not necessarily represent the required behavior of the composite-OA.

Evaluating the model

When a DA adds a new DO to the environment, the DO remains in passive status until the DA links it to a hierarchy and activates it in order to perform a task (e.g.,

activating a Room-DO to evaluate the its daylighting performance). The DO is activated and the newly created Room-OA starts interacting with the appropriate agents which will assist in executing the assigned task.

The problem solving protocols of the Room-DO type which is loaded into the room-OA during its creation prompts it to first try to identify the daylighting requirements for this room (e.g., bedroom, reading space). This can be achieved by either posting a global request to which other agents may respond, or by direct communications (see Communications, Section 3.2.3*)* with the query-agent that is responsible for browsing through the prototypical databases. In either case, the room-OA obtains the required daylighting levels either through the query-agent or from other agents in the environment that have access to such information (such as another room-OA of the same type that is concurrently executing a similar task) or, finally, from the DA if no other agent is able to provide the required information.

The room-OA then assigns an evaluation task to the domain EA (i.e., the agent most related to the task in hand, namely the daylighting-EA in this case). The daylighting-EA requests information about the room-OA such as its dimensions, orientation, where it is located in respect to its neighbors, number of openings, opening sizes, and non-geometric information such as surface reflectivity, glazing type, overhangs and so on.

The OA should be able to provide information about itself, whether this information is geometric or non-geometric. Its geometric boundary and its coordinates are residing in its original DO or obtained through interaction with the CAD-agent. Its geometric relation to other DOs is calculable, upon request. Such calculations should be performed based on actual request, however, selected information may be stored temporarily and therefore calculations may not necessarily be performed upon each request. To obtain such information the room-OA would assign a task to a spatial-relations-agent to find specific information relating to its adjacencies. The spatial-relations-agent performs the necessary calculations, and provide the results back to the room-OA which, in turn, provided to the requester.

If the lighting levels are found to be below the required values (which is obtained from the prototypical database or from the DA), the room-OA notifies the DA

that current lighting level are below required, which in turns require further modifications to the current state in order to meet the performance requirements.

The Level of Abstraction

The information provided to the EA by the OA should be relative to the degree of abstraction of the model. Therefore, an EA should be able to provide the appropriate level of response to the level of model abstraction. For a room-OA, the level of abstraction can vary from a simple 2D geometric configurations to a solid complex objects with attributes, constraints and so on. A Room-DO may be represented as a labeled rectangle or as a 3D solid enclosure. It may be linked in a hierarchy with its walls. A wall can be represented as a solid with attributes such as surface colors and materials and can be linked to openings with attributes such as glazing number, reflectivity, types and so on. The lower the level of abstraction of the OAs the more detailed the EAs' response should be. The minimum level of abstraction that an EA can respond to is dependent on both the task domain and type. For instance, a zoning-EA should be able to perform an evaluation task based on the room use and minimal spatial information such as its coordinates. A structural-EA, meanwhile, may not be able to respond to the same level of information when performing a structural evaluation task. The same information may be sufficient if the task is a structural recommendation task. Based solely on the room dimensions and location in respect to the neighboring rooms the structural-EA should be able to recommend a structural schema (e.g., wood, skeleton, steel), and possibly specify the location and dimensions of the structural elements needed.

How does an EA deal with various levels of abstraction of the information provided by an OA ?

Two main factors contribute to answering this question; the design of the EAs and the role of the interface-agents.

The design of an EA

An EA should not be designed to expect a complete set of information before it provides a response. An EA should also strive to obtain any missing information to complete the minimal set required to provide a response.

Typically an EA requests all the information it needs to provide a detailed response to the assigned task. The OA provides relative information which may be a subset of the information requested by the EA. The algorithms of the EAs should be designed to enable the OA to handle any subset of information received from the OA. The response should be relative to the amount of

information provided by the OA. If necessary, the EA may request more information from a query-agent or from the DA, or may inform the DA that the information provided is inadequate or not compatible with the assigned task.

The role of the interface agents

An alternative to changing the design of the EAs is to make the interface-agents (which is facilitating the interactions between the EA and the OA) responsible for recognizing the level of abstraction of the information provided by an OA before it is delivered to the EA. The interface-agent would also be responsible to complete the minimal set of information needed by the EA to perform the assigned task. In this sense, an interface-agent must know what is needed by each EA in the environment. The knowledge of the interface-agent would be altered when new EAs are added to the environment. Conceptually, this is a violation of the notion of agency for both the interface-agents and the EAs. An interface-agent should not pertain to any domain specific knowledge and accordingly its performance should not be affected when new EAs are added to the environment. The role of the interface-agents should, therefore, be limited to *how* to facilitate the interactions among agents and not to *what* is being interacted with. On the other hand, the design of the EA problem solving protocols should not depend on the existence of intermediate agents to complete, filter or classify the information sent to them. An EA should be able to independently react to any received information.

Executing tasks in parallel

When assigned a task, an EA might be working on a prior task from another agent. In this case, the new request can either place the tasks in a queue, or spawn a duplicate process to perform the task concurrently. An OA, should always handle multiple tasks in parallel by duplicating itself. This requires the OA to have access to process management knowledge (which may reside in some UAs of the environment). This also requires a more complex mechanism for updating the original DO data upon the termination of any task while the concurrent tasks are being executed using the initial un-updated DO data. Note that DO data are not updated until all duplicates have completed their tasks.

3.5 Advancing a design state with multiple OAs

A decision making environment that comprises multiple agents relies, to a large degree, on the contribution of each agent to the collective effort of the group. The contribution of an agent depends on its degree of autonomy and its ability to plan and execute actions. The following sections discuss the various degrees of autonomy of an agent, and the planning capabilities that each type of agent may

incorporate according to the proposed design environment. The following section presents the proposed approach of how the cooperation of agents with different capabilities can support the design activities.

3.5.1 Agent autonomy

The term autonomy describes the degree to which an agent controls its own activation, execution and termination. Non-autonomous agents are slaves to external agents that trigger them. Autonomous agents decide for themselves when they should activate, execute and terminate. Semi-autonomous agents turn to an active state by a combination of their own and external commands.

EAs, such as query agents, are primarily non-autonomous since they can only act upon request for information or service by other agents. However, it may be possible that an SA, especially EAs, can self-activate when they see fit. This requires the EAs to be able to identify those problems that relate to their area of expertise. In systems such as ICADS, the intelligent design tools (IDTs) run continuously to evaluate the current values of the evolving solution. Whenever a new design object is added to the CAD environment, the IDTs are automatically activated [Pohl 92].

Quadrel describes a system comprising an asynchronous team of autonomous agents (only system-agents), that are sensitive to events in the environment at large, in a network like structure [Quadrel 91]. When applied to design tasks, the coordination of such an organization is rather complex even when the kind of agents are limited to SAs only. In an OA-based design environment with a large number of agents (SAs and OAs), coordination is an extremely complex task if the OAs are to be fully autonomous.

Agency behavior implies that an OA, as agent, should have the abilities to self-activate itself when it sees fit. This requires that an OA should have the ability to interpret the other agent actions and to coordinate its actions accordingly. The coordination of the activities conducted by agents depends on the ability of each individual agent to plan its activity and to participate in plans made by other agents in the environment, including the DA. Rothman suggests that agents can be classified to many levels of complexity, but they can only be considered intelligent when they possess planning capabilities [Rothman 93]. An agent creates ‘plans’ based on ‘models’ of itself and the environment from which action sequences, consisting of instruction level commands, are generated. The

models are used to predict possible future events and states. An agent that is not able to anticipate future events through the use of models is called reactive. Reactive agents respond only to the current and past states of the environment. To construct such models, agents must obtain communication capabilities with the rest of the environment in order to be able to acquire information to generate such complex behaviors. Therefore, planning (or intelligence of an agent) is an emergent property of the interactions.

Within the scope of this thesis OAs are semi-autonomous agents. That is, they should be activated when there is a task to be performed. Accordingly, there would be no fully autonomous agents apart from the DA.

3.5.2 Short term planning vs. long term planning in design

Any type of planning aims at a set of DA goals to be achieved and a set of requirements to be met. In short term planning, agents monitor the situation and take actions in reaction to it. The reaction is triggered by information from other agents. These are considered data-driven actions. The agents follow rules to map states to actions without a long-term view of how actions will lead to achieving goals. While Durfee describes this type of planning as 'reactive planning', he considers it also important for a problem solving environment to adopt what he calls 'strategic planning' [Durfee 88]. Strategic planning is a form of long term planning where an entire sequence of actions is to be taken starting from an initial state to a goal state. These are considered goal-directed actions.

In long term planning, a set of global goals are to be accomplished. Local and sub-goals are set to distribute the tasks among the participating agents. It is possible to achieve the long term global goals even if a group of the local and sub-goals are modified or changed during the execution of the plan. However, it is difficult to deal with a long term plan when both global and local goals are subject to continuous modification and change. DAs tend to change a considerable number of their design goals during the process of design. In turn, the goals of cooperation between the various agents involved may differ as the design develops, and the style of cooperation may depend heavily on the problem domain. Accordingly, a dynamic set of coordination mechanisms are needed to allow the agents to achieve the appropriate goals of cooperation in many given situations. Such coordination mechanisms would be necessary if agents are to be responsible for long term planning. This is a very questionable proposition and requires further discussion.

It is important to emphasize that global design goals do exist at any point during the design, but they may differ at different points in time. If the OAs are to be responsible for long term planning and designing a sequence of problem solving procedures, it would be necessary for each OA to take into account the possible role of the other OAs that may be involved, and the requirements of the other OAs in respect to their design domains (e.g., acoustics, structure, cost). There exist many sequences of actions that lead the OAs to different states where these global design goals are totally or partially satisfied. The possible number of paths from a current state to a long term goal state are typically large. There may also exist many ways of judging the relevance of the current state to the goal state. Therefore, it is neither feasible nor necessary to encode, in the knowledge of the OA, either a long term planning strategy (or a set of strategies) that enables it to consider the different possibilities of how to arrive at a goal state, or an evaluation mechanism to judge the current state. On the other hand, if the planning strategy is to adopt one or even a few alternatives it may restrict the system from accommodating many feasible paths. This constrains the creative nature of design and may eliminate interesting design alternatives that could be realized by the DA during the design process.

Since changing goals are a property of design, especially creative design, it is appropriate to emphasize the role of the DA as the principal long term or strategic planner while agents (SAs or OAs) should focus mainly on short term activities, and therefore, should be endowed with knowledge that enables them to only execute short term and reactive plans.

Accordingly, I suggest that the DA should be responsible for the long term planning and for the collective evaluation of the different states of design. However, it does not rule out the possible involvement of the OAs if their knowledge is further enhanced so that they are able to support the capabilities of the DA for long term planning or for evaluation. This endorses the notion of changing goals and emerging ideas that distinguishes design from other practices. Distributing the roles among the DA and the other agents according to such an approach does not impose on the DA a specific design process.

Goals for short term planning of immediate tasks with fewer facets can be defined and evaluated in a less complicated fashion than larger tasks with many facets. Distributing the tasks among small entities, such as the OAs, makes it feasible to set an acceptance criterion for each task.

According to the approach proposed, OAs are to deal with small and immediate tasks, which are more applicable to short term planning strategies. Each OA is provided with problem solving protocols that are appropriate to its data-object type (e.g., wall, floor, space). The protocols contain sets of domain specific rules to deal with conventional problems in each domain (e.g., problems relating to daylighting). The protocols are not intended to provide predefined solutions for predefined problems, they should only provide guidelines for how the OA should react in respect to each class of problems (e.g., decomposing, delegating, managing).[15] The DA should be able to interactively modify these protocols to meet the needs of different design situations. The OAs change their status when necessary and to the extent of the knowledge imbedded in their protocols.

The change of an OA status also depends on the support and response of other agents in the environment. Each change in an OA status is an incremental change for the entire design state. It is up to the DA to decide whether the change made by an OA serves the design goals, even if all other agents in the environment do not object to the change. The DA may not be aware of many of the individual activities of the agents. Further, it is not intended that the DA guide each event conducted by each agent. However, it is the DA's responsibility to guide the efforts of the agents toward a goal state and to force conversions when he/she sees fit. While the agents in the environment may not be aware of the DA's intentions, it is the DA who should recognize solution opportunities and orchestrate the agents to arrive at an acceptable state.

The DA's role is to evaluate the current state (independently or with the support of other agents), and to participate in the process of changing the design state by manipulating the DOs (i.e., introducing new DOs to the CAD environment, modifying attributes of current OAs, etc.), and even by modifying the design goals. More importantly, the DA is required to direct and guide the effort of the other agents to advance the current state towards an acceptable design. An acceptable design, in this case, is the state of the DOs and their relations which the DA considers satisfactory, even if the initial design requirements are not fully met. Such an environment can better accommodate the possible change in DA goals that may occur as a result of the incremental change and development.

15. It is possible to store solutions for conventional domain problems in a prototype database which OAs can access (directly or through query-agent), however, this is beyond the scope of this thesis.

4 From Scenarios to Interaction Algorithms

4.1 Event-trace Charts

An OA-based environment is a highly interactive system. Dynamic models show the time-dependent behavior of the system and the objects in it (agents and DOs in an OA-based environment). In an interactive system, logical correctness depends on the sequence of interactions, not the exact time of the interactions [Rumbaugh et al. 91]. The concept of event-traces are used to demonstrate how an OA-based design environment operates. Each event-trace typically identifies a **scenario of interactions among different agents across time through the performance and execution of a task**. As a concept, event-traces are not novel [Rumbugh, 91]. In this dissertation I have extended the original notion of event-traces to include loops, conditional events, non-interaction events (self executed events) and referencing of external event-traces (simulating function calls where the same members of the event-trace interact to execute related tasks in order to execute the task in hand).

Event-traces are best described through charts that reflect two-dimensional relationships over time between entities (namely DOs and agents), where an event is recognized, in the chart, by the passing of a message between a pair of entities at some time moment. Each message is associated with a sub-task.

In this chapter, I describe charts that illustrate a variety of scenarios for an architectural design session based on an OA design environment. Two sets of charts are developed for general and domain specific tasks. The general charts shows the main interactions among agents needed to perform tasks of the certain type such as evaluation tasks, conflict handling tasks, etc. (no domain is specified). The domain specific charts shows detailed scenarios of the expected

interactions among different agents to perform an actual assigned task such as cost evaluation of a room or handling conflict about a window attribute.

The scenario of events represented by a chart is described after the chart. Each event (or interaction) in the chart is described in detail. I adopt the following convention. Each numbered step of the explanation corresponds to the identically numbered event in the chart. The steps that are explained in smaller italic text correspond to events located in the gray zone of the chart. These are not directly related to the focus events of the chart (located in the white zone of the chart). Note that events may be conditional in the sense that their execution depends on the successful completion of certain tasks within the execution of a given task.

General events charts

Event-trace charts (1-3) are developed for:

1. Activation of a DO and the deactivation of its OA after the execution of an assigned task.
2. Task execution by a leaf OA (the last node in a decomposed task).
3. Conflict handling among two leaf OAs of the same DO over shared attribute.

Domain specific events charts

Event-trace charts (4-7) are developed for four distinct applications.

1. Execution of a material cost evaluation task by BFloor-OA (building floor), classified according to its sub-DO (each rooms material cost) is required.
2. Execution of daylighting evaluation task by a BFloor-DO.
3. Execution of a structural analysis task executed by Building-DO.
4. Execution of a conflict handling session among two Room-OAs[1] over a recommended reduction of a the glazing area attribute of a Window-DO (room cost vs. room daylighting levels).

1. Each OA is responsible for the one task for clarity (though both conflicted attribute values are of the same DO).

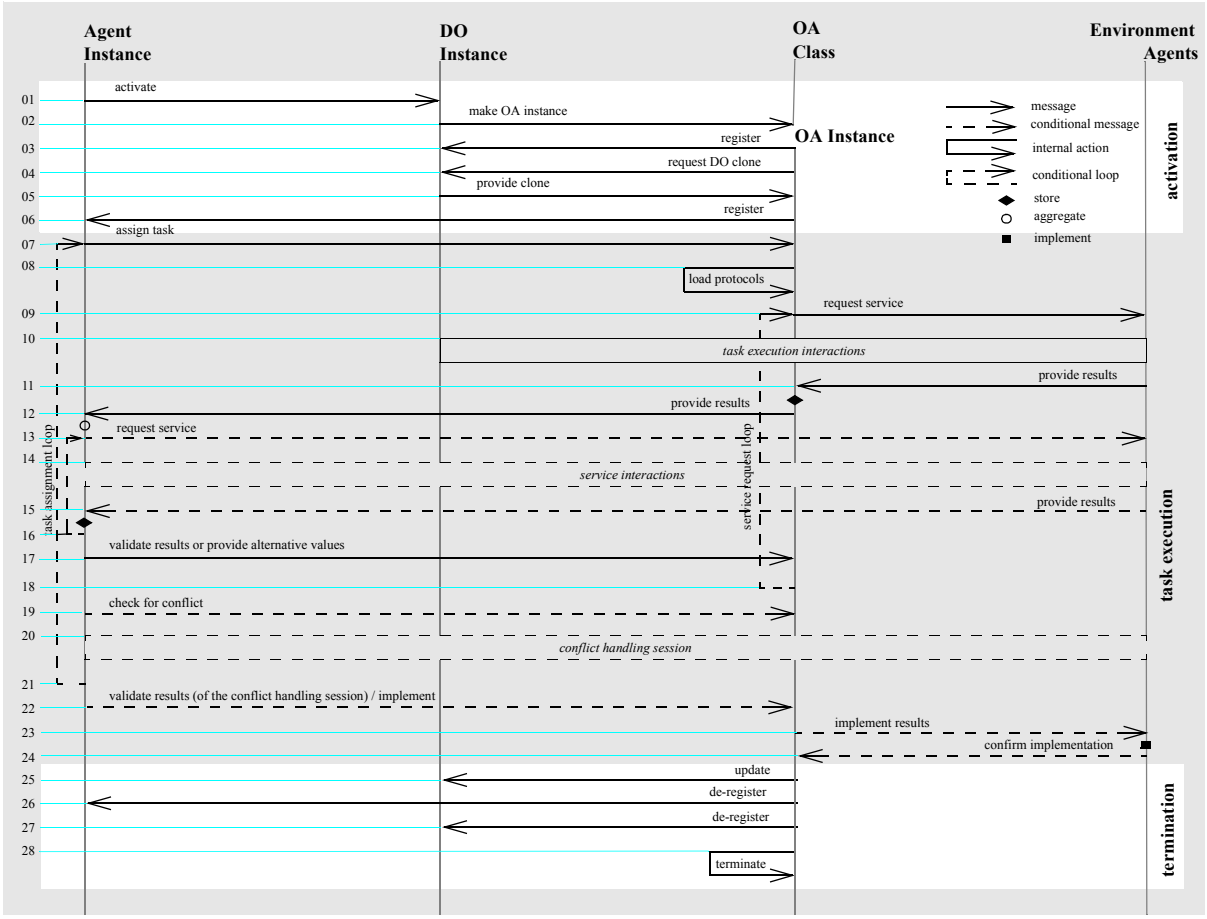


FIGURE 4.1.

Event-trace of the activation of a DO and the deactivation of an OA.

4.2 Chart 1. Activation of a DO/Deactivation of an OA

See Definitions A.3.10 and A.3.11. The event-trace is shown in Figure 4.1 The main steps are 1-6 and 25-28. Steps 7-24 deal with task execution and these are shown in gray.

1. An Agent sends an activation message to a DO.
2. The DO instantiates an OA of its DO type from the OA class.
   An OA of the same DO type is created.
3. The created OA registers itself as an OA of the DO.
4. The created OA requests a clone (a complete copy) of the DO.
5. The DO provides a clone of itself to the sub-OA.
6. The OA registers itself as sub-OA of the super-agent (of step 1).

7. *The super-agent assigns the created sub-OA a task.*
8. *The sub-OA loads the appropriate protocols for the task.*
9. *Interacting with the appropriate agents in the environment the sub-OA initiates a task execution session.*
10. *Task execution interactions (see Chart 2 steps 7-19).*
11. *The environment agent (of step 9) provides it task execution result to the sub-OA. The OA stores the results until validated.*
12. *The sub-OA provides its super-agent (of step 1) with the task execution results. Conditional (if aggregation is needed): The super-agent aggregates the results of its sub-OAs (if more than one sub-OA is executing related tasks).*
13. *Conditional (if aggregation is performed in step 12): The super-agent requests an environment agent (e.g., an EA) to evaluate the aggregation results.*
14. *Conditional (if step 13 is executed): Evaluation interactions to check the aggregation results provided in step 12 (see Chart 2 steps 23-31).*
15. *Conditional (if step 13 is executed): The environment agent (of step 13) provides its evaluation of the aggregation results to the super-agent. The super-agent stores the results until validated.*
16. *Conditional (if the results of step 12 or 15 are not satisfactory): The super-OA remanages either a task execution session in which case steps 7-16 are repeated, or an aggregation session in which case steps 13-16 are repeated.*
17. *The super-agent validates the results of the task execution, or: provides an alternative set of attribute values for task reassignment.*
18. *Conditional (if alternative values are provided in step 17): The OA reassigns the task to the environment agent to examine the alternative attribute values In such case steps 9-18 are repeated.*
19. *Interface option (to allow for conflict check): The super-agent requests a conflict check (with other interested DOs or EAs) before the attribute values of the sub-OA are modified.*
20. *Conditional (if step 19 is executed): Conflict handling interactions, the super-agent manages a conflict handling session regarding the targeted attribute values (see Chart 3 steps 20-45).*
21. *Conditional (if the results provided in step 20 is not satisfactory): The super-agent reassigns the sub-OA with a modified task.*
22. *Conditional (if step 20 is executed): The super-agent validates the results (of step 20), and: Interface option (to allow for implementation): The super-agent requests the implementation of the validated results (if attributes values of the sub-OA are recommended for modification).*

23. Conditional (if requested in step 22): The sub-OA interacts with the appropriate agents in the environment (e.g., CAD-agent) to implement the results. The environment agent carries the implementation of the results.

24. Conditional (if step 23 is executed): The environment agent which carried the implementation confirms its execution.

25. The sub-OA updates its DO information (provided that no other OA of the same DO is performing tasks. If more than an OA is to update the same information of the DO an update management session is required. Such a session may include conflict handling among the different OAs involved).

26. The sub-OA de-registers itself as a sub-agent of the super-agent.

27. The OA de-registers itself as an OA of the DO.

28. The OA terminates itself.

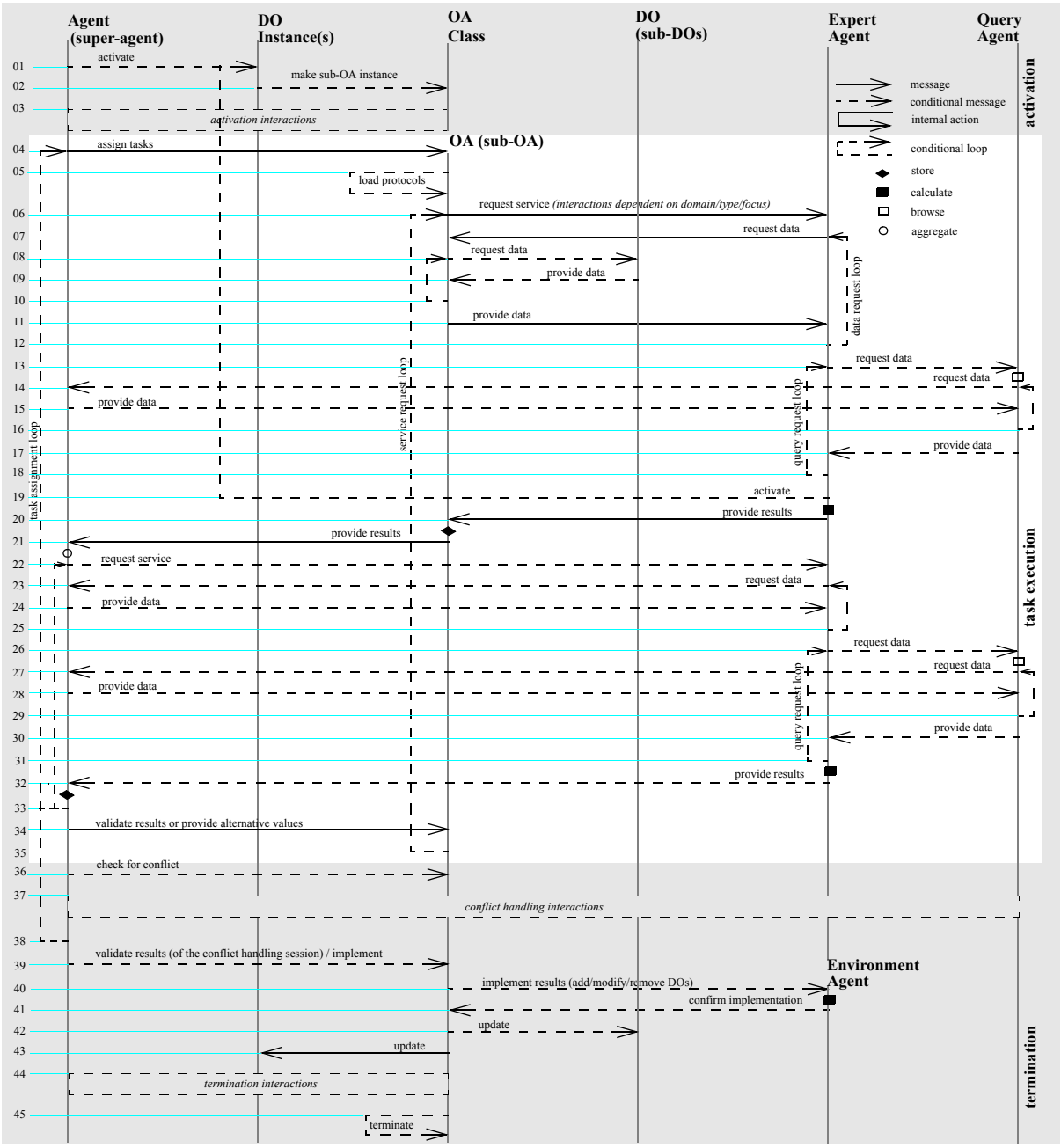


FIGURE 4.2.

Event-trace of task execution by a leaf-OA (where no further task decomposition is applicable).

4.3 Chart 2. Task Execution

See Section A.4 for relevant definitions. The event-trace is shown in Figure 4.2. The main steps are 4-35; all other steps (1-3 and 36-45) deal with necessary task management activities such as agent activation, result validation and so on.

1. *Conditional (if the targeted DO is not activated): An agent sends an activation message to a DO The agent is activating a DO to assign it a task.*
2. *Conditional (if step 1 is executed): The DO instantiates an OA of its DO type from the OA class.*
   An OA of the same DO type is created.
3. *Conditional (if step 2 is executed): Activation interactions (see Chart 1, steps 3-6).*
4. The super-agent assigns a task to the created OA (a sub-OA in the task dependent hierarchy).
5. Conditional (if the appropriate protocols are not loaded): The sub-OA loads the appropriate protocols for the assigned task.
6. Interacting with the appropriate agent in the environment (e.g., an EA) the sub-OA manages a task execution session.
7. The EA requests information from the sub-OA (e.g., geometric and non-geometric information of the sub-OA).
8. Conditional (when information of a sub-DO is needed and the DO is not activated and the activation of such sub-DO is not necessary): The sub-OA requests information from its sub-DO(s) in the hierarchy (if a DO is activated the information must be requested from its OA).
9. Conditional (if step 8 is executed): The sub-DOs provide the information requested to the sub-OA (as available).
10. Conditional (if the information provided in step 9 is not sufficient): The sub-OA requests additional information from its sub-DOs.
11. The sub-OA provides the applicable information to the EA.
12. Conditional (if the information provided in step 11 is not sufficient): The EA requests additional information from the sub-OA.
13. Conditional (if additional information is needed to execute the task): The EA requests information from a query-agent (e.g., prototypical information). The query-agent browses through the available data-base(s).
14. Conditional (if the query-agent was not able to locate the requested information of step 13). The query-agent requests information from the super-agent (or from the DA).
15. Conditional (if step 14 is executed): The super-agent provides the EA with the requested information.
16. Conditional (if the information provided in step 15 is not sufficient): The query-agent requests additional information from the super-agent (or the DA).

17. Conditional (if step is 13 executed): The query-agent provides the EA with the applicable information (as available).
18. Conditional (if the information provided in step 17 is not sufficient): The EA requests additional information from the query-agent In such case steps 13-18 are repeated.
19. Conditional (if the targeted DO is not activated): The EA sends an activation message to another DO(s) if the execution of the task in hand is dependent on other task execution results to be carried by the activated DO(s). It should be noted that such dependencies need to be monitored and controlled by the DA to avoid infinite loops of activations and task assignments. The DA confirmation should be obtained when such activations is requested.
    If the information needed is collected the EA executes the assigned task (of step 6).
20. The EA provides the sub-OA with the task execution results.
    The sub-OA stores the task execution results until validated.
21. The sub-OA provides the super-agent (of step 1) with the task execution results.
    Conditional (if aggregation is needed): The super-agent aggregates the results of its sub-OAs.
22. Conditional (if aggregation is performed in step 21): The super-agent requests an environment agent (e.g., an EA) to evaluate the aggregation results.
23. The EA requests information from the super-agent.
24. The super-agent provides the applicable information to the EA.
25. Conditional (if the information provided in step 24 is not sufficient): The EA requests additional (or missing) information from the super-agent. In such case steps 23-25 are repeated.
26. Conditional (if the results of step 24 are not satisfactory or if prototypical information is required): The EA sends a query request to the query-agent to obtain either prototypical information or any additional information needed to perform the task. The query-agent browses through the environment (or DOs) data-bases to obtain the requested information.
27. Conditional (if the information requested in step 26 is not found in the environment): The query-agent requests the super-agent or the DA to provide the information requested in step 26.

28. Conditional (if step 27 is executed): The super-agent or the DA provides the query-agent with the requested information.
29. Conditional (if the information provided in step 28 is not sufficient): The query-agent requests the super-agent or the DA to provide additional information. In such case steps 27-29 are repeated.
30. Conditional (if step 26 is executed): The query-agent provides the EA with applicable information (as available).
31. Conditional (if the information provided in step 30 is not sufficient): The EA sends another query request for additional information. In such case steps 26-31 are repeated.
    If the information provided to the EA is sufficient the EA evaluates the aggregation results provided by the super-agent in step 22.
32. Conditional (if step 24 is executed): The EA provides its evaluation of the aggregation results to the super-agent.
    The super-agent stores the results until validated.
33. Conditional (if the results provided in either step 21 or 32 are not satisfactory): The super-OA remanages either a task execution session with a modified task in which case steps 4-33 are repeated, or an aggregation evaluation session in which case steps 22-33 are repeated.
34. The super-agent validates the results (of step 21 or 32), or:
    provides the sub-OA with alternative attribute values for task reassignment (or for another aggregation evaluation session).
35. Conditional (if alternative values are provided by the super-agent in step 34): The sub-OA remanages a task execution session to examine the alternative values.

36. Interface option (to allow for conflict check): The super-agent requests a conflict check (with other interested DOs or EAs) before the attribute values of the sub-OA are modified.

37. Conditional (if step 36 is executed): Conflict handling interactions, the super-agent manages a conflict handling session regarding the targeted attribute values (see Chart 3 steps 20-45).

38. Conditional (if the results of the conflict handling session provided in step 37 are not satisfactory): The super-agent reassigns a modified task to the same sub-OA (of step Figure 4).

39. *Conditional (if step 36 is executed): The super-agent validates the results (of step 37), and:*
    Interface option (to allow for implementation): The super-agent requests the implementation of the validated results (if the attribute values of the sub-OA are recommended for modification).
40. *Conditional (if requested in step 39): The sub-OA interacts with the appropriate agents in the environment (e.g., CAD-agent) to implement the results.*
    The environment agent carries the implementation of the results.
41. *Conditional (if step 40 is executed): The environment agent which carried the implementation confirms its execution.*
42. *The sub-OA updates the information of its sub-DOs effected by the implementation.*
43. *The sub-OA updates its DO information (if the DO is activated the update must be conducted through its OA).*
44. *Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).*
45. *Conditional (if step 44 is executed): The OA terminates itself.*

4.4 Chart 3. Conflict Handling

See Section A.6 for relevant definitions. The event-trace is shown in Figure 4.3. The important steps are 20-45. Each attribute of a DO has an attached list of interested DOs, EAs and other attributes (see explanation in Section 3.1 & details in Section 5.2.2). This list is compiled when the DO class is created. Within a session, the list can be modified (after the DO has been instantiated). I recap that an **interested DO** is a DO with at least one attribute registered in the list, and an **interested EA** is an EA registered for at least one attribute in the list. This list is provided by a DO instance upon request and serves as a reminder of which DO attribute or EA may be affected by any modification to the particular attribute value. Such modifications may constitute conflicts with some members of the list. The order of the list members is insignificant, however, when a DA selects a set of members of a list for conflict check it should be a sorted set. The DA sorts the activation of members of the selected set according to a personalized criteria for conflicts that are considered most critical. When a DA selects a set to proceed with a conflict check:

- an OA should be created for each DO (or attribute of the same DO) in the set. For instance, if a depth attribute of a Beam-DO is selected from the $interest_{list}$ of a width attribute of a Room-DO a Beam-OA should be created for the conflict handling session (about the change in the Room-DO width). The activated Beam-OA may perform a stress analysis task to examine the beam strength in respect to the new width of the Room-DO.
- an OA of the same DO should be created for each selected EA in the set. For instance, if a daylighting-EA is selected from the $interest_{list}$ of a glazing-area attribute of a Window-DO, a Window-OA should be created to represent the daylighting-EA in the conflict handling session. The activated Window-OA may perform a daylighting evaluation task to examine the new glazing area of the Window-DO.

The OA approach can accommodate a variety of mechanisms and guide lines for conflict handling. However, the mechanisms and guidelines adopted within the scope of this work are intended to demonstrate the potential benefits of the OA approach.

Conflict handling types

A task dependent hierarchy consists of multiple levels of DOs from the DA down to the leaf DOs. A conflict handling session may involve OAs in the same level or in different levels of the same hierarchy or in multiple hierarchies. The DA

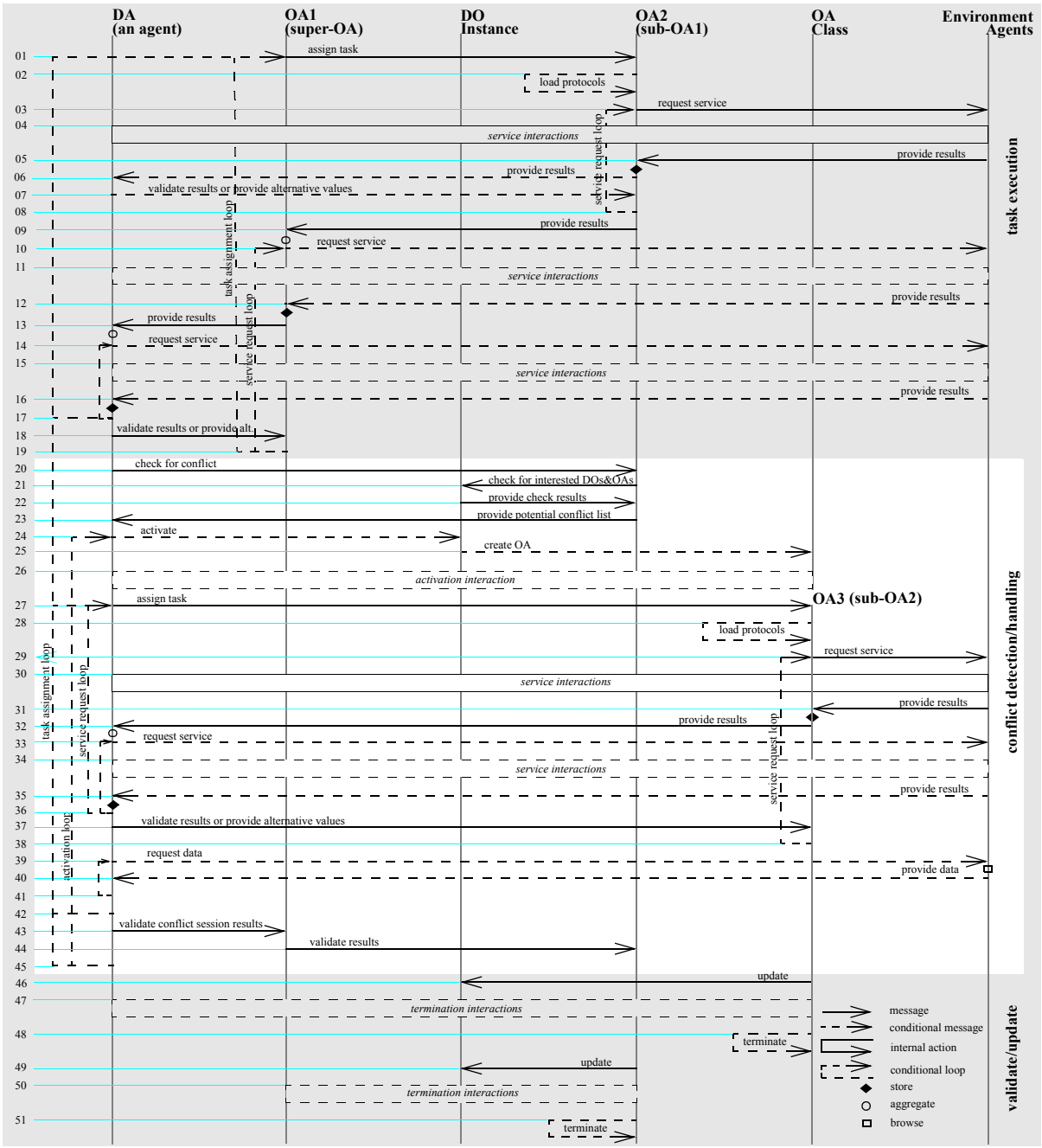


FIGURE 4.3.

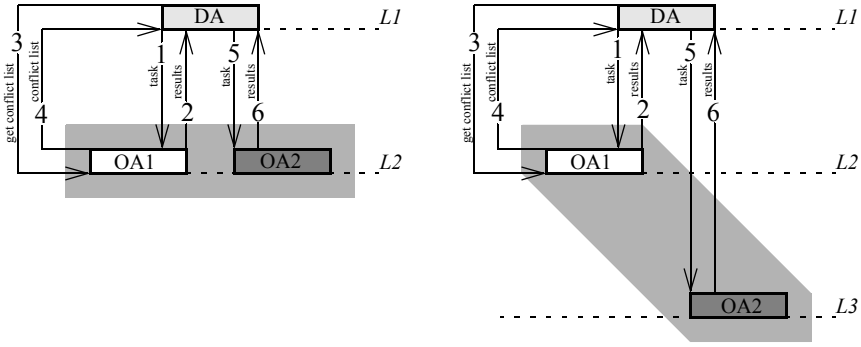
Event-trace of conflict handling among two leaf-OAs.

may inquire about an $interest_{list}$ of an OA that is assigned tasks either by the DA or by another agent. Note that the OA that provides an $interest_{list}$ is considered the **conflict focus** (see Def. A.6.11)**.** The DA selects a sub-set of the $interest_{list}$ to check for conflict, the members of that sub-set are not necessarily from the same

level at the hierarchy nor from the same hierarchy. The two OAs involved in the conflict constitute the **conflict zone**. A **direct conflict handling** session involves two OAs one of which is the conflict focus. An **indirect conflict handling** session involves two OAs neither of which is the conflict focus. Accordingly, the conflict zone does not necessarily include the conflict focus. In other words, two OAs may have a conflict over an attribute value of a a third OA. In such a case the two OAs (who constitute the conflict zone) are involved in an indirect conflict about a third OA attribute (who is the conflict focus). The tasks that triggers the conflict may be assigned by the OAs or the DA, however, the DA may need to interact with OAs which were not directly assigned tasks by DA (the DA interacts to provide alternative values for iterative evaluations and to validate the conflict results). Figure 4.4 shows various cases of direct and indirect conflict handling among OAs in the same or in different levels of a hierarchy. The various conflict cases in the figure illustrate that conflict types may require different patterns of interaction between the DA and the OAs involved. In particular, the figure shows how an OA environment can help the designer be aware of the most relevant DOs to a conflict. Chart 3 is an example of a direct conflict handling session for two OAs in the same level (case C of Figure 4.4). Chart 7 is an example of an indirect conflict handling session for two OAs in the same level (case E of Figure 4.4). The two Charts demonstrate how the conflict type affects the pattern of DA interaction with the OAs of the task zone and task focus.

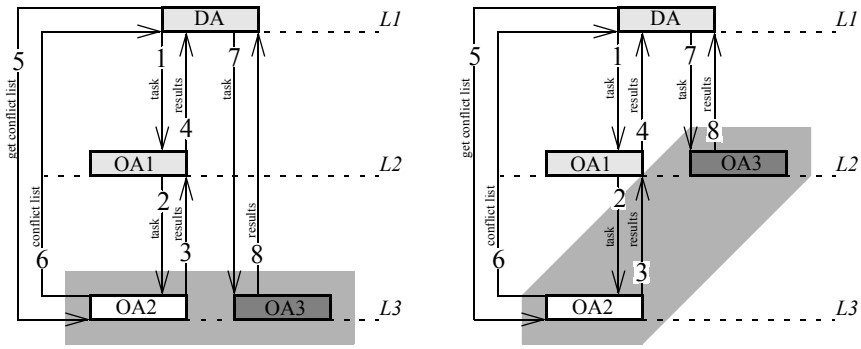
Details of the interactions

1. *A super-OA assigns a task to a sub-OA (sub-OA1 in this chart).*
2. *Conditional (if the appropriate protocols are not loaded): Sub-OA1 loads the appropriate protocols for the assigned task.*
3. *Interacting with the appropriate agent in the environment (e.g., an EA) sub-OA1 initiates a task execution session.*
4. *Task execution interactions (see Chart 2 steps 7-19).*
5. *The environment agent provides its task execution results to sub-OA1. Sub-OA1 stores the results until validated.*
6. *Interface option (to allow the DA to validate task results not assigned directly by him/her): Sub-OA1 provides the DA with the task execution results.*
7. *Conditional (if step 6 is executed): The DA validates the results of sub-OA1, or: provides an alternative set of attribute values for task reassignment.*
8. *Conditional (if alternative values are provided in step7): Sub-OA1 reassigns the task to the EA to examine the alternative values. In such case steps 3-8 are repeated.*



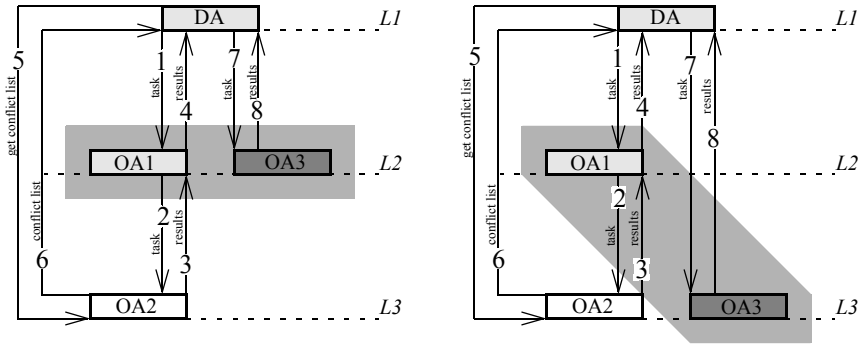
same level different levels

(A & B) direct conflict handling for an OA performing a task assigned by a DA



same level different levels

(C & D) direct conflict handling for an OA performing a task not assigned by a DA



same level different levels

(E & F) indirect conflict handling for an OA performing a task not assigned by a DA

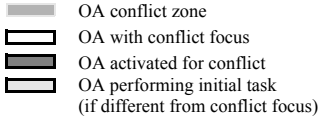


FIGURE 4.4.

Conflict handling cases.

9. *Sub-OA1 provides its task execution results to the super-OA.*
   Conditional (if aggregation is needed): The super-OA aggregates the results of its sub-OAs.
10. *Conditional (if aggregation is performed in step 9): The super-agent requests an environment agent (e.g., an EA) to evaluate the aggregation results.*
11. *Conditional (if step 10 is executed): Evaluation interactions to check the aggregation results provided in step 10 (see Chart 2 steps 23-31).*
12. *Conditional (if step 11 is executed): The environment agent (of step 10) provides its evaluation of the aggregation results to the super-agent.*
    The super-agent stores the results until validated.
13. *The super-OA provides the DA with results (of step 9 or 12) for validation.*
    Conditional (if aggregation is needed): The DA aggregates the provided results.
14. *Conditional (if aggregation is performed in step 13): The DA requests an environment agent (e.g., an EA) to evaluate the aggregation results.*
15. *Conditional (if step 14 is executed): Evaluation interactions to check the aggregation results provided in step 14 (see Chart 2 steps 23-31).*
16. *Conditional (if step 15 is executed): The environment agent (of step 14) provides its evaluation of the aggregation results to the DA.*
    The DA stores the results until validated.
17. *Conditional (if the results provided in either step 13 or step 16 are not satisfactory): The DA/super-agent remanages either a task execution session in which case steps 1-17 are repeated, or an aggregation session in which case steps 14-17 are repeated.*
18. *The DA validates the results (of step 13), or:*
    provides the super-OA with alternative attribute values for task reassignment or to run another aggregation session for the sub-results provided in step 10.
19. *Conditional (if alternative values are provided by the DA in step 18): The super-OA remanages either a task execution session to examine the alternative values, in which case steps 1-19 are repeated, or an aggregation session, in which case steps 10-19 are repeated.*
20. Interface option (to allow for conflict check): The DA requests a conflict check (with other interested DOs or EAs) before the attribute values of the sub-OA1 are modified.
21. Sub-OA1 checks with its DO for a list of DOs and OAs with interest in the targeted attribute values (i.e., which is subject to modification according to the task execution results). The interested DOs or EAs are potential candidates for conflict over the recommended attributes values.
22. The DO provides a list of DOs and EAs interested in the targeted attribute values.
23. Sub-OA1 provides the DA with the list of interested DOs and EAs.

24. Conditional (if the list contains one or more DO or OA): The DA selects a set of DOs and EAs from the list (according to the DA's criteria for most critical conflicts) to activate for conflict check.
    Conditional (if the targeted DO is not activated): The DA sends an activation message to the first DO on the selected set.
25. Conditional (if step 24 is executed): The DO instantiates an OA of its DO type from the OA class.
    Sub-OA2 is created.
26. Conditional (if step 25 is executed): Activation interactions (see Chart 1 steps 3-6).
27. The DA assigns an evaluation task to examine the recommended attribute values (which is the focus of the conflict handling session) with respect to the sub-OA2 (activated in step 24).
28. Conditional (if the appropriate protocols are not loaded): sub-OA2 loads the appropriate protocols for the assigned task.
29. Interacting with the appropriate agent in the environment (e.g., an EA) sub-OA2 initiates a task execution session.
30. Task execution interactions (see Chart 2 steps 7-19).
31. The EA (of step 29) provides sub-OA2 with the task execution results.
    Sub-OA2 stores the results until validated.
32. Sub-OA2 provides its task execution results to the DA.
    Conditional (if aggregation is needed): The DA aggregates the sub-results provided in step 31.
33. Conditional (if aggregation is performed in step 32): The DA requests an environment agent (e.g., an EA) to evaluate the aggregation results.
34. Conditional (if step 33 is executed): Evaluation interactions to check the aggregation results provided in step 33 (see Chart 2 steps 23-31).
35. Conditional (if step 34 is executed): The environment agent (of step 33) provides its evaluation of the aggregation results to the DA.
    The DA stores the results until validated.
36. Conditional (if the results provided in either step 32 or step 35 are not satisfactory): The DA remanages either a task execution session in which case steps 27-36 are repeated, or an aggregation session in which case steps 33-36 are repeated, or:

37. The DA validates the results (of step 32), or:
    provides alternative attribute values for task reassignment.
38. Conditional (if alternative values are provided by the DA in step 37): Sub-OA2 runs another session to examine the alternative values. In such case steps 29-38 are repeated.
39. Conditional (if the results provided in step 32 are not satisfactory): The DA sends a query request to obtain information from the environment agents (e.g., a query-agent) to help in reassigning the task to either sub-OA1 or to sub-OA2 with alternative attribute values.
40. Conditional (if step 39 is executed): The environment agents provide the DA with the applicable information (as available).
41. Conditional (if the information provided in step 40 is not sufficient): The DA sends a modified query request to the environment agents. In such case steps 39-41 are repeated.
42. Conditional (if the results of step 32 are not satisfactory): The DA reassigns another evaluation task to either sub-OA1 in which case steps 1-42 are repeated, or to sub-OA2 in which case steps 27-42 are repeated.
43. The DA validates the conflict handling session results (started in step 20).
44. The super-OA validates the results of sub-OA1 (provided in step 9).
45. Conditional (if there are more than one DO or EA in the set selected in step 24): The DA either activates the next DO in the set, in which case steps 27-45 are repeated, or assigns another evaluation task to sub-OA1 in relation to the next EA in the set, in which case steps 1-45 are repeated.

46. Sub-OA2 updates its DO information (after the implementation of the task results, see Chart 2 steps 39-41).

47. Conditional (if no other task is to be executed): Termination interactions (see Chart 1. steps 26 & 27).

48. Conditional (if step 47 is executed): Sub-OA2 terminates itself.

49. Sub-OA1 updates its DO information (after the implementation of the task results, see Chart 2 steps 39-41).

50. Conditional (if no other task is to be executed): Termination interactions (see Chart 1. steps 26 & 27).

51. Conditional (if step 50 is executed): Sub-OA1 terminates itself.

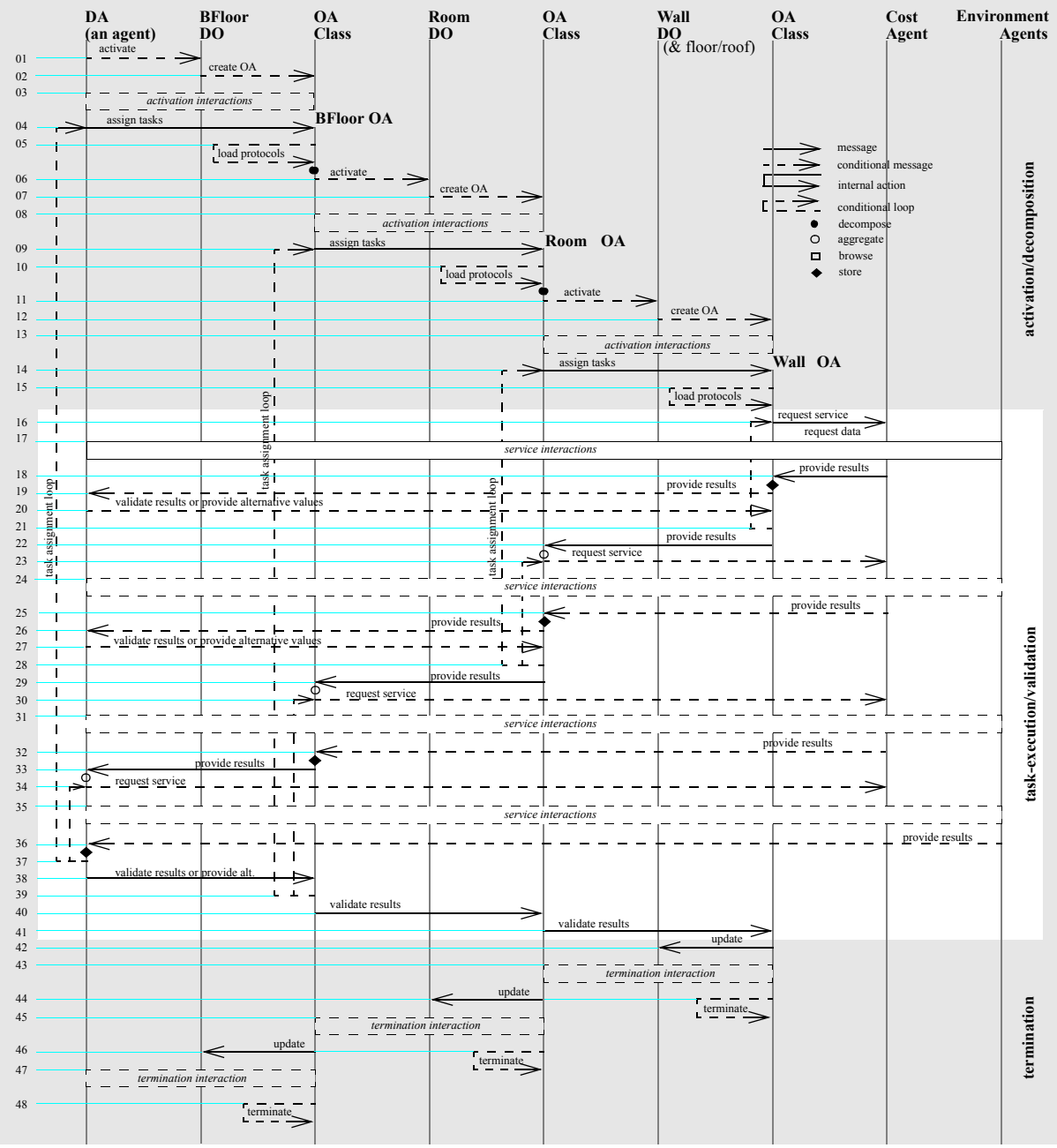


FIGURE 4.5.

Event trace of a painting cost evaluation task executed by a BFloor-OA (classified per Room-DO).

4.5 Chart 4. Cost Evaluation Task (Classified per Room-DO)

This is the first of four applications illustrating the OA-based design environment. The event-trace chart for cost-evaluation is shown in Figure 4.5. For this task I first describe the assigned task, identify the main players, the

major events, the expect results before giving a description of the detailed scenario of interactions.

The assigned task The DA is interested in evaluating the cost of painting a BFloor-DO. The evaluation should include the total painting cost of the BFloor-DO and a cost classification per Room-DO. This includes the total cost of each individual Room-DOs and an itemized cost of its activated sub-DOs such as Wall-DOs, Ceiling-DOs (and any DO that need to be painted).

The main players There are three main players in the scenario for this task: a DA, a BFloor-OA and a cost-EA (and the geometry-EA as a secondary player).

The major events There are three main events which are listed below.

- The DA activates the BFloor-DO and assigns it a classified cost evaluation task, which, in turn, triggers a chain of activation by the BFloor-OA down the hierarchy to its Room-DOs to all leaf-DOs (i.e., the last nodes in the task hierarchy such as Walls, Partitions and Ceilings).
- Interacting with the cost-EA, each activated OA runs a cost evaluation session regarding its own state. The sequence of evaluation sessions starts from the leaf OAs where the results are provided to their super-agents up to the DA level. As a leaf node, a Wall-DO interacts with the cost-EA to calculate its painting cost. The cost-EA requests the total solid area of the Wall-DA. Accordingly, the Wall-DO interacts with the geometry-EA to calculate the required area and then provides the cost-EA with the requested painting cost. Each super-agent aggregates the results provided by its sub-OAs. Note: There may exist many alternatives of how the cost may be calculated. For instance it is possible to aggregate the areas and calculate the total cost at once, however, calculating the cost of each Wall-DO (or Ceiling-DO, etc.) provides more value to the designer.
- The OAs update the information of its DOs and terminates itself.

The following two are additional optional results.

- The DA triggers a conflict handling session.
- The DA requests the implementation of new attribute values.

The expected results There are two results that can be expected from this task and two additional optional result.

1. For each evaluation session[2] the assigned BFloor-OA provides the DA with information[3] that may contain at least one of the following:
   - painting cost of the BFloor-DO, and the classified cost as assigned (see the assigned task);
   - prototypical cost of each of the DOs above;
   - a warning for each DO that exceeds the specified prototypical cost.
2. The DA examines either new DO attribute values for DOs in the task dependent hierarchy or new cost constraints.
3. (Optional) The DA examines new DO attribute values for DOs interested in the attribute values of the DOs in the task dependent hierarchy. This may occur as a result of conflict handling sessions (if executed).
4. (Optional) Implementation of examined DO attribute values and updating of the DO relations.

Details of the interactions

The main steps are 16-41.

1. *Conditional (if the targeted BFloor-DO is not activated): A DA sends an activation message to a BFloor-DO.*
2. *Conditional (if step 1 is executed): The BFloor-DO instantiates a BFloor-OA. A BFloor-OA is created.*
3. *Conditional (if step 2 is executed): Activation interactions (see Chart 1 steps 3-6).*
4. *The DA assigns the painting cost evaluation task to the created BFloor-OA.*
5. *Conditional (if the appropriate protocols are not loaded): The BFloor-OA loads the cost evaluation protocols.*
   The BFloor-OA uses the protocols to decompose the task among potential sub-DOs (linked to the BFloor-OA in a hierarchy)
6. *Conditional (if the targeted sub-DOs are not activated): The BFloor-OA sends activation messages to all qualified sub-DOs (Room-DOs in this case).*

2. An evaluation session is defined as "an evaluation by the EA (namely the cost-EA in this chart) of one set of DO attribute values with respect to either saved prototypical values, predefined constrains, or requirements".
3. The form by which this information is presented (e.g., lists, charts, diagrams) is implementation dependent. The DA should be able to interactively and graphically modify such information.

7. *Conditional (if step 6 is executed): The Room-DO class instantiates a Room-OA for each activated Room-DO.*
   The created Room-OAs are sub-OAs to the BFloor-OA.
8. *Conditional (if step 7 is executed): Activation interactions (see Chart 1 steps 3-6).*
9. *The BFloor-OA assigns painting cost evaluation sub-tasks to the created Room-OAs.*
10. *Conditional (if the appropriate protocols are not loaded): Each Room-OA loads the cost evaluation protocols.*
    Each Room-OA uses the protocols to decompose the task among potential sub-DOs (linked to each Room-OA within the task hierarchy). The protocols identifies the shared-DOs in the hierarchy (such as walls, floors etc.) Other environment agents (e.g., geometry-agent) identifies the portion of any shared DO to be considered as a sub-DO of its room-super-agent[4].
11. *Conditional (if the targeted DOs are not activated): Each Room-OA sends activation messages to all its qualified sub-DOs. For the shared DOs, the activation may only occur by one room-super-agent, the rest of the room-super-agents (sharing the same DO) may only assign tasks to the created sub-OA.*
12. *Conditional (if step 11 is executed): The Wall-DO/Ceiling-DO/Floor-DO classes (and any other sub-DO in the hierarchy that requires painting) instantiate Wall-OA/ Ceiling-OA/Floor-OA etc. for each activated DO.*
    Each created OA is a sub-OA to its Room-OA (as a super OA).
13. *Conditional (if step 12 is executed): Activation interactions (see Chart 1. steps 3-6).*
14. *The Room-OA assigns painting cost evaluation sub-tasks to the created sub-OAs.*
15. *Conditional (if the appropriate protocols are not loaded) Each sub-OA loads the cost evaluation protocols. The decomposition process terminates when no other sub-DOs are found in the hierarchy. Therefore, the sub-OAs of step 12 are considered leafs in the decomposition tree.*
16. Interacting with the cost-EA each leaf OA initiates an evaluation task execution session.
17. Task execution interactions (see Chart 2 steps 7-19). This may include interactions among a geometry-EA and the each leaf-OA to provide the cost-OA with requested areas of each leaf-OA.
18. The cost-EA provides its cost evaluation results (as assigned; current cost, relation to prototypical cost etc.) to the each sub-OA.
    Each sub-OA stores its results until validated.

4. The recognition of the sub-parts depends on a large degree on the representation of DOs and on the capabilities of the agents that are concerned with interpreting geometric representations.

19. Interface option (to allow the DA to validate task results not assigned directly by him/her): Each sub-OA provides its cost evaluation results to the DA.
20. Conditional (if step 19 is executed): DA validates the sub-OAs cost evaluation results, or:
    provides the sub-OA with alternative attribute values for task reassignment.
21. Conditional (if alternative values are provided in step 20): The sub-OA reassigns the task to the cost-EA to examine the alternative values. In such case steps 16-21 are repeated.
22. Each sub-OA provides its cost evaluation results to its super-OA (Room-OA).
    Conditional (if more than one sub-OA provided sub-results to its super-OA): The Room-OA uses its domain protocols to aggregate the sub-results collected from its sub-OAs.
23. Conditional (if aggregation is performed in step 22): The Room-OA interacts with the cost-EA to evaluate its aggregation results, in respect to the design requirements and the prototypical values in the database.
24. Conditional (if step 23 is executed): Task execution interactions (see Chart 2 steps 23-31).
25. Conditional (if step 24 is executed): The cost-EA provides its aggregation evaluation results (as assigned; current painting cost, relation to prototypical cost etc.) to the Room-OA.
    The Room-OA stores the results until validated.
26. Interface option (to allow the DA to validate task results not assigned directly by him/her): Each Room-OA provides its cost evaluation results to the DA.
27. Conditional (if step 26 is executed): The DA validates the results of the cost evaluation of each Room-OA if satisfactory, or:
    provides the Room-OA with alternative attribute values for task reassignment.
28. Conditional (if alternative values are provided by the DA in step 27): The Room-OA either reassigns the task to the cost-EA to examine the alternative values, in which case steps 14-28 are repeated, or run another aggregation evaluation session, in which case steps 23-28 are repeated.
29. Each Room-OA provides its painting cost evaluation results to its super-OA (the BFloor-OA in this case).
    Conditional (if more than one Room-OA submitted sub-results to the BFloor-OA): The BFloor-OA uses its protocols to aggregate the sub-results collected from the Room-OAs.

30. Conditional (if aggregation is performed in step 29): The BFloor-OA interacts with the cost-EA to evaluate the aggregated results (in respect to the design requirements or the prototypical values in the database).
31. Conditional (if step 30 is executed): Task execution interactions (see Chart 2 steps 23-31).
32. Conditional (if step 31 is executed): The cost-EA provides its aggregation evaluation results (as assigned; current painting cost, relation to prototypical cost etc.) to the BFloor-OA.
    The BFloor-OA stores the results until validated.
33. The BFloor-OA provides its task execution results to the DA.
    Conditional (if more than one BFloor-OA provided sub-results to the DA): The DA aggregates the sub-results of the BFloor-OAs.
34. Conditional (if aggregation is performed in step 33): The DA interacts with the cost-EA to evaluate the aggregated results to evaluate the aggregated results.
35. Conditional (if step is 34 executed): Task execution interactions (see Chart 2 steps 23-31)
36. Conditional (if step 35 is executed): The cost-EA provides the DA with its aggregation evaluation results (as assigned; current cost, relation to prototypical cost etc.).
    The DA stores the results until validated.
37. Conditional (if the results of steps 33 or 36 are not satisfactory): The DA either reassigns the initial task (of step 4) to the BFloor-OA with alternative attribute values or requirements, in which case steps 4-37 are repeated, or run another aggregation evaluation session, in which case steps 34-37 are repeated.
38. The DA validates the results of the BFloor-OA (of step 33), or:
    provides alternative attribute values for task reassignment.
39. Conditional (if alternative values are provided in step 38): The BFloor-OA either reassigns the task to the cost-EA to examine the modified values, in which case steps 9-39 are repeated, or run another aggregation evaluation session, in which case steps 30-39 are repeated.
40. The BFloor-OA validates the results (of step 29) of each Room-OA.
41. Each Room-OA validates the results (of step 22) of its sub-OAs.

42. *Each sub-OA (of step 12) updates its DO information.*

43. *Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).*
    Conditional (if step 42 is executed): Each sub-OA terminates itself.

44. *Each Room-OA updates its DO information.*

45. *Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).*
    Conditional (if step 44 is executed): Each Room-OA terminates itself.

46. *The BFloor-OA updates its DO information.*

47. *Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).*

48. *Conditional (if step 47 is executed): The BFloor-OA terminates itself.*

4.6 Chart 5. Daylight Evaluation Task

The event-trace chart is shown in Figure 4.6. As before I describe the assigned task, identify the main players, major events and expected results.

The assigned task

The DA is interested in evaluating the daylighting of a BFloor-DO. The evaluation should include daylighting levels classified according to each Room-DO of the BFloor-DO during a specified range of hours of the day.

The evaluation should also include the daylighting levels of each individual opening within each Room-DO. The bfloor-evaluation should include statistical information about the number of Room-DOs within the BFloor-DO satisfying the daylighting constrains and prototypical values.

The main players

The scenario of this chart involves three main players; a DA, a BFloor-OA and a daylighting-EA.

The major events

- The DA activates the BFloor-DO and assigns it a daylighting evaluation task, which, in turn, triggers a chain of activation by the BFloor-OA down the hierarchy to its Room-DOs (which are the last nodes in the daylighting task dependent hierarchy).
- Interacting with the daylighting-EA, each activated OA runs a daylighting evaluation session regarding its own state. The sequence of evaluation sessions starts from the leaf OAs where the results are provided to their super-agents. Each super-agent aggregates (when needed) the results provided by its sub-OAs, and so forth up to the DA level.
- The OAs update the information of its DOs and terminates itself.

The following two events are optional.

- The DA triggers a conflict handling session.
- The DA requests the implementation of new attribute values.

The expected results

1. For each evaluation session the assigned BFloor-OA provides the DA with information that may contain at least one of the following:
   - daylighting levels of each Room-DO (see the assigned task);
   - prototypical daylighting levels of each of the Room-DO types (e.g., living room, bedroom);

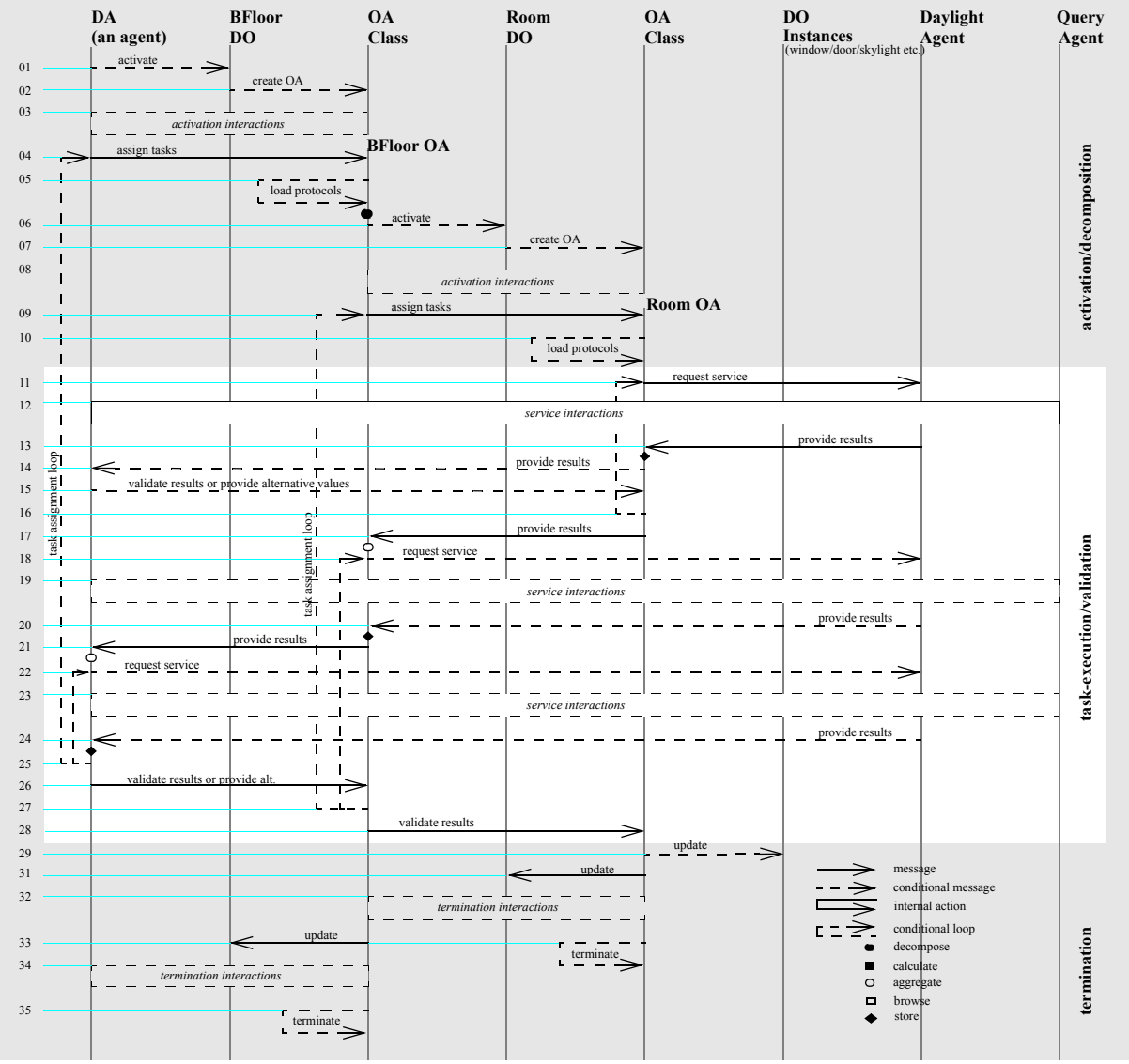


FIGURE 4.6.

Event-trace of a daylighting evaluation task for a BFloor-OA.

- a warning for each DO that is below the specified or prototypical daylighting levels.

2. The DA examines either new DO attribute values for DOs in the task dependent hierarchy or new daylighting constraints.
3. (Optional) The DA examines new DO attribute values for DOs interested in the attribute values of the DOs in the task dependent hierarchy. This may occur as a result of conflict handling sessions (if executed).

4. (Optional) Implementation of examined DO attribute values and updating of the DO relations.

Details of the interactions

The main steps are 11-28.

1. *Conditional (if the BFloor-DO is not activated): A DA sends an activation message to a BFloor-DO.*
2. *Conditional (if step 1 is executed): The BFloor-DO instantiates a BFloor-OA.*
   A BFloor-OA is created.
3. *Conditional (if step 2 is executed): Activation interactions (see Chart 1 steps 3-6).*
4. *The DA assigns a daylighting evaluation task to the created BFloor-OA.*
5. *Conditional (if the appropriate protocols are not loaded): The BFloor-OA loads the daylighting evaluation protocols.*
   The BFloor-OA uses the protocols to decompose the task among potential sub-DOs (linked to the BFloor-OA in a task dependent hierarchy).
6. *Conditional (if the targeted DOs are not activated): The BFloor-OA sends an activation message to its qualified room-sub-DOs.*
7. *Conditional (if step 6 is executed): The Room-DO class instantiates Room-OAs for each Room-DO.*
   The created Room-OAs are sub-OAs to the BFloor-OA.
8. *Conditional (if step 7 is executed): Activation interactions (see Chart 1 steps 3-6).*
9. *The BFloor-OA assigns daylighting evaluation sub-tasks to the created Room-OAs.*
10. *Conditional (if the appropriate protocols are not loaded): Each Room-OA loads the daylighting evaluation protocols.*

11. Interacting with the daylighting-EA each Room-OA initiates a daylighting evaluation task session.
12. Task execution interactions (see Chart 2 steps 7-19).
    Note: The daylighting-EA activates any adjacent Room-DO (see Chart 2 step 19) which shares one or more internal openings with any Room-OA being evaluated. The daylighting-EA evaluates the daylighting levels in such rooms in order to obtain the lighting levels reflected on the shared openings of the Room-OAs being evaluated. It should be noted that infinite loops of Room-DO activations and dependencies may occur if the depth and number of adjacent Room-DOs to effect the lighting levels on a shared opening are not controlled.

13. The daylighting-EA provides its daylighting evaluation results (as assigned; daylighting levels, relation to prototypical values etc.) to each Room-OA requested service.
    Each Room-OA stores its results until validated.
14. Interface option (to allow the DA to validate task results not assigned directly by him/her): Each Room-OA provides its daylighting evaluation results to the DA.
15. Conditional (if step 14 is executed): The DA validates the Room-OAs daylighting evaluation results if satisfactory, or:
    provides alternative attribute values for task reassignment.
16. Conditional (if alternative values are provided in step 15): Any Room-OA may reassign the task to the daylighting-EA to examine the alternative values. In such case steps 11-16 are repeated.
17. Each Room-OA provides its daylighting evaluation results to the BFloor-OA.
    Conditional (if aggregation is needed): The BFloor-OA uses it protocols to aggregate the sub-results collected from its Room-OAs.
    Note: The nature of aggregation of the BFloor-OA daylighting sub-results is different from that of the cost evaluation sub-results (as in Chart 4) No additional calculations by the daylighting-EA are required, instead a listing of all daylighting levels in each of its Room-DO may be sufficient (possibly done by the BFloor-OA itself).
18. Conditional (if aggregation is performed in step 21): The BFloor-OA requests the daylighting-EA to evaluate the aggregated results.
19. Conditional (if step is 18 executed): Task execution interactions (see Chart 2 steps 23-31)
20. Conditional (if step 19 is executed): The daylighting-EA provides its aggregation evaluation results (as assigned; current cost, relation to prototypical cost etc.) to the BFloor-OA.
    The BFloor-OA stores the results until validated.
21. The BFloor-OA provides its evaluation results to the DA.
    Conditional (if aggregation is needed): The DA aggregates the sub-results (if more than one BFloor-OA were executing the daylighting evaluation task simultaneously).
22. Conditional (if aggregation is performed in step 21): The DA requests the daylighting-EA to evaluate the aggregated results.

23. Conditional (if step is 22 executed): Task execution interactions (see Chart 2 steps 23-31)
24. Conditional (if step 23 is executed): The daylighting-EA provides its aggregation evaluation results (as assigned; current cost, relation to prototypical cost etc.) to the DA.
    The DA stores the results until validated.
25. Conditional (if the results provided in step 21 or 23 are not satisfactory): The DA either reassigns the initial task to the BFloor-OA with modified values or requirements, in which case steps 4-25 are repeated, or runs another aggregation evaluation session, in which case steps 22-25 are repeated.
26. The DA validates the results (of step 25) of the BFloor-OA, or:
    provides alternative attribute values for task reassignment.
27. Conditional (if alternative values are provided in step 26): The BFloor-OA either reassigns the daylighting task to the daylighting-EA to examine the alternative values in which case steps 9-27 are repeated, or runs another aggregation evaluation session in which case steps 18-27 are repeated.
28. The BFloor-OA validates the results (of step 17) of each Room-OA.
29. *Conditional (if attribute values of related DOs are to be modified): Each Room-OA updates the information of its sub-DOs (e.g., opening).*
30. *Each Room-OA (of step 7) updates its DO information.*
31. *Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).*
    Each Room-OA terminates itself.
32. *The BFloor-OA updates its DO information.*
33. *Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).*
34. *Conditional (if step 33 is executed): The BFloor-OA terminates itself.*

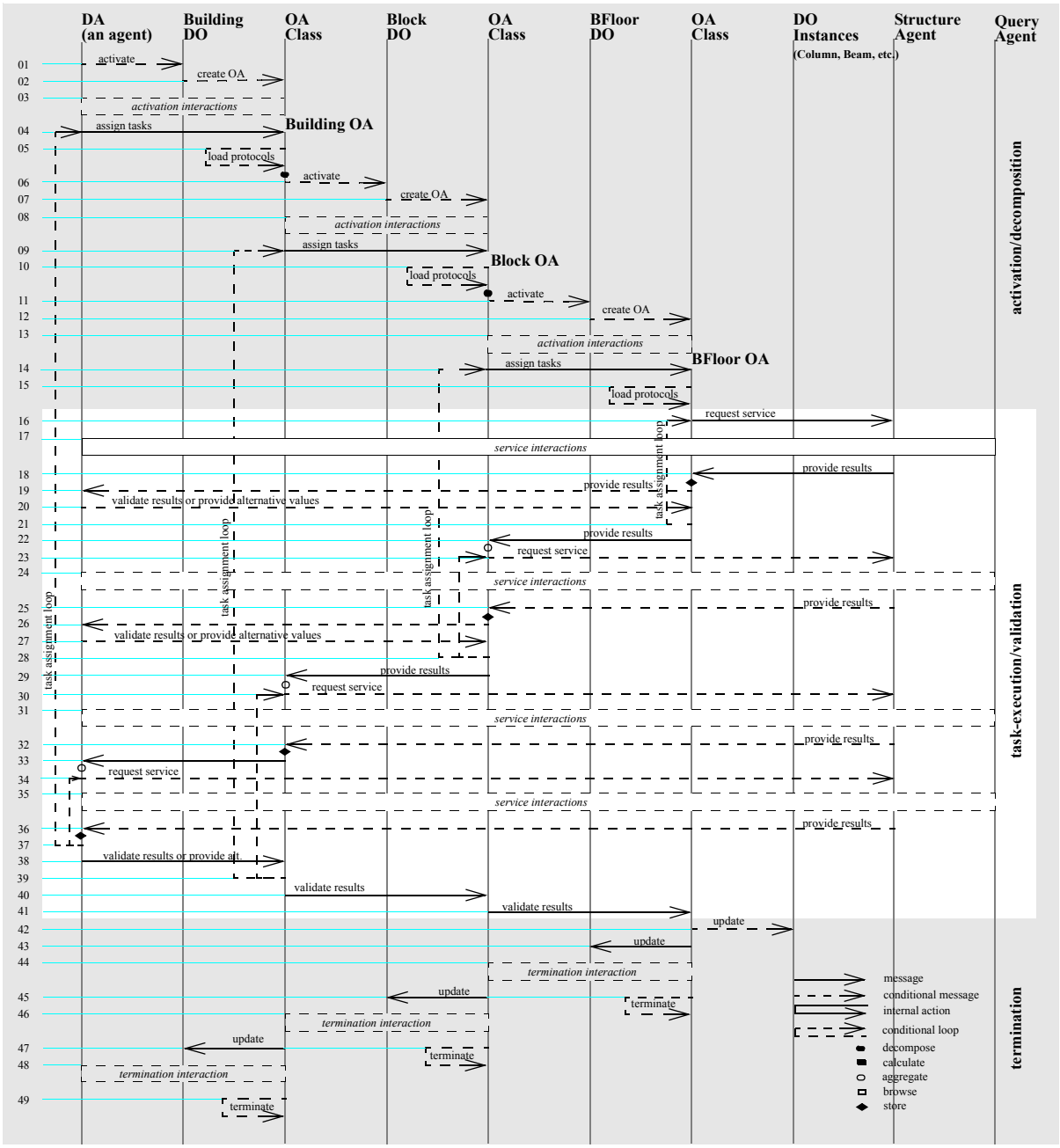


FIGURE 4.7.

Event-trace of a structural analysis task executed by a Building-OA.

4.7 Chart 6. Structural Analysis Task

In a structural analysis task, the DA is interested in analyzing the structural stability of a Building-DO. This should include analysis of building loads classified according to each block-DO and BFloor-DO. The analysis should also

include the individual structural elements within each Block-DO or Floor-DO (e.g., columns, beams, trusses, sheer walls). The analysis should include statistical information about the number of structural elements (or zones) with specifications not sufficient for the prospected loads. The event-trace chart for such a structural analysis is shown in Figure 4.7.

The main players

The scenario of this task involves three main players; a DA, a Building-OA and a structure-EA.

- The DA activates the Building-DO and assigns it a structural analysis task, which, in turn, triggers a chain of activation by the Building-OA down the hierarchy to its Block-DOs to their BFloor-DOs (which are the last nodes in the structural analysis task dependent hierarchy).
- Interacting with the structural-EA, each activated OA runs a structural analysis session regarding its own state. The sequence of analysis sessions starts from the leaf OAs where the results are provided to their super-agents. Each super-agent aggregates (when needed) the results provided by its sub-OAs, and so forth up to the DA level.
- The OAs update the information of its DOs and terminates itself.

The following are optional events.

- The DA triggers a conflict handling session.
- The DA requests the implementation of new attribute values.

1. For each analysis session the assigned Building-OA provides the DA with information that may contain at least one of the following:
   - structural stability of each block -DO and BFloor-DO (see The task above);
   - a warning for each structural DO (or zone) where its current specification is not sufficient for the expected loads.
2. The DA examines either new DO attribute values for DOs (or a collection of DOs in a zone).
3. (Optional) The DA examines new DO attribute values for DOs interested in the attribute values of the DOs in the task dependent hierarchy. This may occur as a result of conflict handling sessions (if executed).
4. (Optional) Implementation of examined DO attribute values and updating of the DO relations.

Details of the interactions

The main steps are 16-41.

1. *Conditional (if the targeted BFloor-DO is not activated): A DA sends an activation message to a Building-DO.*
2. *Conditional (if step 1 is executed): The Building-DO instantiates a Building-OA. A Building-OA is created.*
3. *Conditional (if step 2 is executed): Activation interactions (see Chart 1 steps 3-6).*
4. *The DA assigns a structural analysis task to the created Building-OA.*
5. *Conditional (if the appropriate protocols are not loaded): The Building-OA loads the structural analysis protocols.*
   The Building-OA uses the protocols to decompose the task among qualified sub-DOs (linked to the Building-OA in a task dependent hierarchy)
6. *Conditional (if the targeted block-DOs are not activated): The Building-OA sends activation messages to its block-sub-DOs.*
7. *Conditional (if step 6 is executed): The block-DO class instantiate block-OAs for each block-DO.*
   The created block-OAs are sub-OAs to the Building-OA.
8. *Conditional (if step 7 is executed): Activation interactions (see Chart 1 steps 3-6).*
9. *The Building-OA assigns structural analysis sub-tasks to the created block-OAs.*
10. *Conditional (if the appropriate protocols are not loaded): Each block-OA loads the structural analysis protocols.*
    The block-OAs uses the protocols to decompose the task among potential sub-DOs (linked to each block-OA within the task dependent hierarchy).
11. *Conditional (if the targeted BFloor-DOs are not activated): Each block-OA sends an activation messages to all its BFloor-DOs.*
12. *Conditional (if step 11 is executed): The bfloor class instantiates BFloor-OAs for each BFloor-DO. The created BFloor-OAs are sub-OAs to the block-OAs.*
13. *Conditional (if step 12 is executed): Activation interactions (see Chart 1 steps 3-6).*
14. *The block-OA assigns structural analysis sub-tasks to the created BFloor-OAs.*
15. *Conditional (if the appropriate protocols are not loaded): Each BFloor-OA loads the structural analysis protocols.*
    The decomposition stops when no other sub-DOs are found in the task dependent hierarchy, the current BFloor-OAs are leafs in the decomposition tree.
16. Interacting with the structural-EA each BFloor-OA initiates a structural analysis task session.
17. Task execution interactions (see Chart 2 steps 7-19).
    In a multi story building, the analysis of the lower BFloor-OAs are dependent on the higher ones. Therefore, the structural analysis starts from the upper

BFloor-OAs to the lower BFloor-OAs.

Note1: If the DA assigned a structural analysis task to a lower BFloor-OA the later will activate the higher BFloor-OA to execute similar task and so forth until all higher (or adjacent) BFloor-OA loads are calculated (see Chart 2 step 19).

Note 2: The information requested from the BFloor-OAs during the interaction with the structural-EA can be obtained directly from the each BFloor-OA such as the building floor dimensions. Other information of the sub-DOs (which is not activated) such as beam dimensions and material requires the interaction of each BFloor-OA with its structural elements sub-DOs within the task dependent hierarchy (Sub-DOs such as baring Wall-DOs, Beam-DOs, and Column-OAs). Those sub-DOs may be activated during the task execution session, but for a basic structural analysis task it is not necessary to do so. The BFloor-OA can access the information needed from such linked DOs (see Chart 2 steps 8-10). No agent properties are necessary for those DOs during the session.

18. The structure-EA provides each BFloor-OA with its structural analysis results (as assigned; load and stress analysis, stability, relation to prototypical cases etc.).
    Each BFloor-OA stores its results until validated.
19. Interface option (to allow the DA to validate task results not assigned directly by him/her): Each BFloor-OA provides its structural analysis results to the DA with.
20. Conditional (if step 19 is executed): DA validates the BFloor-OAs structural analysis results if satisfactory, or:
    provides alternative attribute values for task reassignment.
21. Conditional (if alternative values are provided in step 20): Any BFloor-OA may reassign the task to the structure-EA to examine the alternative values. In such case steps 16-21 are repeated.
22. Each BFloor-OA provides its structural analysis results to its block-OA. Each block-OA uses it protocols to aggregate the results collected from its BFloor-OAs.
    Note: The nature of aggregation of the BFloor-OA structural sub-results is different from the aggregation of the cost or daylighting evaluation sub-results (as in Chart 4 & 5). The aggregation can be a classification of loads

according to the structural elements (e.g., column, barring walls, cores) where the loads are transferred from the higher BFloor-OAs to the lower ones.
Each block-OA stores its results until validated.

23. Conditional (if aggregation is performed in step 22): Each block-OA requests the structural-EA to evaluate its aggregated results.
24. Conditional (if step is 23 executed): Task execution interactions (see Chart 2 steps 23-31)
25. Conditional (if step 24 is executed): The structural-EA provides its aggregation evaluation results to each block-OA.
Each block-OA stores the results until validated.
26. Interface option (to allow the DA to validate task results not assigned directly by him/her): Each block-OA provides its structural analysis results to the DA.
27. Conditional (if step 26 is executed): The DA validates the block-OA results or provide alternative attribute values for task reassignment.
28. Conditional (if alternative values are provided in step 27): A block-OA reassigns the task to the structure-EA to examine the alternative values. In such case steps 14-28 are repeated or runs another aggregation evaluation session In such case steps 23-28 are repeated.
29. Each block-OA provides its structural analysis results to the Building-OA. The Building-OA uses it protocols to aggregate the results collected from its block-OAs.
Note: As in step 22, the nature of aggregation of the Building-OA sub-results is different from that of the cost or daylighting evaluation sub-results (as in Chart 4 & 5). It is also different from the aggregation of the BFloor-OA results by their block-OA. More structural analysis may be required.
30. Conditional (if aggregation is performed in step 29): The Building-OA requests the structural-EA to evaluate its aggregated results.
31. Conditional (if step is 30 executed): Task execution interactions (see Chart 2 steps 23-31)
32. Conditional (if step 31 is executed): The structural-EA provides its aggregation evaluation results to the Building-OA.
The Building-OA stores the results until validated.
33. The Building-OA provides its evaluation results to the DA.
Conditional (if more than one Building-OA provided structural analysis results): The DA aggregates the results of the Building-OAs.

34. Conditional (if aggregation is performed in step 33): The DA requests the structural-EA to evaluate its aggregated results.
35. Conditional (if step is 34 executed): Task execution interactions (see Chart 2 steps 23-31)
36. Conditional (if step 35 is executed): The structural-EA provides its aggregation evaluation results to the DA.
37. Conditional (if the results provided in step or 36 are not satisfactory): The DA either reassigns the initial task to the Building-OA with modified values or requirements, in which case steps 4-37 are repeated, or runs another aggregation evaluation session, in which case steps 34-37 are repeated.
38. The DA validates the results (of step 33) of the Building-OA, or: provides alternative attribute values for task reassignment.
39. Conditional (if alternative values are provided in step 38): The Building-OA reassigns the task to the structure-EA to examine the alternative values. In such case steps 9-38 are repeated, or runs another aggregation evaluation session In such case steps 30-38 are repeated.
40. The Building-OA validates the results (of step 29) of each of its block-OAs.
41. Each block-OA validates the results (of step 22) of each of its BFloor-OAs.

42. Conditional (if attribute values of related DOs are to be modified): Each BFloor-OA updates the information of its sub-DOs (e.g., beams, columns).

43. Each BFloor-OA (of step 12) updates its DO information.

44. Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).
Each BFloor-OA terminates itself

45. Each block-OA (of step 7) updates its DO information.

46. Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).
Each block-OA terminates itself

47. The Building-OA updates its DO information.

48. Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).

49. Conditional (if step 48 is executed): The Building-OA terminates itself.

4.8 Chart 7. Handling Conflict Over Window Glazing Area

This possible conflict situation arises when the DA is interested in evaluating the cost of a Room-DO. The cost evaluation which includes a cost classification according to the room sub-DOs within Room-OA1 hierarchy (e.g., windows, doors, walls). The cost evaluation results returns the total cost of the Room-DO and an itemized cost of its activated sub-DOs. The DA may find the cost of the Room-OA is above the prototypical value, and elects examine the reduction of the glazing area to cut the total cost of the Room-OA. The $interest_{list}$ of the glazing area attribute of the Window-DO includes the daylighting-EA. The DA activates another OA of the same Room-DO (namely Room-OA2) to conduct a daylighting evaluation task in respect to the reduced glazing area. The DA runs multiple evaluation sessions between Room-OA1 and Room-OA2 until cost and daylighting evaluation results are validated. The event-trace chart is shown in Figure 4.8.

The main players

The scenario of this chart involves six main players; a DA, two Room-OAs of the same Room-DO, a Window-OA, a cost-EA and a daylighting-EA.

The assigned task

The DA is interested in evaluating the cost of a Room-DO. The evaluation should include a cost classification according to the room sub-DOs within the Room-OA hierarchy (e.g., windows, doors, walls). This includes the total cost of the Room-DO and an itemized cost of its activated sub-DOs. It should also include the total cost of each DO type of the Room-DO such as the total cost of all Wall-DOs, all Window-DOs etc.

The major events

- The DA activates the Room-DO and assigns it a classified cost evaluation task, which, in turn, triggers a chain of activation by Room-OA1 down the hierarchy to its Window-DOs, Wall-DOs and all other leaf-DOs of the hierarchy.
- Interacting with the cost-EA, each activated OA runs a cost evaluation session regarding its own state. The sequence of evaluation sessions starts from the leaf OAs where the results are provided to Room-OA1. Room-OA1 aggregates the results provided by its sub-OAs, and provides it to the DA. The cost of the Room-DO is found to be above the prototypical value. Reviewing the cost of each sub-DO (as provided by Room-OA1) the DA decides to reduce the window size to reduce the total room cost.

- The DA triggers a conflict handling session about the Window-OA1 to check for conflicts over the new window size. Accordingly, the Window-OA provides the DA with a list of interested DOs and EAs, the list includes the daylighting-EA. The DA activates another Room-OA (Room-OA2) of the same Room-DO to evaluate the daylighting levels in respect to the new window size. The DA examines various window sizes until both Room-OA1 and Room-OA2 provide satisfactory cost and daylighting evaluation results. If the results are not satisfactory the DA may run other conflict handling sessions over suggested attribute values of other sub-DOs of the Room-DO (such sub-DOs may need to be activated first).
- The OAs update the information of its DOs and terminates itself.

Additionally the DA may request

- The implementation of new attribute values of the Window-DO (or the any other attribute values of any sub-DO of the Room-DO).

The expected results

1. For each evaluation session the assigned Room-OA1 provides the DA with information that may contain at least one of the following:
   - cost of the Room-DO, and the classified cost as assigned (see the assigned task);
   - prototypical cost of each of the DOs above;
   - a warning for each DO that exceeds the specified prototypical cost.
2. The DA examines either new DO attribute values for DOs in the task dependent hierarchy or new cost constraints.
3. (Optional) The DA examines new DO attribute values for DOs interested in the attribute values of the DOs in the task dependent hierarchy. This may occur as a result of conflict handling sessions (if executed).
4. (Optional) Implementation of examined DO attribute values and updating of the DO relations.

Details of the interactions

The main steps are 25-50.

1. *A DA assigns a cost evaluation task to a Room-OA (namely Room-OA1 in this chart).*
2. *Conditional (if the appropriate protocols are not loaded): Room-OA1 loads the cost evaluation protocols.*
   Room-OA uses it protocols to decompose the cost evaluation task among the qualified sub-DOs (e.g., walls, doors, windows, floors, ceilings).

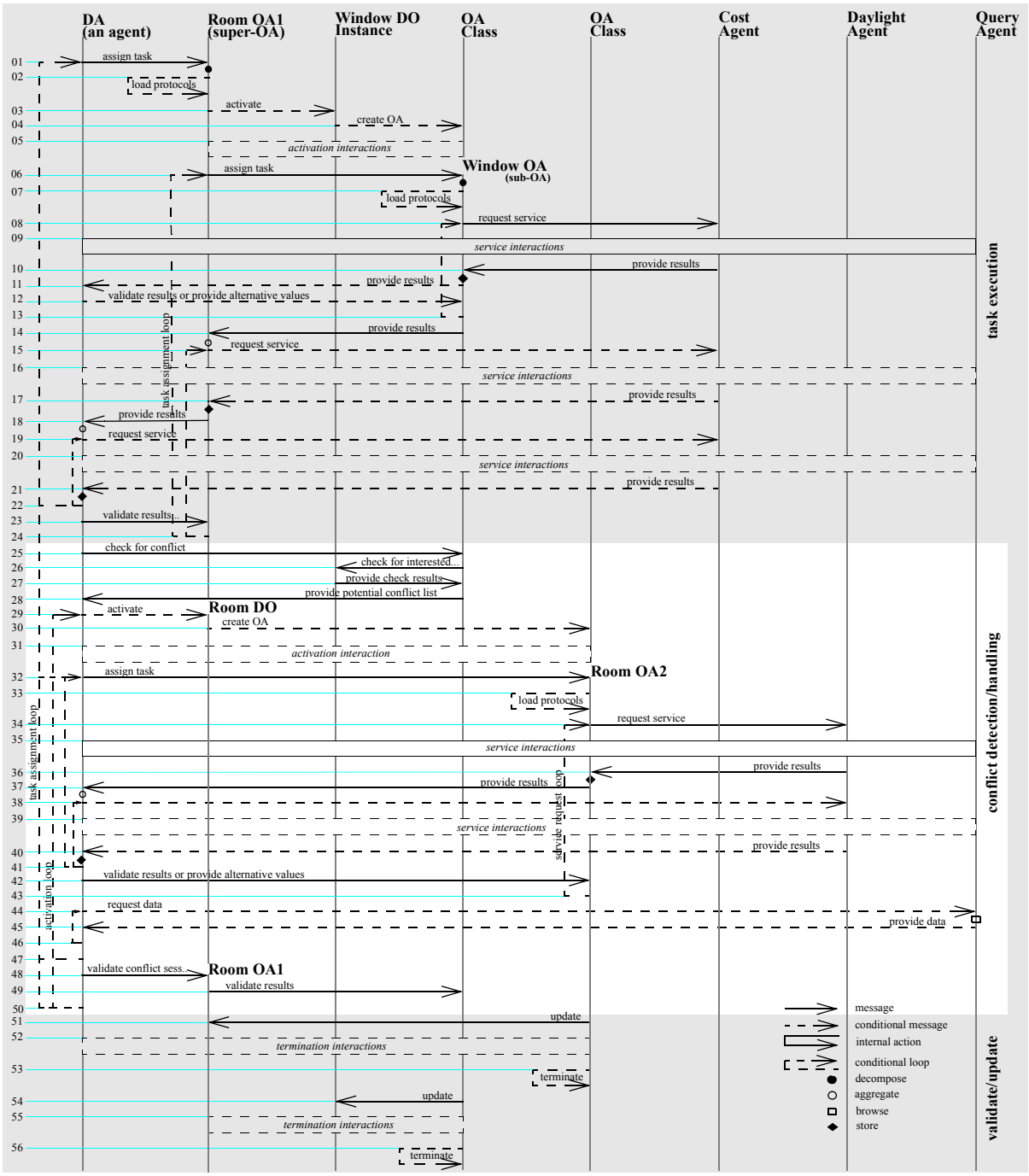


FIGURE 4.8.

Event-trace of a conflict handling session over a Window-OA glazing area attribute.

3. *Conditional (if the targeted Window-DO is not activated): Room-OA1 sends an activation message all its qualified sub-DOs, however, in this Chart the Window-DO will be the only mentioned DO since it is the focus of the conflict handling session described.*

4. *Conditional (if step 3 is executed): The Window-DO class instantiates a Window-OA. A Window-OA is created.*
5. *Conditional (if step 4 is executed): Activation interactions (see Chart 1 steps 3-6).*
6. *Room-OA1 assigns a cost evaluation sub-task to the Window-OA (a sub-OA in this chart).*
7. *Conditional (if the appropriate protocols are not loaded): The Window-OA loads the cost evaluation protocols.*
   Note: No further decomposition is carried, the Window-DO is a leaf on the task dependent hierarchy.
8. *Interacting with the cost-EA the Window-OA initiates a task execution session.*
9. *Task execution interactions (see Chart 2 steps 7-19).*
10. *The cost-EA provides its task execution results to the Window-OA.*
    The Window-OA stores the results until validated.
11. *Interface option (to allow the DA to validate task results not assigned directly by him/her): The Window-OA provides the DA with the task execution results.*
12. *Conditional (if step 11 is executed): The DA validates the results of the Window-OA, or:*
    provides an alternative set of attribute values for task reassignment.
13. *Conditional (if alternative values are provided in step 12): The Window-OA reassigns the task to the cost-EA to examine the alternative values. In such case steps 8-13 are repeated.*
14. *The Window-OA provides its task execution results to Room-OA1.*
    Conditional (if aggregation is needed): Room-OA1 aggregates the results of its sub-OAs (windows, doors, walls etc.).
15. *Conditional (if aggregation is performed in step 14): Room-OA1 requests the cost-EA to evaluate the aggregated results.*
16. *Conditional (if step 15 is executed): Evaluation interactions (see Chart 2 steps 23-31) to examine the aggregated results provided in step 14.*
17. *Conditional (if step 16 is executed): The cost-EA provides its evaluation of the aggregated results to Room-OA1.*
    Room-OA1 stores the results until validated.
18. *Room-OA1 provides the DA with its task execution results for validation.*
    Conditional (if more than one Room-OA provided results): The DA aggregates the provided results.
19. *Conditional (if aggregation is performed in step 18): The DA requests the cost-EA to evaluate the aggregated results.*
20. *Conditional (if step 19 is executed): Evaluation interactions (see Chart 2 steps 23-31) to evaluate the aggregated results provided in step 18.*

21. *Conditional (if step 20 is executed): The cost-EA provides its evaluation of the aggregated results to the DA.*
    The DA stores the results until validated.

22. *Conditional (if the results provided in either step 18 or step 21 are not satisfactory): The DA runs either a task execution session in which case steps 1-22 are repeated, or an aggregation session in which case steps 19-22 are repeated.*

23. *The DA validates the results (of step 18), or:*
    provides Room-OA1 with alternative attribute values for either task reassignment or aggregation reevaluation of the sub-results provided in step 14.
    For this scenario the DA selects to reduce the window size to reduce the window cost and room cost in turn.

24. *Conditional (if alternative values are provided by the DA in step 23): The Room-OA runs either a task execution session to examine the alternative values, in which case steps 15-24 are repeated, or an aggregation session, in which case steps 6-24 are repeated.*
    The new changes in the window dimensions satisfies Room-OA1 cost requirements. This attribute value change requires a conflict check with other DOs and EAs interested in the window dimension.

25. Interface option (to allow for conflict check): The DA requests a conflict check (with other interested DOs or EAs) before the attribute values of the Window-OA are modified.

26. The Window-OA checks with its Window-DO for a list of DOs and OAs with interest in the targeted attribute values (i.e., which is subject to modification by the DA in respect to the cost evaluation task results). The interested DOs or EAs are potential candidates for conflict over the recommended attributes values. The interested DOs may have attributes values that are linked to the Window-OA dimensions. The Window-OA may also have EAs (e.g., daylighting) that are interested in the dimensions subject to modification. In this scenario, the DA request from the Window-OA a list of all DOs and EAs interested in its dimensions attribute value.

27. The Window-DO provides a list of DOs and EAs interested in the targeted attribute values.

28. The Window-OA provides the DA with the list of interested DOs and EAs.

29. Conditional (if the list contains one or more DO or OA): The DA selects a set of DOs and EAs from the list (according to the DA's criteria for most critical conflicts) to activate for conflict check.
    Conditional (if the targeted DO is not activated): The DA sends an activation message to the first DO on the selected set. For an interested EA, such as the

daylighting-EA, the DA may reassign the new daylighting evaluation task to the same Room-OA1 or to a new Room-OA (namely Room-OA2 in this scenario).

30. Conditional (if step 29 is executed): The DO instantiates another Room-OA (of the same Room-DO) from the OA class.
    Room-OA2 is created.
31. Conditional (if step 30 is executed): Activation interactions (see Chart 1 steps 3-6).
32. The DA assigns a daylighting evaluation task to Room-OA2 to examine the suggested window dimensions of step 21 (which is the focus of the conflict handling session).
33. Conditional (if the appropriate protocols are not loaded): Room-OA2 loads the daylighting protocols.
34. Interacting with the appropriate daylighting-EA Room-OA2 initiates a daylighting evaluation session.
35. Task execution interactions (see Chart 2 steps 7-19).
36. The daylighting-EA provides its evaluation results to Room-OA2.
    Room-OA2 stores the results until validated.
37. Room-OA2 provides its evaluation results to the DA.
    Conditional (if aggregation is needed): The DA aggregates the sub-results provided in step 36.
38. Conditional (if aggregation is performed in step 37): The DA requests the daylighting-EA to evaluate the aggregated results.
39. Conditional (if step 38 is executed): Evaluation interactions to evaluate the aggregation results provided in step 37 (see Chart 2 steps 23-31).
40. Conditional (if step 39 is executed): The daylighting-EA provides its evaluation of the aggregated results to the DA.
    The DA stores the results until validated.
41. Conditional (if the results provided in either step 37 or step 40 are not satisfactory): The DA runs either a task execution session in which case steps 32-41 are repeated, or an aggregation session in which case steps 38-41 are repeated, or:
42. The DA validates the results (of step 37), or:
    provides alternative attribute values for task reassignment.

43. Conditional (if alternative values are provided by the DA in step 42): Room-OA2 runs another session to examine the alternative values. In such case steps 34-43 are repeated.
44. Conditional (if the results provided in step 37 are not satisfactory): The DA sends a query request to obtain information from the query agent(s) to help in reassigning the task to examine alternative attribute values to either Room-OA1 or Room-OA2.
45. Conditional (if step 44 is executed): The query-agent provides the DA with the applicable information (as available).
46. Conditional (if the information provided in step 45 is not sufficient): The DA sends a modified query request to the query-agent. In such case steps 44-46 are repeated.
47. Conditional (if the results of step 37 are not satisfactory): The DA reassigns another evaluation task to either Room-OA1 in which case steps 1-47 are repeated, or to Room-OA2 in which case steps 32-47 are repeated.
48. The DA validates the conflict handling session results (started in step 25).
49. Room-OA1 validates the results of Window-OA1 (provided in step 14).
50. Conditional (if there are more than one DO or EA in the set selected in step 29): The DA either activates the next DO in the set, in which case steps 29-49 are repeated, or assigns another evaluation task to Room-OA1 in relation to the next EA in the set, in which case steps 1-49 are repeated.
51. *Room-OA2 updates its DO information (after the implementation of the task results, see Chart 2 steps 39-41). In this scenario the Room-DO still represented by Room-OA1, therefore, any update is managed by Room-OA1.*
52. *Conditional (if no other task is to be executed): Termination interactions (see Chart 1 steps 26 & 27).*
53. *Conditional (if step 52 is executed): Room-OA2 terminates itself.*
54. *The Window-OA updates its DO information (after the implementation of the task results, see Chart 2 steps 39-41).*
55. *Conditional (if no other task is to be executed): Termination interactions (see Chart 1. steps 26 & 27).*
56. *Conditional (if step 55 is executed): The Window-OA terminates itself.*

5 Task Handling Algorithms

5.1 Which Tasks?

In a decision making session the success of an OA depends on its ability to manage and utilize the environment resources to execute self-initiated or assigned tasks. Self-initiated tasks require the OAs to have reasoning capabilities in order to justify the initiation of the task, a property that can be considered for an advanced version of an OA-based Environment. Such a property requires an OA to be:

- Permanent in the environment, or in an active status even when not assigned tasks (i.e., have the capabilities to observe the environment activities);
- Able to interpret changes in the environment (whether these changes are observed by the OA or received through an update mechanism);
- Able to execute tasks that are related either to its own state or to the states of other DOs or OAs in the environment.[1]

Within the scope of this thesis, an OA may either execute tasks that are directly assigned by other agents or tasks that are executed within the scope of another assigned task. The latter are a limited form of self-initiated tasks. An OA in this sense is merely initiating tasks as a reaction to the originally assigned task.[2] Therefore, within the scope of this work, OAs are considered as:

- Temporal and only existing during the execution of an assigned tasks;

1. Sub or super-DOs or OAs in a hierarchy, or even hierarchy non-related DOs and OAs.
2. See reactive planning in Chapter 3.

- Not observing the environment and thus, cannot interpret more than a finite set of tasks that are directly assigned to each;
- Only able to execute tasks pertaining to their own state.

The expansibility of the OA capability to include self-initiated tasks requires further research and experimentation. OAs with such capabilities are considered autonomous. In Chapter 3, I argued against fully autonomous agents in design environments, and therefore, only consider semi-autonomous OAs. The fundamental ability of an OA to execute assigned tasks is the core of an OA-based environment. This chapter is concerned with the development of algorithms needed by the OAs to carry on the execution of assigned tasks. An OA algorithm, in this sense, is a dynamic short term plan (or reactive plan) designed to enable an OA to handle a family of tasks. Within an algorithm multiple OA protocols (namely object-type, task-type and domain protocols) are utilized.

Event-trace charts presented in Chapter 4 provide the basis for developing the set of task handling algorithms developed in this chapter. Two main interaction algorithms are developed following the charts of Chapter 4; the first is for executing evaluation tasks; the second is for conflict handling (which is a DA controlled iteration of local evaluation tasks involving more than one OA).

Evaluation tasks are dependent on the notion of decomposition among applicable candidates of the DOs of the environment. This chapter presents decomposition mechanism that are crucial to the execution of evaluation tasks.

Executing generation tasks, on the other hand, are not dependent on the notion of task decomposition. An OA would interact directly with a generative EA to produce alterative solutions. The generation mechanism is domain specific and is applied by the EA, the OA may interact only to provide information about its own state when requested. It is, therefore, not necessary to develop interaction algorithms for generation tasks. Executing implementation tasks is environment dependent[3] and is also not significant to this framework.

3. For instance, implementing a recommended modification of a wall-OA thickness attribute depends on the applications utilized on the environment. Most CAD system has a set of function calls (or a language, etc.) to execute attribute modifications (e.g., AutoLisp in AutoCAD). An implementation task may be complex and require a sequence of actions by the CAD system to geometrically modify an attribute value.

5.2 OA Task Execution Algorithms

Once an OA is activated (i.e., a class instance is created and registered with its super-agent, see details in Chapter 6) the super-agent assigns a task to the OA. The super-agent provides the task-type, task-domain, and task-focus (e.g., a value of an attribute to be examined. This sets the context which the task revolve around. The assigned task is classified by the OA and handled by the appropriate algorithm (Evaluate, Implement etc.).

Algorithm: ***ClassifyTask (task)***

```
start
1   if task-type == ?
2       case 1. ? == evaluate        => Evaluate (Task-Domain, Task-Focus)
3       case 2. ? == generate        => Generate
4       case 3. ? == check-conflict=> ConflictChecking&Handling
5       case 4. ? == implement       => Implement (attribute, new-value)
6   else
7       error message
8   end if
end
```

5.2.1 Evaluation

An OA handles an evaluation task according to the task-domain and task-focus. The OA uses the domain decomposition protocol to delegate the task to the sub-DOs in its own OA-hierarchy. The position of the OA in the domain-hierarchy implies whether further decomposition is applicable. For instance, if an OA is representing a leaf-DO in a task hierarchy (e.g., Plumbing-DO in Figure 5.1), no further top-down decomposition would be applicable regardless of the nature of the task being executed. In such cases, the leaf OA needs to execute the evaluation task interacting with the EA of the task-domain. Figure 5.1 shows a decomposition of a Block-DO for a cost evaluation task. The decomposition is centered around construction categories (such as framing, roofing, etc.) instead of the Block-DO spatial elements (i.e., floors, zones, rooms, etc.). This is a flat/ simple decomposition which corresponds to the example in Figure 3.3A. The activated DOs (e.g., Foundation-DO) uses (and activate) related DOs (e.g., Slab-DOs, Footing-DOs) during the course of evaluating its own cost.

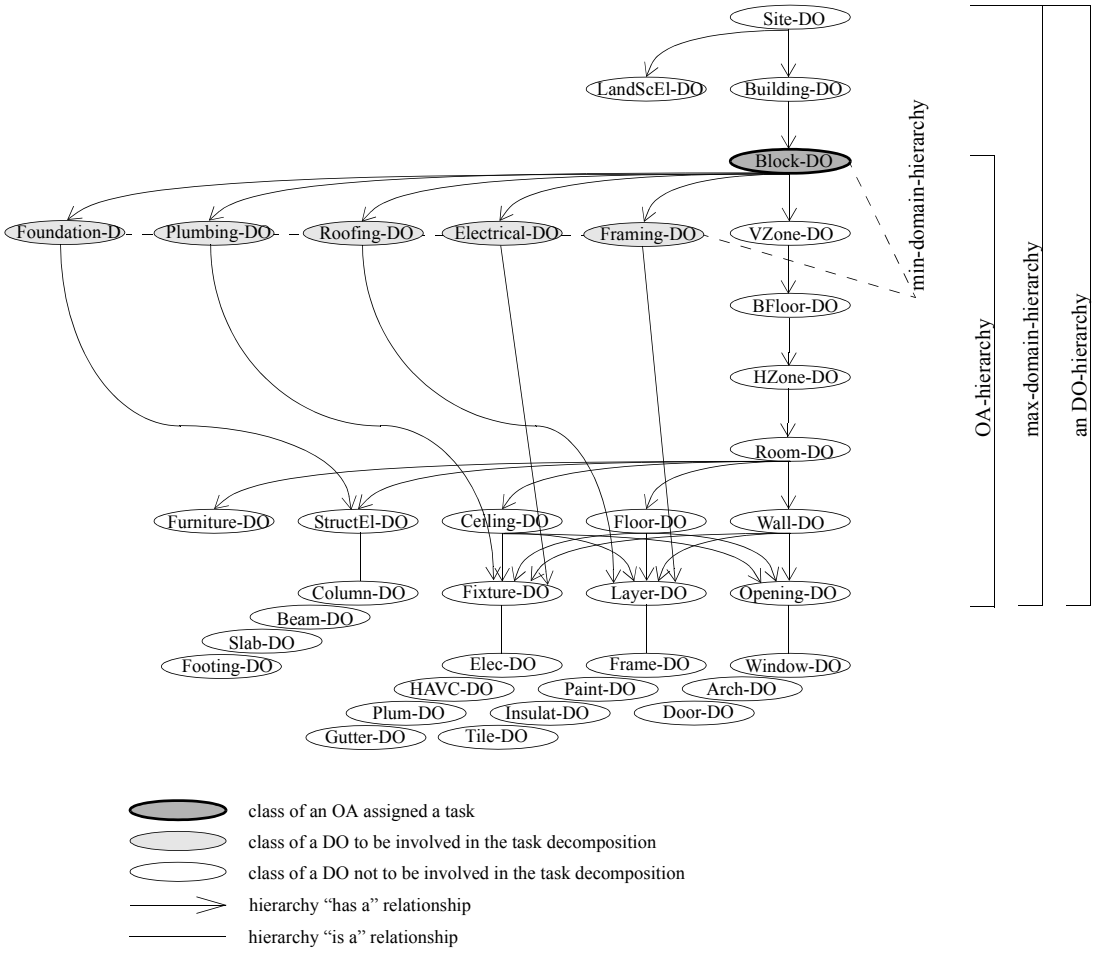


FIGURE 5.1.

Decomposition of a Block-DO cost evaluation task.

The task-focus affects the decomposition in a different manner. If the task-focus implies a classification (i.e., the context of the task is a classified evaluation, such as cost of a Block-DO per BFloor-DO) the DO of classification (the Block-DO in this case) must be included in the decomposition.

The following examples show various task-focus cases which affect either the decomposition or the aggregation when a **DO** is executing an assigned cost evaluation task such as:

- Block-DO total cost classified per BFloor-DO:

  As illustrated in Figure 5.2, all leaf-DOs of the construction categories are included in the decomposition since the cost of the Block-DO is the aggregation of its construction categories leaf-DOs' cost. The BFloor-DO

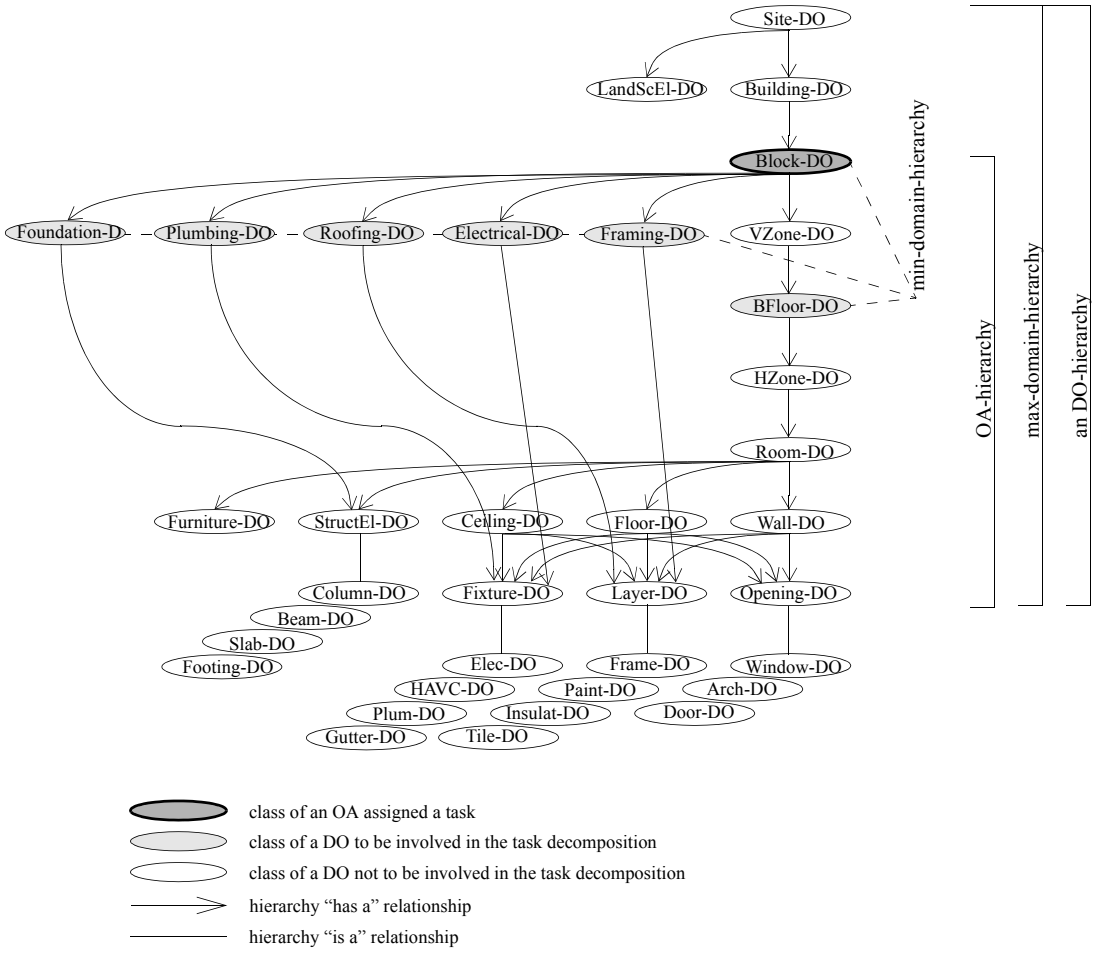


FIGURE 5.2.

Decomposition of a Block-DO cost evaluation task (classified per BFloor-DO).

is also included in the decomposition as the DO of classification ($DO_{classification}$ hereafter). In the aggregation process, the cost is aggregated according to each BFloor-DO. The BFloor-DOs costs are then aggregated to provide the total cost of the Block-DO.

- Total cost of the StructElement-DOs classified per VZone-DO:

  Figure 5.3 shows a decomposition of a Block-DO cost evaluation task of StructElement-DOs' (classified per VZone-DO). The VZone-DO class as the $DO_{classification}$ presents a different decomposition case.

- Total cost of a Block-DO that includes shared DOs:

  When a Wall-DO is shared among Block-DOs, another layer of computation is needed for decomposition and aggregations. Geometric

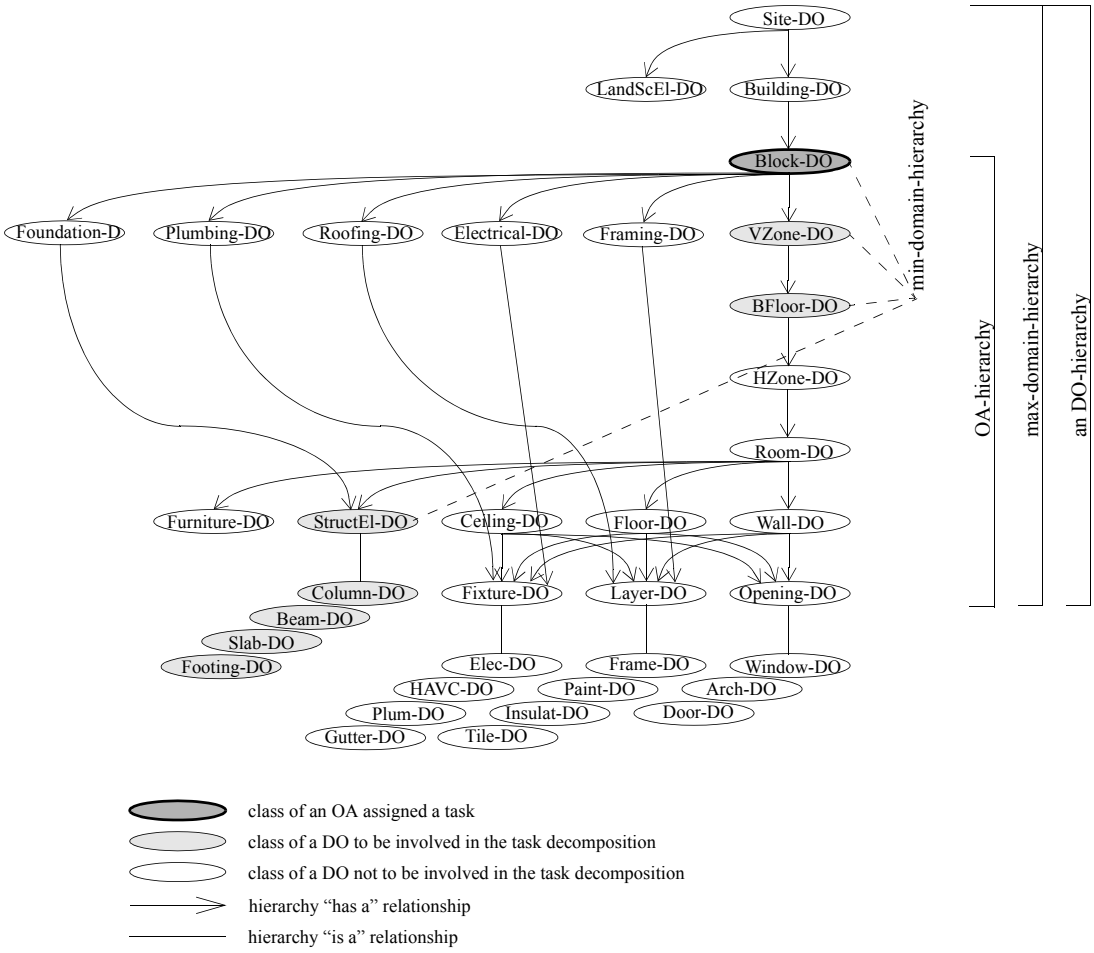


FIGURE 5.3.

Decomposition of a Block-DO cost evaluation task of StructElement-DOs' (classified per VZone-DO).

computations may be needed to determine the portion of each shared DO (e.g., Wall-DO in each Block-DO). This implies that **special DOs (e.g., shared DOs) may require another layer of computation for task decomposition and sub-results aggregation**.

- Total cost of a Block-DOs when DO of classification is not a super-DO to the leaf-DOs:

  DO classes of the same level but in different branches of the OA hierarchy may act as super-OAs to each other. For instance, in Figure 5.2, if the $DO_{classification}$ is the VZone-DO instead of the BFloor-DO, the construction DOs (e.g. Framing-DO, Foundation-DO, Painting-DO) would be in the same level of the hierarchy as the VZone-DO. That is, both VZone-DO and construction DOs are direct sub-DOs of the Block-DO class. The flow of

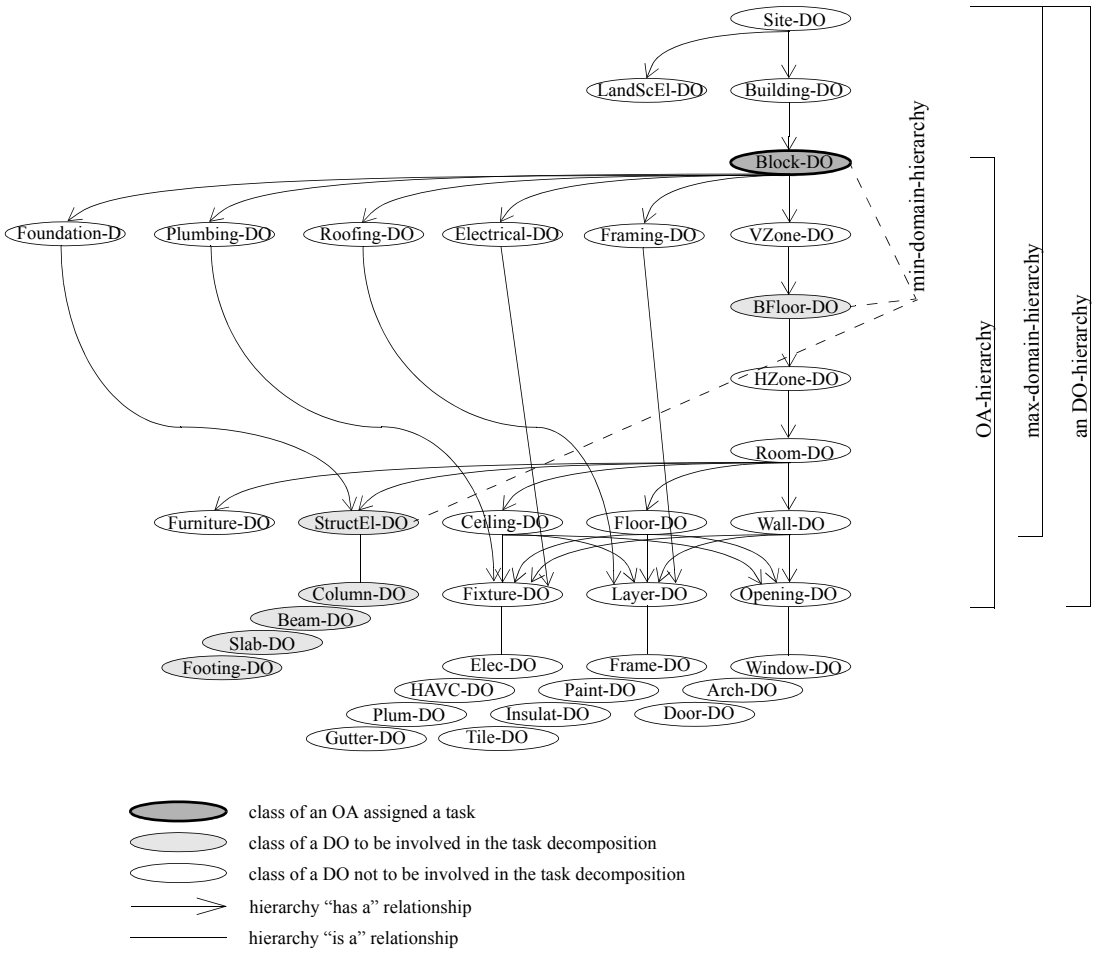


FIGURE 5.4.

Decomposition of a Block-DO structural analysis task.

the cost task decomposition assigned by the Block-OA to the construction OAs goes through the VZone-OA. The VZone-OA acts as super-OA to the construction OAs even though they share the same level in the hierarchy. The Framing-OA interacts directly with the cost-EA to find its own cost. The VZone-OA, as the super-OA to the construction OAs within the context of this task, aggregates the cost results of each construction OA. In other words, **task decomposition does not necessarily follow a top down order in the hierarchy**.

- Total painting cost:

  If paint is not represented explicitly as a DO class (i.e., Painting-DO or Paint-DO classes) in the OA-hierarchy each DO pertaining to the “paint” attribute, of a Wall-DOs or Ceiling-DOs, would be used by the other OA while interacting with the appropriate EA in order to compute the paint area

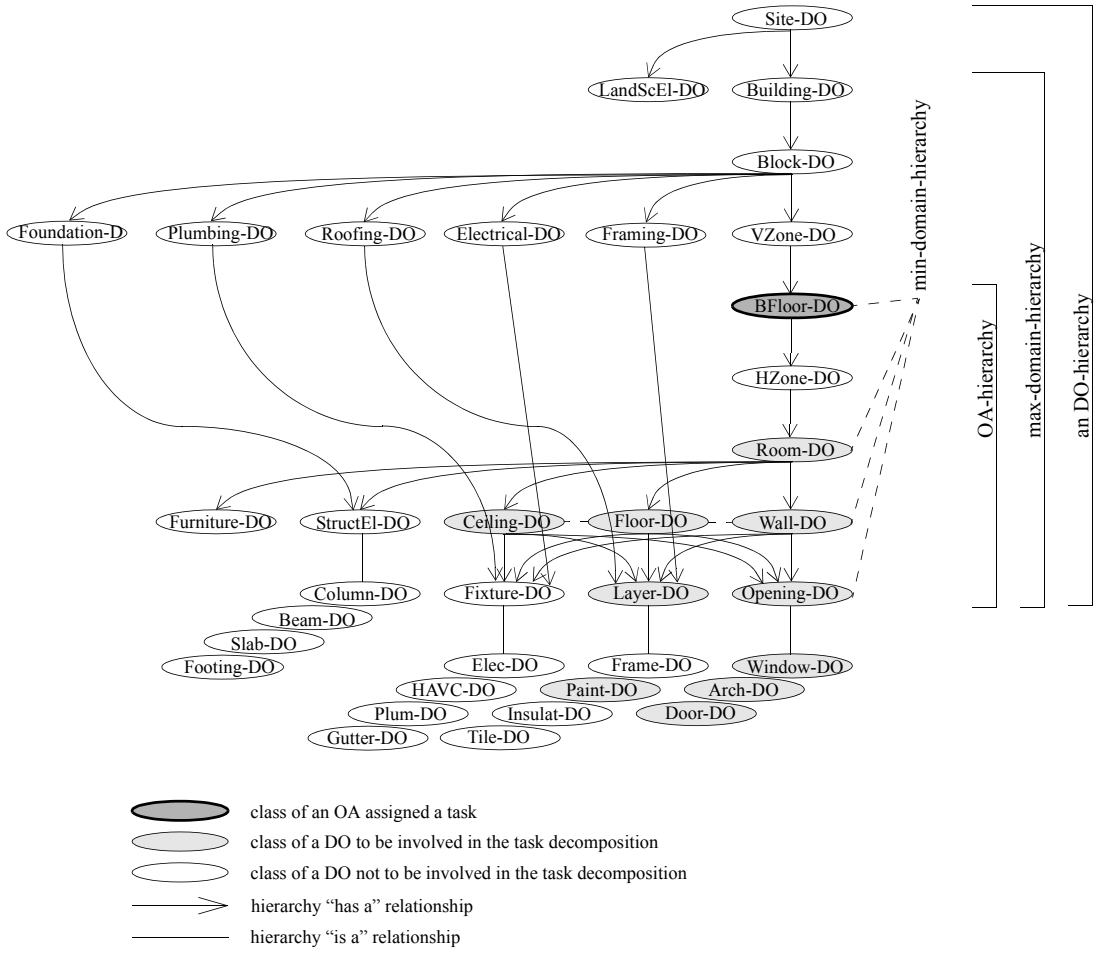


FIGURE 5.5.

Decomposition of a BFloor-DO daylighting evaluation task.

and its cost. This implies that, **computation on a DO attribute may substitute for further decomposition**.

Accordingly evaluation tasks fall into one of the following cases:

If no task decomposition is not required (e.g., the task is assigned to a leaf-OA in the DO-hierarchy or to an OA that hold attributes which substitute for leaf-OAs):

- *case A: (task requires no decomposition)*
  evaluate OA(e.g., Wall-OA cost) or;
  evaluate OA attribute (e.g., Wall-OA paint cost)
  ===>

  1. OA interacts with domain EA (e.g., Wall-OA interacts with Cost-EA)

If task decomposition is required (e.g., the task is assigned to an OA that is neither a leaf-OA nor holds any attributes which substitute for leaf-OAs):

- *case B (task requires decomposition)*
  evaluate OA (e.g. BFloor-OA cost) or;
  evaluate OA attribute (e.g., BFloor-OA painting cost)
  ===>
  1. decompose task among applicable leaf sub-DOs of the hierarchy (use the P_Domain decomposition protocol of this task-domain to identify applicable sub-DOs, then activate and assign sub-tasks to the activated sub-OAs. To evaluate an attribute, only the sub-DOs which holds such attribute should be activated;
  2. repeat case A step 1 for each assigned sub-OA;
  3. aggregate results (use the OA P_Domain aggregation protocol).

- *case C: (task requires decomposition and classification of sub-results)*
  evaluate OA classified by a sub-DO type (e.g., cost of a Block-OA classified per BFloor-DO) or;
  evaluate OA attribute classified by a sub-DO type (e.g., cost of a Building-OA paint classified per Block-DO)
  ===>
  1. decompose task among DOs of classification (e.g., Block-DOs);
  2. repeat case B step 1-3 for each activated DO (e.g., BLock-OA);
  3. aggregate results (use the OA P_Domain aggregation protocol).

Algorithm: ***Evaluate (Task-Domain, Task-Focus)***

start
1 get P_Domain protocols (decomposition, sorting, aggregation)
2 *makeactivationlist* using the P_Domain decomposition protocol,
object-type, and task-focus => return $sortedActivation_{list}$
3 **if** $sortedActivation_{list}$ is not empty
4 get P_ObjectType protocols
5 *Activate (DOs)* first DO class instances in the $sortedActivation_{list}$
=> return sub-OAs
6 *Assign* evaluation sub-task(s) to sub-OAs => return sub-result(s)
7 *Validate (sub-results)*
8 *Aggregate (sub-results)* (use P-Domain aggregation protocol)
9 *Validate (aggregation-result)*
10 **else** (no decomposition is applicable, execute task by the OA itself)
11 *InteractForService (domain-EA)* => return result
12 **if** shared-DO
13 *InteractForService (domain-EA)*[4] to distribute results
14 **else** (not a shared-DO)
15 **end if**
16 *Validate (result)* => return validation
17 **end if**
18 **if** this is not a conflict handling session
19 *ConflictChecking&Handling*
20 **else** (a conflict handling session)
21 get validated conflict session results
22 **end if**
23 **return** results
24 *Update (results)*
25 *Implement (results)*
end

4. The domain-EA, in turn, may interact with the geometric-EA to determine the portion of the shared-OA located on each of its super-OAs (e.g., to find the portion of a Wall-OA on two adjacent Room-OAs).

line 2: Making the activation list

In a hierarchy of an OA (OA-hierarchy hereafter, i.e., all constituent DOs and all their hierarchies) which is executing an assigned task, it is not necessary to decompose the task amongst all DOs that are members of the hierarchy. In respect to the task being executed by an OA, a DO that is a member of an OA-hierarchy is either:

1. related to the task and crucial to the execution of the task.
2. related to the task but not crucial to the execution of the task.
3. not related to the task.

Therefore, an activation list (referred to as $\mathbf{activation_{list}}$ hereafter) of the sub-DOs to participate in the task decomposition must be compiled. This list is compiled using the parameters provided by the P_Domain decomposition protocol (which is used by the OA to execute a task). The decomposition protocol provides the DOs which belongs to the first category, those DOs are the "**min-domain-hierarchy"**. The DA may participate in compiling the $activation_{list}$ by providing members of the second category.

An **activation order** may also be required for execution of certain tasks. The default activation of DOs follows a top-down order through the OA-hierarchy. If the context of the task implies an activation order which differs from the default order, the $activation_{list}$ must be sorted accordingly. For instance, in the classified cost evaluation task of Figure 5.3, if the initial DO-hierarchy is structured so that both VZone-DO class and BFloor-DO class are direct sub-DOs of the Block-DO class a VZone-OA must activate the BFloor-DOs though both may co-exist in the same level within the hierarchy. In such a case, the activation follows a horizontal order in the hierarchy instead of the top-down order (see discussion in page 100). Another example is the day-lighting task of Figure 5.5, the day-lighting of a Room-DO with interior Wall-DOs would depend on the daylighting of an adjacent Room-DO with exterior Wall-DOs. In such a case, the dependency among Room-DOs also requires horizontal decomposition order within the OA-hierarchy. Other cases may require a decomposition that goes up in the hierarchy before it follows the default top-bottom order.

The context of a task is passed from a super-OA to a sub-OA throughout the decomposition. In this sense, the $activation_{list}$ is global within the realm of the task hierarchy, and is used by each sub-OA in the hierarchy to activate the next

set of DOs. Accordingly, to compile the activation$_{list}$ the following procedures are needed:

1. The set of all DO classes (of the existing hierarchy) which are eligible for activation according to the task domain must be defined (see Figure 5.1-5.5). This set is the "**max-domain-hierarchy"** and is bounded by two variables "**domain-hierarchy$_{top}$**" and "**domain-hierarchy$_{bottom}$**, which are the top and bottom classes (or bottom class level, which is a set of classes sharing similar positions in the hierarchy, e.g., the set of all leaf-DOs specifies a class level and may act as domain-hierarchy$_{bottom}$). These two variables differ from domain to another and, therefore, must be provided by the P_Domain decomposition protocol of the task in hand. For instance, in the hierarchy of Figure 5.4, if a structural analysis task is to be performed the Block-DO class may act as the domain-hierarchy$_{top}$ and the StructElement-DO class should be the domain-hierarchy$_{bottom}$. The max-domain-hierarchy is then compiled as a list of all DO classes that is located between the domain-hierarchy$_{top}$ and domain-hierarchy$_{bottom}$ classes (in addition to the domain-hierarchy$_{bottom}$ class itself).
2. The DO classes that should not to be activated are compiled in a "**skip$_{list}$**." The making of the skip list follows the logic explained in the sequel.
3. The "**activation$_{list}$**" is then compiled as the difference between the max-domain-hierarchy$_{list}$ and skip$_{list}$. The DA may elect to insert additional DO classes from the max-domain-hierarchy$_{list}$ to the activation list. Any class added by the DA must conform with the constraints of making the skip$_{list}$ (see "Making the skip$_{list}$" on page110).
4. The activation$_{list}$ is sorted using the P_Domain sorting protocol to determine the order of activating the DOs included in the activation$_{list}$.

The following definitions and properties can be deduced from the preceding discussion and are necessary for the remaining algorithms[5]:

5. **DO-** and **OA-hierarchies** are defined in Appendix A.

max-domain-hierarchy: the set of all eligible classes for task decomposition in respect to this particular domain. This set is defined by the P_Domain decomposition protocol of the task in hand. The max-domain-hierarchy is necessarily a subset of a DO-hierarchy.

min-domain-hierarchy: the minimum set of DO-classes necessary to execute an assigned task. This set is defined by the P_Domain decomposition protocol of the task in hand. The min-domain-hierarchy is also necessarily a subset of a DO-hierarchy and of the max-domain-hierarchy.

domain-hierarchy$_{\mathbf{bottom}}$: a class or a set of classes which represent the lower boundary of a max-domain-hierarchy .

domain-hierarchy$_{\mathbf{top}}$: a DO class, which represent the top boundary of the max-domain-hierarchy.

leaf-DOs: a set of DO-classes that contains all DOs at the lower end of each branch of a hierarchy

DO$_{\mathbf{classification}}$: a DO class used for classifying the results of executing an assigned task (e.g., evaluate cost of a BFloor-DO per Room-DO. The Room-DO class is the DO$_{\text{classification}}$ of such task).

activation$_{\mathbf{list}}$: a set of DO classes of which instances are to be activated to execute sub-tasks during the executing of a task. It is necessarily a subset of the max-domain-hierarchy$_{\text{list.}}$

skip$_{\mathbf{list}}$: a list of DO classes to be skipped during the activation of sub-DOs of an OA-hierarchy. It is a subset of the max-domain-hierarchy.

interest context: the reasons for which an agent or a DO attribute is interested in another DO attribute (e.g., Wall-DO width for Room-DO acoustics).

activation order: the order of activating DOs in an OA-hierarchy during the execution of a task.

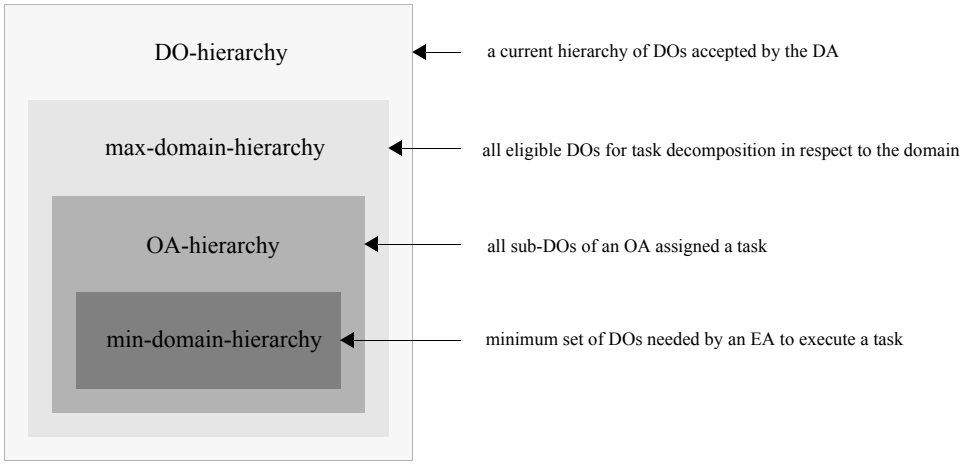


FIGURE 5.6.

Relation between Hierarchies (general case): min-domain-hierarchy < OA-hierarchy < max-domain-hierarchy.

Making the $skip_{list}$ (as a requirement for making the $activation_{list}$)

The $skip_{list}$ is used by the "MakeActivationList" algorithm to compile the $activation_{list}$. If no DO classes are explicitly specified by the P_Domain decomposition protocol or by the DA, the $skip_{list}$ is compiled so as to minimize the number of DOs to be activated, that is, the $skip_{list}$ is maximized. The "MakeActivationList" algorithm uses the following logic to compile the $skip_{list}$.

1. P_Domain specified: The P_Domain protocol must provide a min-domain-hierarchy and may provide a $P\_Domain\_Skip_{list}$. If the P_Domain protocol provided a $P\_Domain\_Skip_{list}$ all DO classes included must be in the final $skip_{list}$. For instance, in Figure 5.4 all DO classes between the BFloor-DO class and the StructElement-DO class and all leaf-DO classes other than StructElement-DO class must be included in the $skip_{list}$.
2. DA specified: If the DA provided a DA_Skip_{list} all DO classes included must be in the final $skip_{list}$ (provided that none of the members of the DA_Skip_{list} is a member of the min-domain-hierarchy) For instance, in Figure 5.4, the DA may elect to include the VZone-DO class in the $skip_{list}$. On the other hand, the DA may chose to activate all the sub-DOs of the OA-hierarchy. In such cases, all sub-DOs in the OA-hierarchy are activated excluding any members of the $P\_Domain\_Skip_{list}$ (if applicable). No DA_Skip_{list} is provided in such case.

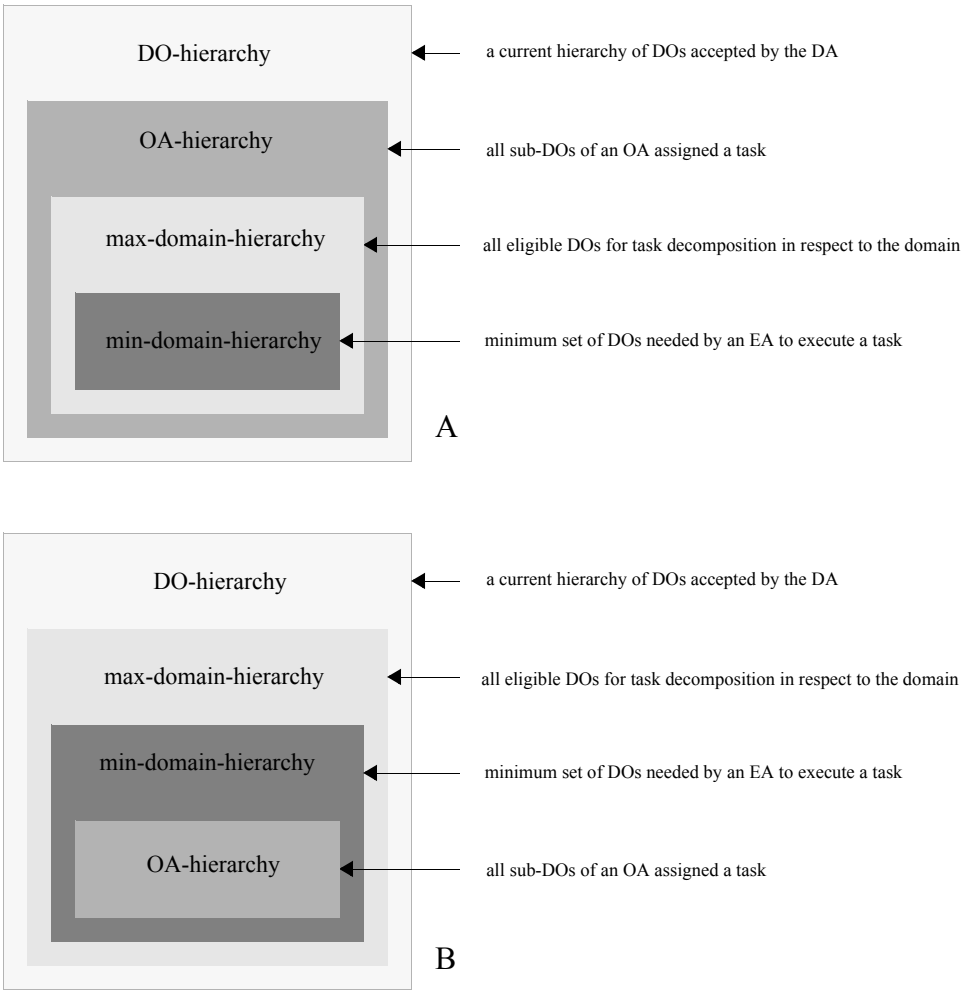


FIGURE 5.7.

Special Case Relation Between Hierarchies:
A) Case 1: min-domain-hierarchy < max-domain-hierarchy < OA-hierarchy.
B) Case 2: OA-hierarchy < min-domain-hierarchy < max-domain-hierarchy

3. Default: A Default_Skip$_{list}$ is compiled if no skip$_{list}$ is provided by either the P_Domain protocol or by the DA. The Default_Skip$_{list}$ includes all the classes of the OA-hierarchy and excludes the classes of the min-domain-hierarchy. For instance, in Figure 5.1, in addition to the Building-DO class all classes below the Block-DO and leaf-DOs (in the spatial branch not in the construction categories branch) would be included in the skip$_{list}$.

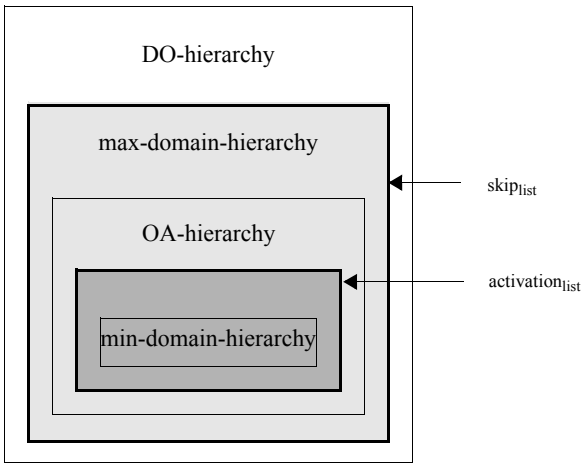


FIGURE 5.8.

Relation Between a Skip$_{list}$ and an Activation$_{list}$ (general case).

From the previous discussion we can deduce that:

- An OA-hierarchy is a subset of a provided DO-hierarchy (see Figure 5.6);
- An OA-hierarchy is typically a subset of the max-domain-hierarchy (see Figure 5.6). It is possible that a max-domain-hierarchy be a subset of a OA-hierarchy (see Figure 5.7A). In such cases, all classes between the OA and the domain-hierarchy$_{top}$ should be included in the skip$_{list}$;
- An OA-hierarchy may be a subset of the min-domain-hierarchy (see Figure 5.7B). In such case the activation$_{list}$ may include DOs that are higher in the hierarchy then the OA assigned the task. This presents a case of task decomposition which includes DOs outside the OA-hierarchy.
- The min-domain-hierarchy is a subset of the max-domain-hierarchy;
- The min-domain-hierarchy must exist as a subset in the activation$_{list}$;
- The activation$_{list}$ is a subset of the max-domain-hierarchy;
- The skip$_{list}$ is a subset of the OA-hierarchy, and in most cases is also a subset of the max-domain-hierarchy. In both cases its disjoint from the min-domain-hierarchy (see Figure 5.8);
- If a min-domain-hierarchy = activation$_{list}$ => skip$_{list}$ is maximum;
- If max-domain-hierarchy = activation$_{list}$ => skip$_{list}$ is minimum (may be empty).

Algorithm: ***MakeSkipList***

start

check that DA_Skip_{list} does not include min-domain-hierarchy DOs

1 **if** *Intersection (DA_Skip_{list}, min-domain-hierarchy)* ≠ null

2 get DA validation to $mod\_DA\_Skip_{list}$

3 **return** $mod\_DA\_Skip_{list}$

else

4 **end if**

5 $mod\_DA\_Skip_{list}$ ← *Difference (DA_Skip_{list}, min-domain-hierarchy)*

6 **if** $P\_Domain\_Skip_{list}$ ≠ null

7 $skip_{list}$ ← *Union ($P\_Domain\_Skip_{list}$, $mod\_DA\_Skip_{list}$)*

8 **else** (default: no $skip_{list}$ is provided by the P_Domain protocol)

9 $skip_{list}$ ← *Difference (OA-hierarchy, min-domain-hierarchy)*

10 **end if**

end

Algorithm: ***MakeActivationList***

start

1 **if** $DO_{classification}$ is *in* max-domain-hierarchy

2 $activation_{list}$ ← *Union* (*Difference (max-domain-hierarchy, $skip_{list}$)*, *$DO_{classification}$*)

3 *SortActivationList ($activation_{list}$)* using activation order provided by the domain decomposition protocol => return $sortedActivation_{list}$

4 **else**

5 error message

6 **end if**

end

Algorithm: ***SortActivationList (activation$_{list}$)***

start
1 get P_Domain sorting protocol
2 oldSorted$_{list}$ ← null
3 newSort$_{list}$ ← null
4 **for** each DO in the activation$_{list}$
5 *InsertDOInSortedList (DO, oldSorted$_{list}$, newSorted$_{list}$)*
6 **end for**
7 sortedActivation$_{list}$ ← *Append (oldSorted$_{list}$, newSorted$_{list}$)*
8 **return** sortedActivation$_{list}$
end

Algorithm: ***InsertDOInSortedList (DO, oldSortedList, newSortedList)***

start
1 **if** oldSorted$_{list}$ == null
2 *Insert* (*DO, oldSorted$_{list}$)*
3 **else**
4 **if** first DO of oldSorted$_{list}$ is lower in the activation order
(as provided by the P_Domain sorting protocol)
5 *Insert (DO, oldSorted$_{list}$)*
6 **else**
7 newSorted$_{list}$ ← *Append (first DO of oldSorted$_{list}$, newSorted$_{list}$)*
8 oldSorted$_{list}$ ← *Remove (first DO, oldSorted$_{list}$)*
9 *InsertDOInSorted$_{list}$* (DO, oldSorted$_{list}$, newSorted$_{list}$)
=> return (oldSort$_{list}$, newSort$_{list}$)
10 **end if**
11 **end if**
end

Line 8: Aggregation

Any decomposition of a task to sub-tasks is counter balanced with an aggregation of the sub-results of the execution of the sub-tasks. Similar to decomposition, aggregation is a domain dependent activity that is necessary to the execution of a task by an OA. For instance, the cost of an OA is the aggregation of the cost of the leaf-DOs of the OA-hierarchy (provided that all components of the cost are represented as leaf-DOs in the OA-hierarchy). In addition to the leaf-DOs, the cost of an OA may include the cost of attributes of its sub-DOs (typically, such attributes substitute for sub-DOs that are not explicitly represented as leaf-DOs in the OA-hierarchy). For instance, in Figure 5.1, the cost of the Painting-DO is the aggregation of all Paint-DOs' cost (represented in the hierarchy as a member of the Layer-DO class). If paint is represented as an attribute of other DOs such as Wall-DOs, Ceiling-DOs and not explicitly represented in the hierarchy as Paint-DOs, then the cost of the Painting-DO is the aggregation of the cost of all paint attributes in all DOs. A third possibility is that paint is not even represented as an attribute, in such case, paint may be calculated based on selected DOs total area (such as Wall-DOs and Ceiling-DOs).

Aggregation is not necessarily an addition of sub-results, it may require further computation by the domain-EA of the task in hand. For instance, the aggregation of structural loads of intersecting StructElement-DOs (e.g., sloped beam resting on a column) requires computation beyond mere addition of loads. Therefore, **to aggregate sub-results additional interactions between the OA aggregating the sub-results and the domain-EA are typically required after the sub-OAs return their sub-results to the OA**. Such interactions are guided by the P_Domain aggregation protocol of the that task-domain.

Algorithm: ***Aggregate (sub-results)***

start

1 check sub-DO IDs for repeated sub-results, or check results IDs

2 $\text{sub-results}_{\text{list}} \leftarrow \text{sub-results}_{\text{list}}$ with repeated results removed

3 get P_Domain aggregation protocol

4 *InteractForService (domain-EA)* to evaluate $\text{sub-results}_{\text{list}}$
=> return aggregation results

5 **return** aggregation results

end

Line 7: Validating results

Any task execution or aggregation result must be validated before it is returned to the super-agent which assigned the task. The validation can occur according to various preset validation modes. The validation mode can either be set by the designer or by the domain protocol. Four modes are identified; DA validation, super-OA, self validation and no validation.

Algorithm: ***Validate (result, validation mode)***

```
start
1    if validation mode == ?
2         case 1. ? == DA validation requested => DAValidation (result)
3         case 2. ? == super-OA validation requested
                                                => SuperOAValidation (result)
4         case 3. ? == self validation          => SelfValidation (result)
5         case 4. ? == no validation requested => NoValidation (result)
6    else
7         error message
8    end if
end
```

Algorithm: ***DAValidation (result)***

```
start
1    get DA validation
2    if results are not validated
3         get DA's new values for re-evaluation (or for task re-assignment)
4         if no new values are provided by the DA
5              error message
6         else (DA provided new values)
7              InteractForSerivce (domain-EA)
                    to re-evaluate aggregation results or
8                   re-Assign (evaluation task) to sub-OA
9         end if
10   else (results are validated)
11        return validation
12   end if
end
```

Algorithm: ***SuperOAValidation (result)***

start
1 get super-agent validation
2 **if** results are not validated
3 error message
4 **else** (results are validated)
5 **return** validation
6 **end if**
end

Algorithm: ***SelfValidation (result)***

start
1 *InteractForSerivce (domain-EA)* to evaluate results => return results
2 **if** results are not validated
3 error message
4 **else** (results are validated)
5 **return** validation
6 **end if**
end

Algorithm: ***NoValidation (result)***

start
1 **return** validation
end

5.2.2 Conflict Handling

As explained in section 3.1 of Chapter 3 and illustrated in Chart 7 of Chapter 4, the conflict handling process is mainly composed of two steps; conflict detection and conflict resolution (ignoring conflict prevention).

Within the framework of this thesis, conflict detection is dependent on the OAs informing the DA of potential agents or DO attributes which are interested in the OA attribute value being modified. In an advanced mode of conflict detection, the OA should be permitted to independently activate interested DOs and assign to these evaluation tasks to examine the new attribute value being modified. In such cases, the DA should only be notified upon the recognition of an actual conflict. The conflict resolution is heavily dependent on the interaction among the DA and the OAs involved in the conflict.

Within the framework of this thesis conflict resolution is a series of DA controlled local evaluations conducted by the OAs to examine alternative attribute values. Accordingly, resolving conflict is done through iterations of local evaluation not through direct negotiations among OAs (as argued in Chapter 3).

The conflict detection uses attribute lists of interested agents and attributes from other DOs and within the same DO (see details of compiling an interest list below). For instance, an interest-list (interest$_{list}$ hereafter) of a glazing area attribute for a Window-DO may include

- Agents such as daylighting-EA, cost-EA, thermal-EA and elevation-EA;
- Attributes of DO classes such as coordinates of StructElement-DO, Wall-DO width and height;
- Attributes of the same DO class such as Window-DO glazing-type, window-type and shading device types.

If the Window-DO glazing is modified each of the agents or DO attributes above may be affected in various capacities. A Window-OA provides the interest$_{list}$ to the DA, the DA chooses the more critical members of the list to examine in order. Accordingly, the DA activates the necessary DOs and assigns evaluation tasks to examine the new glazing area. If the evaluations are satisfactory, the DA validates the conflict handling session and accordingly validates the modified glazing area.

How is the interest$_{list}$ compiled?

Each attribute of a DO has an interest$_{list}$ associated with it upon creation. The selection of the members of each list is done when a DO class is defined[6]. The list includes possible interested DOs and agents from those provided in the environment upon the initiation of a session.[7] A DO attribute or agent may exist in the list even if it is not present in the environment in the current session.

It is possible to dynamically add more members in an interest$_{list}$ during a session. The addition of new members can be automated or left to be solely a property of the DA. For instance, a constraint between two attributes provides a reasonable basis for the automation of interest$_{list}$ registration. That is, **the constraint parties must be registered on each others interest$_{lists}$**. For instance, if a constraint links the glazing area attribute to the width attribute of a Window-DO, the glazing area attribute must be registered as a member of the width attribute interest$_{list}$ and vice versa. On the other hand **agents or DO attributes that are not linked to an attribute by constraints should be explicitly added by the DA (if needed)**.

When an attribute value is being modified and checked for potential conflict each member of the interest$_{list}$ is a candidate for conflict check. Which members are selected for conflict checking is left to the DA preferences. Two types of members may register in the interest$_{list}$; domain-EAs and DO attributes. A DO attribute can either be of the same DO or of another DO. This constitutes three cases as demonstrated by the following examples:

1. a daylighting-EA should be in the interest$_{list}$ of any Window-DO glazing attribute;
2. a Wall-DO thermal mass attribute should be in the interest$_{list}$ of its own Wall-DO thickness attribute;

6. The dependency of a DO attribute on the existence of other attributes, DO classes or agents may seem to violate a fundamental principle of object oriented design, however, the interest$_{list}$ is merely used to inform the DA of the potential members of the environment that may have conflict with the attribute value being modified. This cannot be considered as an explicit dependency among objects. An interest$_{list}$ has no implications on any external object.
7. No new classes may be admitted at run time. This may seem to contradict the earlier notion (discussed in Chapter 3) of the Composite-DO which is compiled out of a collection of DO classes. A Composite-DO can be created during a session but cannot be recognized during a session.

3. a Door-DO thickness attribute should be in the $interest_{list}$ of the Wall-DO thickness (where the Door-DO is located).

The first example represents the general case and can be applied to any instances of a Window-DO (i.e., daylighting-EA can be included by default in any Window-DO glazing area attribute). Therefore, an $interest_{list}$ registration of this type can be made as a part of the DO class definition upon creation of the class. In the second example interest is among two attributes of the same DO, this is a specialization of first case. The $interest_{list}$ registration can also be made in the DO class definition upon creation of the class. The third example represents a more restricted specialization case since the Door-DO is registered in a specific Wall-DO instance and cannot be generalized (i.e., only in the Wall-DO where the Door-DO is geometrically located). In this case, the registration in the $interest_{list}$ may also be in the class definition but attached with a constraint to limit it to a certain Wall-DO. The $interest_{list}$ registration in this case can also be done in run time by the DA.

To avoid redundancy in representation of attribute relations, $attribute_1$ should not be in the $interest_{list}$ of $attribute_2$ within a context if there is a constraint linking $attribute_1$ to $attribute_2$ within the same context.

A domain-EA registered in the $interest_{list}$ of an $attribute_1$ of OA_1 may participate in a conflict handling session about a new value for $attribute_1$ if the domain-EA:

- is currently providing service to OA_2 using the same $attribute_1$. OA_2 can either be another OA of the same DO performing a different task simultaneously, or an OA of another DO performing a task that require the use of $attribute_1$. In such case, OA_2 must run another evaluation session interacting with the domain-EA to examine the new value of $attribute_1$.
- previously provided service to an OA_2 regarding the $attribute_1$. This case may only be considered if the domain-EAs are capable of keeping the history of services provided to OAs during a session (or previous sessions). If OA_2 is no longer activated then the domain-EA must first request the activation of OA_2 DO.

Each member in the $interest_{list}$ is coupled with an "**interest context**". The context conveys the reason(s) for which this member is interested in this attribute. **A member of an attribute $interest_{list}$ is activated to examine the suggested attribute value in respect to the interest context of that member**)

.

TABLE 5.1. An example of an $Interest_{list}$ of a DO Attribute

an $interest_{list}$ of Wall-DO_1 thickness attribute				
#	**DO-inst/Agent**	**interested attribute**	**interest context**	**interact/activate**
1	Acoustic-EA	N/A	Room-DO_1 noise level	Room-DO_1
2	Wall-DO_1	thermal time lag (hr)	Room-DO_1 thermal comfort	Thermal-EA
3	Door-DO_1	thickness (in)	Wall-DO_1 assembly (cont.)	Geometry-EA

In Table 1 an $interest_{list}$ of a Wall-DO thickness attribute may include members such as: member "1" is a Domain-DO, member "2" is an attribute of the same DO, and member "3" is an attribute of another DO. Each member has its own interest context and recommended party to interact with in connection with this context. The interest context may evolve around a third party (i.e., DO). In such cases, activation of this party may also be required (as in member "1" and "2").

How is the $interest_{list}$ sorted with respect to the degree of importance of conflicts?

The order of an $interest_{list}$ should be subject to change according to the task in hand. To do so, the question to be addressed would be; how important is a potential conflict with respect to the task in hand? To establish a scale to measure the importance of a DO attribute or an agent in respect to the attribute value being modified a weight mechanism is needed. An OA-based environment can certainly benefit from such a mechanism and can easily adopted it. There may be various ways of establishing such a mechanism, but these are all beyond the scope of this framework. Therefore, in this framework, **the DA is the only reference to sorting the $interest_{list}$**.

Controlling the conflict handling dependencies

The recursive effect of changing a single attribute value of a DO on other DOs may reach deeply nested levels causing dependences and infinite loops of evaluations and conflict handling sessions. However, such dependencies should be prevented in any evaluation session (see Section 3.1). As an interface aid to the DA it may be of great advantage to display a graph of the possible dependencies (as a warning) before confirming a modification of an attribute value. It is also possible to design an automated mechanism to allow the

designer to control the depth of nested conflict checking and to reduce the number of participants in a conflict. Developing such a mechanism is a research topic on its own right. Within the framework of this thesis, to reduce conflict checking dependencies, **a DO that is activated during a conflict handling session may not trigger other conflict handling sessions**.

Constraint Satisfaction and the interest$_{list}$ mechanism

The notion of an attribute interest$_{list}$ may project a conflict with the notion of a constraint network which may link multiple attribute values of DOs. If a Window-DO height is linked to the width by a proportion constraints, say, for aesthetic purposes, then changing the glazing area for a daylighting would effect such constraint. A constraint satisfaction mechanism would attempt to modify the height and width to maintain the specified proportions under the new glazing area. Doing so, the width and height attributes of the Window-OA would trigger a series of conflict checks that are independent from the initial conflict checks triggered by the change in the glazing area. A constraint satisfaction mechanism may work as a drive for modifying attribute values which, in turn, triggers a series of conflict checking sessions using the interest$_{list}$ mechanism. Therefore, the two mechanisms may work separately or conjointly and there is no conflict among these.

Algorithm: ***ConflictChecking&Handling***

start

1 provide the $interest_{list}$ of $attribute_1$ (being modified) of OA_1 to the DA[8]

2 prompt the DA to *SortInterest$_{list}$ (interest$_{list}$)* add or remove members
=> return $sortedInterest_{list}$

3 **for** each member of the $sortedInterest_{list}$

4 **if** member == ?

5 case 1. ? == domain-EA &

5 domain-EA is performing $task_2$ for OA_2 about $attribute_1$
=> *DomainEAConflictSession (OA_1, OA_2)*

6 case 2. ? == domain-EA & domain is in active mode[9]
=> *ActiveDomainEAConflictSession (OA_1)*

7 case 3. ? == DO_4 => *AttributeConflictSession (DO_4, OA_1)*

8 case 4. ? == $attribute_2$=> *AttributeConflictSession (DO_1, OA_1)*

9 **else**

10 error message

11 **end if**

12 **end for**

end

8. To deal with the conflict cycle effect (as explained in section 3.1), the algorithm should include steps to mark the visited attribute (but only if its value changes as a result of the conflict session). In this way, conflict cycles can be detected; however, the marking of a visited attribute may not be straightforward and requires further investigation that is not covered in this dissertation.

9. A Domain-EA can be in passive or active mode. If in active mode and it is a member of the $interest_{list}$ of an attribute it would automatically evaluate any new attribute value for this attribute upon modification.

Algorithm: ***DomainEAConflictSession (OA_1, OA_2)***

start

1 *Assign* OA_2 an evaluation task to examine the new value of $attribute_1$
in respect to the current task of OA_2 => return result

2 **if** results are validated

3 **return** conflict session validated results

4 **else** (at least one of the results is not validated)

5 *EvaluateConflictAlternatives (OA_1, OA_2)*

6 **end if**

end

Algorithm*: ActiveDomainEAConflictSession (OA_1)*

start

1 *Activate* (*DO)*[10] (a duplicate of OA_1 to conduct the new evaluation session)
=> return OA_2

2 *Assign* OA_2 an evaluation task to examine the new value of $attribute_1$
in respect to the context of the domain-EA interest
=> return results

3 **if** results are validated

4 **return** conflict session validated results

5 **else** (at least one of the results is not validated)

6 *EvaluateConflictAlternatives (OA_1, OA_2)*

7 **end if**

end

10. If the OA is allowed to execute more than one task simultaneously this step would be skipped and OA_2 should be replaced by OA_1in the next step. This corresponds to Chart 3 of Chapter 4.

Algorithm: ***AttributeConflictSession(DO, OA_1)***

start

1 **if** DO ≠ DO1

2 *Activate (DO)* => return OA_2 (an OA of a different DO)

3 **else** (DO == DO1, local attribute)

4 *Activate (a clone of DO_1)* => return OA_2 (a duplicate of OA_1)

5 **end if**

6 *Assign* OA_2 an evaluation task to examine the new value of $attribute_1$ in respect to the context[11] of $attribute_2$ interest (as in the $interest_{list}$) => return results

7 **if** results are validated

8 **return** conflict session validated results

9 **else** (at least one of the results is not validated)

10 *EvaluateConflictAlternatives (OA_1, OA_2)*

11 **end if**

end

11. This task assignment may, in turn, require OA_2 to activate other DOs to perform sub-tasks (according to the interest context). During such process DO_1 itself may need to be activated more than once. However, if the task needed (according to the interest context) is of the same task-type of the original task of OA_1 then OA_1 can be reassigned the same task instead of activating a duplicate OA.

Algorithm: ***EvaluateConflictAlternatives (OA_1, OA_2)***

start

1 get alternative attribute values from DA
2 **if** alternative attribute values are provided
3 *Assign* OA_1 its current evaluation task with new value
=> return results
4 *Assign* OA_2 its current evaluation task with new value
=> return results
5 **if** both results validated
6 **return** conflict session validated results
7 **else** (at least one of the results is not validated)
8 *EvaluateConflictAlternatives (OA_1, OA_2)*
9 **end if**
10 **else** (no alternative values are provided)
11 error message
12 terminate conflict handling session
13 **end if**

end

Line 24: Updating

After the results of both task execution and conflict handling are validated, the OA has to update its original DO information. Since conflict handling process is intended to insure the integrity of the information of the DO being modified in respect to the interested DO attributes and agents, updating attribute values is preceded by two other checks to insure integrity of the DO information in respect to constraints and DO clones.

- Firstly, checking the constraints network. The DO attributes being updated may be linked with other DO attributes by constraints. Each constraint may be associated with other constraints of DO attributes through a propagation. Hence, any attribute value update requires a series of constraint satisfaction checks to insure that no other constraints are violated by the current update. If a constraint is violated the DA must be notified and validates the update. As the case with the conflict handling, the DA involvement is necessary to control dependencies.
- Secondly, to update the information of any clones of the DO which are used simultaneously by other OAs. In particular, those OAs which are performing tasks on the same DO attribute being modified.

Algorithm: ***Update (new-values)***

```
start
     # check for constraints satisfaction with other DOs #
1    for each constraint of the attribute being updated
2         ApplyConstraint (new-value)
3              for each propagated constraint repeat check
4                   Activate DOs involved in the constraint
                      (to evaluate the new values[12]) => return OAs
5                   for each activated OA
6                     ApplyConstraint (new-value)
7                   for end
8              end for
9    end for
10   get DA validation for update
     # check for OAs that are currently using the DO being updated#
11   if there is more than an OA for the same DO
12        for each OA of the same DO
13             send an update message of the new values
14        end for
15   else
16   end if
end
```

Algorithm: ***CheckConstraint (constraint, new attribute value)***

```
start
1    ApplyConstraint (new-value)
2    if new value does not satisfy constraint
3         error message
4    else
5         return constraint application result
6    end if
end
```

12. Conflicts and updates for the activated DOs must be controlled by the DA to avoid dependencies & nested checks.

5.3 Examples of P_Domain protocols.

As demonstrated by the algorithms of this chapter, the OA protocols are mainly domain dependent. This section provides an examples of domain depend OA protocols such as decomposition, sorting and aggregation. The P_Domains provided present variables that are required by the P_TaskType algorithms (e.g., P_TT_Evaluation, P_TT_Recommendation, see Figure 6.1 of Chapter 6) to execute an assigned task (see also lines 2 and 8 in the task evaluation algorithm).

5.3.1 Cost Evaluation Protocols

Cost evaluation decomposition protocol

- $skip_{list}$: N/A
- min-domain-hierarchy: all leaf-DO classes (of the construction branch);
- max-domain-hierarchy: [$domain\text{-}hierarchy_{top}$ -- $domain\text{-}hierarchy_{bottom}$];
- $domain\text{-}hierarchy_{top}$: Site-DO;
- $domain\text{-}hierarchy_{bottom}$: leaf-DO level (of the construction branch).

Cost evaluation sorting protocol

- decomposition order: [Site-DO, Building-DO & LandScElement, Block-DO, and Leaf-DOs (of the construction branch)];
- typical evaluation order: top-down;
- special evaluation order: $DO_{classification}$ is first in level.

Cost evaluation aggregation protocol

- aggregation-type:
  Site-DO (and below) => Request service from cost-EA

5.3.2 Structural Analysis Protocols

Structural analysis decomposition protocol

- $skip_{list}$: Site-DO, all leaf-DO classes excluding the StructElement-DO class;
- min-domain-hierarchy: [BFloor-DO, StructElement-DO];
- max-domain-hierarchy: [$domain\text{-}hierarchy_{top}$ -- $domain\text{-}hierarchy_{bottom}$];
- $domain\text{-}hierarchy_{top}$: Building-DO;
- $domain\text{-}hierarchy_{bottom}$: StructElement-DO.

Note: In a Structural recommendation protocol min-domain-hierarchy does not include StructElement-DO (only the BFloor-DO is necessary).

Structural analysis sorting protocol

- decomposition order: [Building-DO, Block-DO, VZone-DO BFloor-DO, HZone-DO, StructElement-DO];
- typical analysis order: carried loads (top-down order, higher loads are added to the lower ones, e.g., higher BFloor-DOs must be analyzed first);
- special analysis order: suspended loads (bottom-up order, lower loads are added to the higher ones, e.g., lower loads of a suspended bridge must be analyzed first).

Structural analysis aggregation protocol

- aggregation-type:
  Site-DO => listing (e.g., table)
  Building (and below) => Request service from structural-EA

5.3.3 Daylighting evaluation protocols

Daylighting evaluation decomposition protocol

- $skip_{list}$: LandScElement-DO, all leaf-DO classes excluding the Opening-DO class and the Layer-DO class;
- min-domain-hierarchy: [Room-DO, Wall-DO, Ceiling-DO, Floor-DO, Opening-DO];
- max-domain-hierarchy: [$domain\text{-}hierarchy_{top}$ -- $domain\text{-}hierarchy_{bottom}$]
- $domain\text{-}hierarchy_{top}$: Building-DO;
- $domain\text{-}hierarchy_{bottom}$: Opening-DO.

Daylighting evaluation sorting protocol

- decomposition order: [Building-DO, Block-DO, VZone-DO BFloor-DO, HZone-DO, Room-DO, Opening-DOs];
- typical evaluation order: top-down;
- special evaluation order: must exhaust all Room-DOs with shared Opening-DOs.

Daylighting aggregation protocol

- aggregation-type:
  BFloor-DO (and above) => listing (e.g., table)
  Room-DO (and below) => Request service from daylighting-EA

6 Implementation Design

6.1 Object Oriented Implementation

Originally, the implementation of the OA model started within a rule-based development environment, namely CLIPS 4.3. The entire activation mechanism was developed on top of the ICADS project [Pohl 92] and [Myers 93]. DO attributes were implemented as set of facts. DO facts were asserted in the global CLIPS fact list when needed. The protection of such DO data required an additional implementation considerations. An OA was implemented as a collection of its DO facts in addition to a set of related protocols, implemented as rules, all were copied into an OA file. The file was created and configured in the environment at run time. New facts about any newly created OA were asserted into the fact list so that all agents of the environment might interact with it. Though it was possible to create and configure new OAs at run time, OA protocols had to be loaded during the initiation of a session. Therefore, the local protocols of an OA (a set of CLIPS rules) were mere reference to a subset of the globally protocols loaded during initiation. To overcome such shortcoming it was necessary to be able to initiate a new CLIPS session for each created OA. In turn, this requires a message passing system to link distributed CLIPS sessions each containing independent fact list. Using such a message handling system, facts were asserted simultaneously across distributed fact lists to facilitate interaction between distributed agents. OA protocols were loaded on run time and were not necessarily global. An OA executes its protocols locally within its own CLIPS session and only communicates selected execution results to the appropriate fact lists. Certainly, distributed CLIPS sessions each with a smaller number of facts and rules, made the control of the execution flow an easier implementation task. However, such distribution required another layer of rules

for update and truth maintenance of distributed data that was continually being modified.

A later version of CLIPS (namely CLIPS6.0) provided a rule-based object-oriented test bed for the OA model. An OA prototype of the activation mechanism was developed using object attributes and message-handlers as the object methods. The problem of OA protocols was now handled in a different fashion. With classes and inheritance an OA could be an instance of one or a set multiple OA classes and may inherit all its protocols from its super-classes. A DO attributes is no longer public in the fact list, it is rather protected or private. Objects and their attributes could be used for pattern matching to execute rules. An addition that allowed a more sophisticated yet better controlled OA model to be implemented. However, fact lists which could grow exponentially large drastically slowed the execution of tasks when hundreds (or even thousands) of objects were involved in a single CLIPS session. In addition, parts of the OA model were more appropriately implemented in a procedural language. Though, CLIPS provided alternative ways of either implementing procedural code or integrating exported code, it was more efficient to implement the model in a uniform development environment.

A later decision to design the implementation of the OA model using a complete object oriented development environment (OODE), namely C++, was made to take advantage of various aspects. In addition to execution speed, uniformity of code, complete data encapsulation and protection, the separation of local object methods from the message handling system, the ease of controlling the execution flow, the ability to establish more complex object compositions, the ability to use graphical object modeling tools (such as OMTool) to conceptually represent the OA model (which can then be translated into C++ code), the availability of more object libraries such ET++ all contributed to such implementation design shift.

The reminder of this chapter focuses on the development of the general object model for an OA-based design environment and an exemplary domain specific object model. The objects, of the general model, and their relation are defined and detailed. In addition, the implementation of the activation process of a DO is detailed to illustrate what an OA would look like (the collection of objects that constitutes an OA) once it is realized in the environment. This detailed activation process is based on Chart 1 of Chapter 5, and can directly be transformed into a DO activation algorithm. Therefore, the activation algorithm is omitted from this to avoid redundancy.

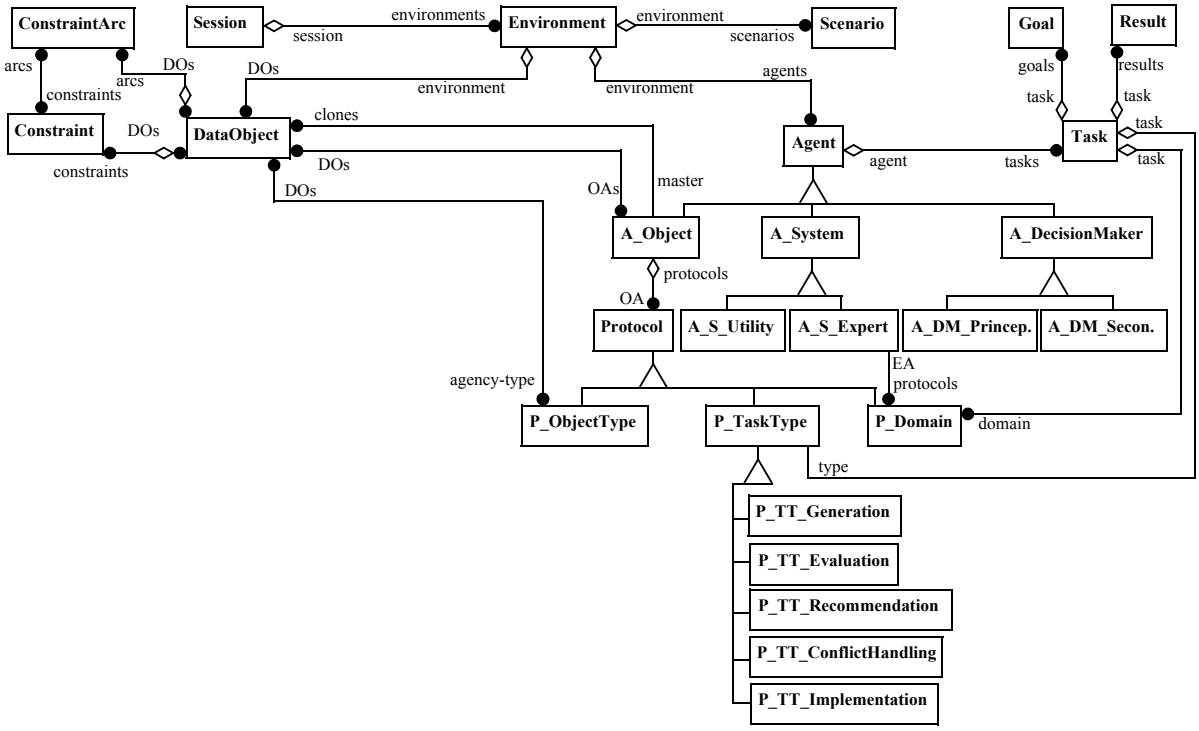


FIGURE 6.1.

A general object model of an OA environment

6.2 The Object Models

Using an OODE, object models are necessary to represent the architecture of any OA-based environment. In this chapter I describe object models that are developed using the Object Modeling Tool OMTool [Rumbaugh, 91]. Two sets of models are developed; the first set provides a general model that is designed to accommodate different design domains with examples of domain specific objects (a set of related DOs); the second set provides a domain specific model (an alternative hierarchies and relations among related DOs).

6.2.1 The general object model

The general object model, shown in, Figure 6.1, is structured around the 'Environment' class, which is the container of two major classes; 'Agent' and 'DataObject'. The 'Environment' class is a constituent of the 'Session' class and contain the 'Scenario' class. All three are created for the purpose of version management.

The ‘Agent’ class is the super-class of a classified hierarchy of different agent classes including the OAs, which are represented by the ‘A_Object’ class (the focus of this thesis). The notion of agency is represented through task related classes such as ‘Goal’, ‘Result’ and ‘Task’. The latter class is linked to the ‘Agent’ class since, in this frame work, **the ability to execute tasks constitutes agency**. On the other hand, the ‘A_Object’ class contains a classified hierarchy of interaction and problem solving protocols that are necessary for an OA to execute tasks. The relation between the ‘Task’ class and the ‘P_TaskType’ class and the ‘P_Domain’ class are also crucial to the notion of agency. Tasks (not OAs) manage the loaded domain and task-type protocols. This allows an OA to **execute more than one task simultaneously**, another property of agency.

The ‘DataObject’ class is the super-class of any hierarchy of the domain DOs. It holds links to ‘Constraint’ and ‘ConstraintArc’ classes. Both classes, along with the DataObject class, are designed to accommodate constraints and constraint propagation.

The relation between the ‘DataObject’ class and both ‘A_Object’ class and ‘P_ObjectType’ class in addition to the relation between the ‘A_S_Expert’ class and the ‘P_Domain’ class are important to OAs. Parties of both relations are paired. That is, each sub-class of these classes is associated with a sub-class in the other side of the relation. Each DO, which is a sub-class of ‘DataObject’, is paired with a set of object-type protocols, a sub-class of ‘P_ObjectType’. Each domain-EA, which is an instance of ‘A_S_Expert’ is paired with domain protocols, an instance ‘P_Domain’.

A brief explanation of the object relation is necessary before discussing the unique relations of the general model.

Object Relations

The general object model, shown in Figure 6.1, comprises a collection of objects with **aggregation**, **association** and **generalization** relationships. A relation is graphically represented by a line connecting two labeled boxes (where each box represents a class).[1] The nature of the relationship may be explicitly represented

1. Four types of association relationships are provided in OMT; one → one, one → many, many → one and many → many.

using roles of a relation, or through the attributes and methods of the two classes involved in the relation.[2]

Association

An association relationship is a link between two classes. There may exist more than one association relationship between the same two classes, each of which are identified by different roles. For example, in Figure 6.1, the 'A_Object' and 'DataObject' classes are linked with two association relationships. The first is a 'one → many' relation marked by two roles; 'clones' and 'master'. An instance of an 'A_Object' class (i.e., master) may have links to multiple clone instances of the 'DataObject' class (i.e., clones).[3] The second is a 'many → many' relationship marked by two roles; 'DOs' and 'OAs'. An instance of the 'DataObject' class, may have links to multiple instances of the 'A_Object' class (i.e., OAs) performing different tasks simultaneously. In addition, an instance of the 'A_Object' class may have links to multiple instances of the 'DataObject' class, (i.e., DOs).[4]

Aggregation

An aggregation relationship is marked by a diamond shape attached to the class box indicating the 'whole' side of the relation. Multiplicity is marked by a filled circle attached to the class box indicating the 'many' side of the relation.[5] For

2. OMTool does not generate specific code to represent the nature of the link, therefore, it should be explicitly described in the attributes and methods of the objects involved in the relation. This is applicable to association and aggregation relationships. When generating C++ code OMT does not make a distinction between an association or an aggregation relationship. For instance, in an aggregation relationship, if the super-class, which is the 'container', is deleted the sub-class, which is the 'constituent', are not deleted (as it conceptually should). The deletion of the sub-classes should be explicitly represented by a method in the super-class. On the other hand, a generalization relationship is distinct from aggregation and association since inheritance is represented in the OMTool generated C++ code.
3. For both roles 'clone' and 'master', when OMTool generate C++ code it creates an attribute named 'clones' of the type SeqCollection in the 'A_Object' class, and an attribute named 'master' in the 'DataObject' class. The methods needed to access such attributes such as 'addClone', 'getClones' and 'removeClone' are not automatically generated and must be explicitly developed.
4. The multiplicity here is intended to permit composite OAs. A composite OA represents more than one DO at the same time.
5. Four types of the aggregation relations are provided in OMT some of which are utilized in the general model; one part → one whole, one part → many whole' many parts → one whole, many parts → many whole.

example, in Figure 6.1, the two classes 'Environment and 'Agent' are connected by an aggregation relation of the type 'many parts → one whole', where the 'Environment' has many 'Agents' (or many 'Agents' are part of the Environment).

Conceptually, an aggregation relationship is a specialization of an association relationship. It represents the conventional relationship 'part-of/has' also known as 'constituent/container'.

Generalization

A generalization relationship represents the relation sub-class/super-class, and is marked by an triangular shape on the line connecting the super-class and its sub-classes. Hierarchies are defined through generalization. For example, in Figure 6.1, the 'Agent' class is the super-class of the 'A_object', 'A_System', and the 'A_DecisionMaker' classes. The 'A_DM_Princepal' and 'A_DM_Secondary' are sub-classes of the 'A_DecisionMaker' class and, therefore, they belong to the same hierarchy of the 'Agent' class. Hierarchies constitute inheritance of object attributes, methods and relations. Inheritance may be constrained locally within each object in the hierarchy. Attributes and methods may be made public, protected or private as explained below in this section.

The general object model comprises the following object classes. All object classes are identified as either concrete or abstract.

Session

An instance of the 'Session' class would contain information needed to manage the decision making session. Management of a session includes saving retrieving, augmenting, freezing, deleting a session with all its current environments. A DA may compile one or more environments for decision making experimentation within the same session. 'Session' is a concrete class.

Environment

An instance of the 'Environment' class would contain the main players needed for decision making during the current session, such as domain agents and domain DOs. A DA initializes an environment and load the appropriate agents and DOs and initiate a scenarios where tasks can be assigned. Many environments can be compiled within the same session and many scenarios can be managed for any single environment. 'Environment' is a concrete class.[6]

Scenario

An instance of the 'Scenario' class holds the information of a sequence of decision making events (the order of activating and deactivating agents and DOs, task assignments and execution). 'Scenario' is a concrete class.

Agent

This class contains all possible agent classes within an environment. The common behavior of an agent is represented in this class. An agent may have multiple tasks to execute each with a goal to accomplish and a result for each task. 'Agent' is an abstract class.

Task

An instance of the 'Task' class holds information about an assigned task. Such information as task-type, task-domain, task-focus, context and any related variables needed for the execution of the task. It also holds links to; the super-agent (who assigned the task), the sub-agents (if the task is decomposed and delegated to other agents). The 'Task' class is linked to the domain protocol 'P_Domain' class and to the task-type protocol 'P_TaskType' class. 'Task' is a concrete class.

Result

An instance of the 'Result' class holds information about the result of the execution of the assigned task. 'Result' is a concrete class.

Goal

An instance of the 'Goal' class holds information about what need to be accomplished as a result of the execution of a task. Such information can be used as an evaluation criteria against the execution results. An additional set of classes such as specifications or functional requirement classes can expand the notion of the 'Goal' class. However, within the scope of this work the functions of such classes would be incorporated as methods and attributes within the 'Goal' class. In other words, the 'Goal' class holds the context of the assigned task. 'Goal' is a concrete class.

A_DecisionMaker

This class contains all possible DM classes within an environment. A DM

6. Instantiation is permitted only in concrete classes.

initiate sessions, compile environments, manage agent interactions, experiment with various scenarios, assign tasks, and evaluate results. The common behavior expected from a DM is represented in this class. 'A_DecisionMaker' is an abstract class.

A_D_Principal

An instance of the 'A_D_Principal' class holds information about a principal DM who is granted all possible authorities and capabilities as the main agent in the environment. 'A_D_Principal' is a concrete class.

A_D_Secondary

An instance of the 'A_D_Secondary' class holds information about a DM who is given limited authorities and capabilities as an agent in the environment. 'A_D_Secondary' is a concrete class.

A_System

This class contains all possible SA classes, namely all UAs that are essential to the environment, and all EAs necessary for the execution of various tasks. The common behavior expected from an SA is represented in this class. 'A_System' is an abstract class.

A_S_Utility

An instance of the 'A_S_Utility' class is a UA which is environment specific agent such as a communication-UA, a configuration-UA, or a CAD-UA. Such agents are provides the infrastructure of any design environment. Replacement of such agents may effect the architecture of the entire system. However, it is expected that different domains may require different behavior from some of these agents. Therefore, a DM should be able to dynamically tune the behavior of any of these agents for each session. 'A_S_Utility' is a concrete class.

A_S_Expert

An instance of the 'A_S_Expert' class is an EA which is a domain specific application (an executable entity) such as a daylighting-EA, a cost-EA or a geometry-EA. Such agents are loaded or unloaded in an environment (initially) upon DA request. Each domain EA is coupled with a set of protocols to facilitate the execution of task of this domain by the OAs. This is represented by an explicit relationship between the 'A_S_Expert' class and the 'P_domain' class. 'A_S_Expert' is a concrete class.

A_object

An instance of the 'A_object' class is an OA which is a temporal agent of DO performing an assigned task (Figure 6.7). This class contains the 'Protocol' class. During the course of executing an assigned task, an OA loads the object-type protocols 'P_ObjectType' of the same OA-type to interact with other agents in the environment. 'A_object' is a concrete class.

Protocol

This is the super class of all the protocol classes. The sub-class of this class contain the interaction and the problem solving protocols needed by an OA to perform a task. 'Protocol' is an abstract class.

P_ObjectType

This is the super class of all the DO-coupled protocol classes. Each DO in the environment must be coupled with a 'P_ObjectType' sub-class (of the same type) within the same environment. A subclass of this class contains interaction instructions for a specific DO-type (e.g., building-DO, room-DO). Such interaction protocols carries what can be communicated with this DO-type. This, in turn, constitutes the behavior expected from an OA amongst the environment agents with this DO-type. An instance of the coupled P_ObjectType sub-class is acquired upon creation of an OA. 'P_ObjectType' an abstract class.

P_TaskType

This is the super-class of all protocols related to the task-types that can be performed in the environment. Types of tasks may vary according to the environment. In a design environment task-types may be 'evaluation', 'recommendation', 'generation', 'implementation' and 'conflict handling'. For example, an evaluation protocol may be described as the general execution plan to be used by an OA to perform an evaluation task (see Chart2, Chapter 5 and Section 6.2.1). An instance of a sub-class of the 'P_TaskType' is loaded after the OA is created and assigned a task by its super-agent. An evaluation protocol should be independent from both task-domain and DO-type (of the OA performing the task). However, the 'P_Domain' and 'P_ObjectType' should feed into a 'P_TaskType' (as variables or functions) during the execution of a task (see Section 6.2.1). There may exist various types of evaluation protocols in the same environment, and ideally, the DM should be able save modified versions of a 'P_TaskType' protocol as instances. 'P_TaskType' is an abstract class.

P_Domain

An instance of the 'P_Domain' class holds information about the task-domain (e.g., cost, structural). A 'P_Domain' object contains instructions that enables an OA to interact with the appropriate EA. It also contains instructions of how an OA can decompose a task or aggregate the results when needed (see Section 6.3). An instance of any sub-class of the 'P_Domain' is loaded by the 'Task' object after the OA is created and assigned a task by its super-agent. Each domain-EA in the environment must be associated with a set of 'P_Domain' objects. 'P_Domain' is a concrete class.

DataObject

This is the super class of all DO classes within an environment. Various domain specific DO classes (architectural, structural, mechanical, etc.) may be added as a sub-class of the 'DataObject' class. Each DO is a representation of a real world object (e.g., a wall or a window in architectural domain). As mentioned earlier, each DO class in the environment is associated with an OA class of the same type. This association is represented by a relation between the 'DataObject' class and the 'A_Object' class. The behavior expected from an activated DO (i.e., an OA) is reflected by the execution of the its protocols. Therefore, there is an explicit relation between the 'DataObject' class and the 'P_ObjectType' class, which indicates that each DO in the environment is coupled with a set of protocols that is specific for such DO-type. 'DataObject' is an abstract class.

Constraint

An instance of the 'Constraint' class represents a constraint on a single attribute value of a DO. A 'Constraint' object mainly has two attributes; an upper and a lower bounds of the attribute value. For instance, a Door-DO width attribute ***DW*** may be represented using a 'Constraint' object *Const1* with lower bound attribute $LB = 2$ ft. and with upper bound attribute $UB = 6$ ft., or:

$$2ft \geq DW \geq 6ft$$

'Constraint' is a concrete class.

ConstraintArc

An instance of 'ConstraintArc' class represents a constraint relation between;

- Two attribute values, which may belong to the same DO or to two different DOs;

- An attribute value and another constraint arc;
- Two ‘Constraint’ objects;
- Two ‘ConstraintArc’ objects.

A ‘ConstraintArc’ object has one or more logical expressions or mathematical equations that represent the constraint between the two ends of the arc. For instance, a proportion between a Wall-DO width attribute ***WW*** and a Door-DO width attribute ***DW*** may be represented using a ‘ConstraintArc’ object ***ConstArc1*** which holds an expression such as:

$$WW \text{ Š } DW$$

Using ConstraintArc objects allow for constraint propagation among DO attributes. An attribute value may depend on the evaluation of another ConstraintArc. For instance, the Wall-DO width ***WW*** may depends on the Door-DO width. In such case ***ConstArc1*** would link the Wall-DO width attribute ***WW*** with constraint object ***Const1*** of ***DW*** (and not to the attribute ***DW*** itself). ‘ConstraintArc’ is a concrete class.

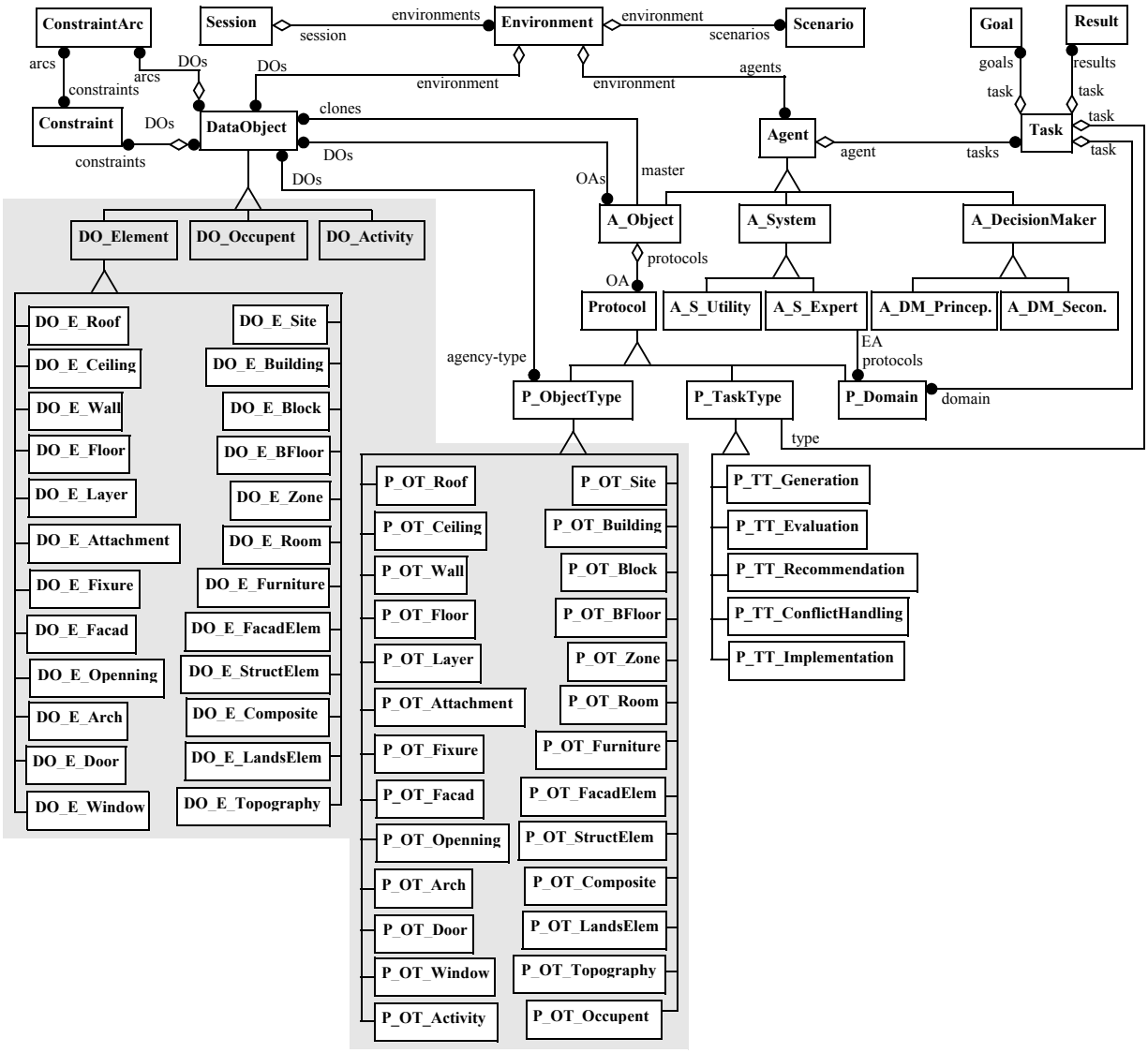


FIGURE 6.2.

An object model of an architectural environment.

6.2.2 A domain specific object model

Two domain specific object models are presented in this section. The first is for architectural design as shown in Figure 6.2, while the second is for structural engineering as shown in Figure 6.3. In each model, two sets of concrete classes are added to the general object model of Figure 6.1. The first set is a collection of domain specific DOs added as sub-classes of the 'DataObject' class, while the second set is a collection of related interaction protocols added as a sub-classes to

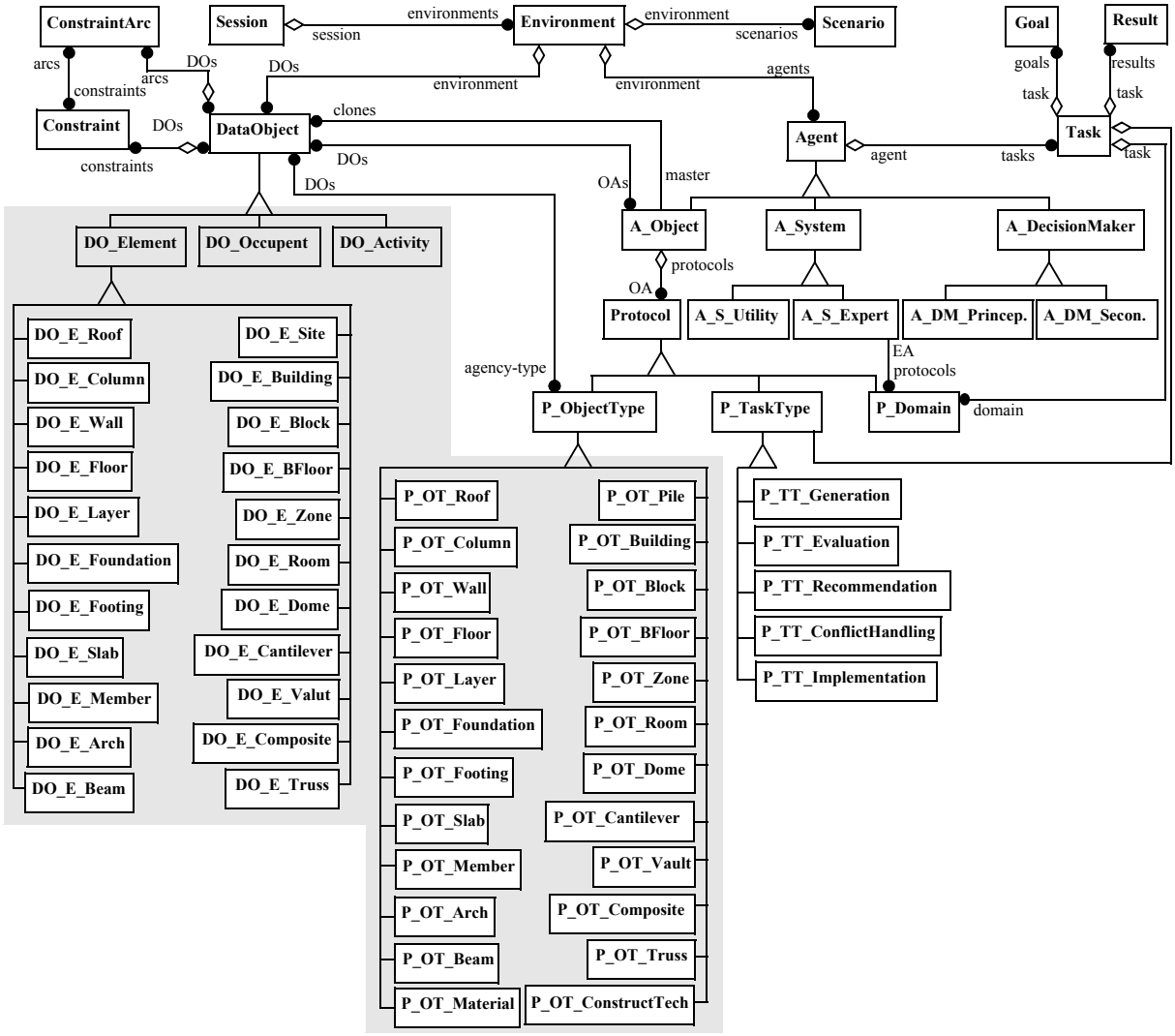


FIGURE 6.3.

An object model of a structural environment.

the 'P_ObjectType' class. Each sub-class of the 'P_ObjectType' is coupled with a sub-class of the 'DataObject' class (or one of its classified sub classes).

In the first model, the first added set is a set of architectural DOs classified as sub-classes of the 'DO_Element' class which is a classification of the 'DataObject' class. The DO_Element set comprises of design elements needed by the architectural domain including site and landscape elements. An architectural design element may include site, building, floor, and wall etc. An instance of a site element would hold information about a site such as location,

area, coordinates, orientation, topography and links to existing landscape objects. On the other hand, 'DO_Activity' class, which is a holder of any activity objects is intend to accommodate additional architectural EAs [El_Attar 97], same applies to the 'DO_Occupant' class, which is a holder of any occupant objects.

This classification is exemplary and can be collapsed or expanded to accommodate other related DOs. A DM may remove some of these DO classes during a session. However, any newly defined DO classes need to be introduced before any session (unless the implementation environment permits that).

The second set is a group of sub-classes of the 'P_ObjectType' class. It provides the associated object-type protocol classes which is a partial set of the necessary protocols needed by a DO to act as an OA. Each sub-class of 'P_ObjectType' is coupled with one of the classified sub-classes of the 'DataObject' class. Those are used after the activation of a DO. When a DO is activated to an OA an instance of the coupled sub-class is created and loaded into the OA (for details see Section Heading2).

Figure 6.3 provides a second example of domain specific model for structural engineering. As in, Figure 6.2 a set of structural 'DataObject' sub-classes and coupled 'P_ObjectType' sub-classes are added to the general model of Figure 6.1.

6.3 DO-Hierarchies

DO-hierarchies are used for task decompositions (see Sections 4.3.4 and 6.2.1). The DM may either:

1. Establish a new hierarchy of the DOs of the environment.
2. Use existing hierarchy which may either be provided by the environment or saved from a previous session. The DM may use an entire hierarchy or a subset of it. The DM may modify an existing hierarchy by adding DOs or changing the nature of relations between DOs.

Agents of the environment uses the current DO-hierarchies (as established by the DM) and may not modify or establish new hierarchies. Figure 6.4 shows an exemplary object model of an architectural hierarchy which may be provided in the environment. The model covers a variety of possible relations between

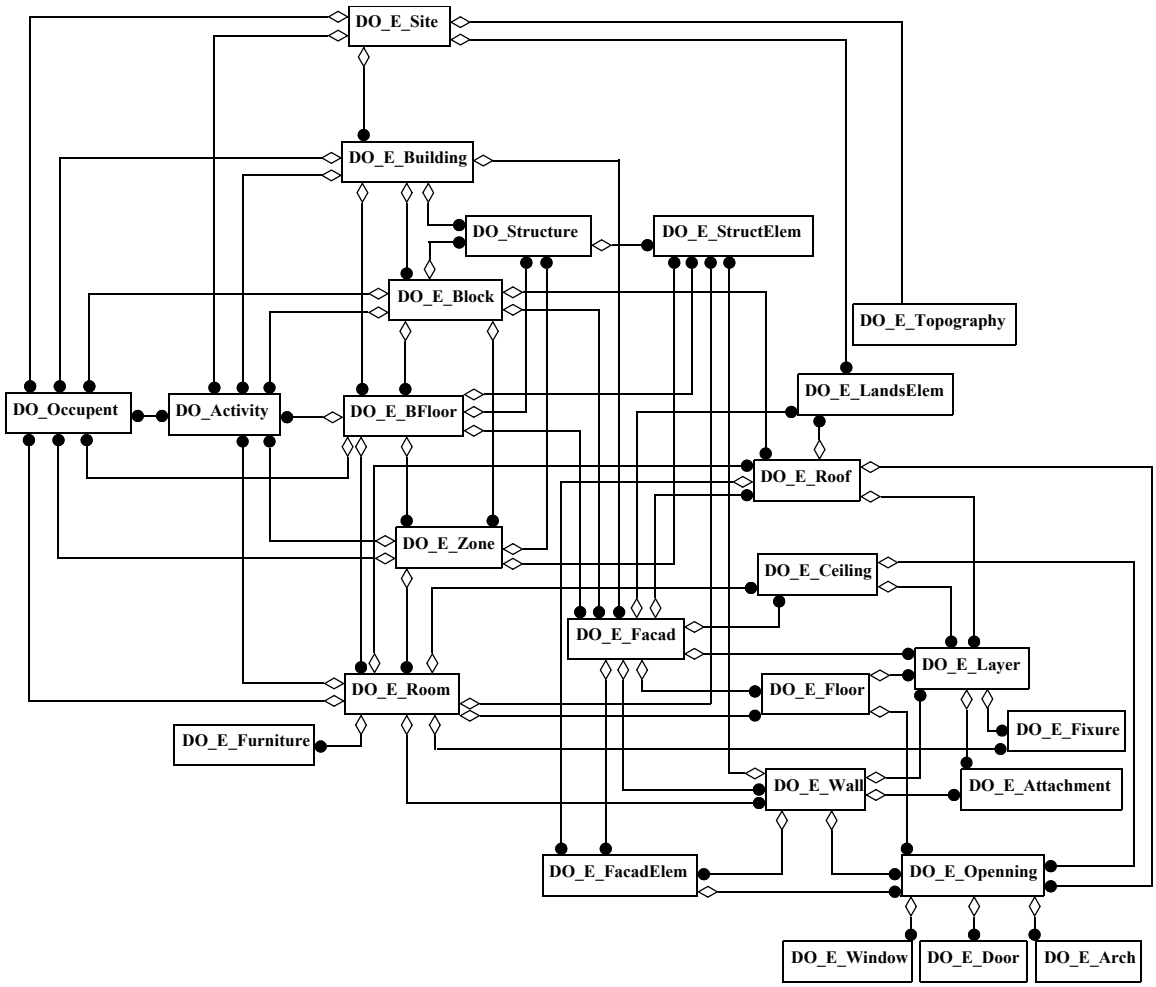


FIGURE 6.4.

An architectural object hierarchy.

architectural objects. A DM may use the entire model as the current DO-hierarchy or may compile a DO-hierarchy as a subset of this model. A DM should be able to graphically compile a subset of any DO-hierarchy. Certain relations and constraints may have to be maintained in any subset hierarchy, therefore, it may be required to develop a warning mechanism if such relations or constraints are violated.

The model also includes few non-design objects such as Occupant, Activity, Landscape, and Topography. The earlier two objects may be expanded to include hierarchies about the different occupants and different activities that may occur in an architectural space [El_Attar, 97]. Landscape and Topography objects may be expanded to include a wide range of natural site objects.

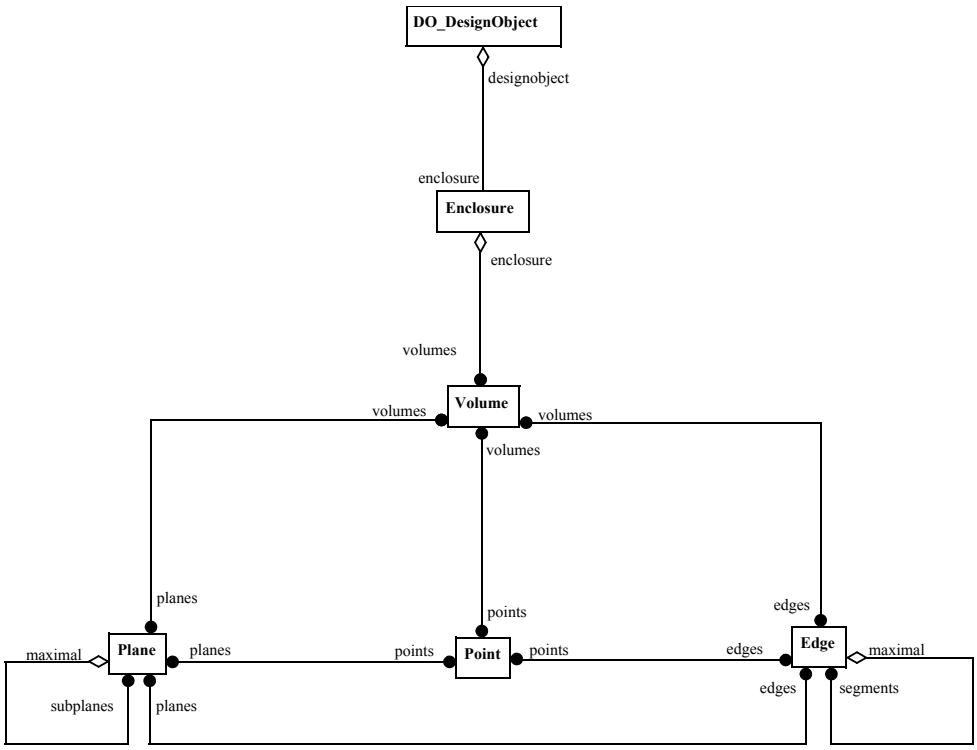


FIGURE 6.5.

Object model for geometrical representation.

Except Occupant and Activity objects, each object in Figure 6.4 typically has an enclosure with volume that has regular geometry. Figure 6.5 provides an object model for objects with enclosure. The model is intend to accommodate various geometrical configurations. Such hierarchy may operates under solid or two dimensional representations of objects. The existence of such hierarchy within the environment is essential and permits EAs with geometric interpretation capabilities to participate in a design session and produce vital information to the execution of tasks by other agents in the environment.

6.4 Implementation Design of the Activation Process

The following scenario is provided to illustrate how an OA is created according to the architectural object model provided in Figure 6.2.

- An agent (e.g., DM) requests the activation of a DO_E_Room to perform a daylighting evaluation task.

- The request is received by the 'A_Object' class which, in turn, creates a Room-OA instance (an A_O_Room according to the naming convention used in this model as described in Section 6.5.1).
- The A_O_Room requests a clone of the DO_E_Room (a copy of the exact DO_E_Room being activated).
- Provided the clone, the A_O_Room would load the interaction protocols related to its DO-type (makes an instance of the 'P_OT_Room' class).
- The DM assigns the task to the created A_O_Room, which, in turn, loads the task-type protocols (makes an instance of the 'P_TT_Evaluation' class). This enables the A_O_Room to proceed with the execution of the evaluation task.
- The A_O_Room also loads the domain specific daylighting protocols (makes an instance of the 'P_D_Daylighting' class) which provide the daylighting parameters needed by the A_O_Room for the decomposition, aggregation and sorting of the daylighting task (all are subclasses of the 'P_D_Daylighting' class).
- The A_O_Room is now ready to interact with the environment agents to complete the execution of the assigned task (for more details on the execution of a daylighting evaluation task see Chart 5 of Chapter 5).

This Scenario is generalized in the activation diagram provided in Figure 6.6. The implementation of the activation process starting from the activation request till the completion of the OA with all its necessary objects is shown in this figure. The steps are marked sequentially to illustrate the activation process in detail if implemented in an object-oriented development environment. Each gray rectangle indicates an object with its sequential lists that are necessary for the creation of an OA and the assignment of a task. Each arrow represent a specific event as explained below. These elaborated 28 steps are derived from the first eight steps of activation event-trace chart (Chart 1 of Chapter 5).

Detailed steps of the activation process shown in Figure 6.6:

1. activate: a DM (or any agent) send an activation message to a DO;
2. new OA: the DO instantiates an OA object;
3. register: the OA registers itself in the OAs list of the DO;
4. register: the DO registers itself in the DOs list of the OA;

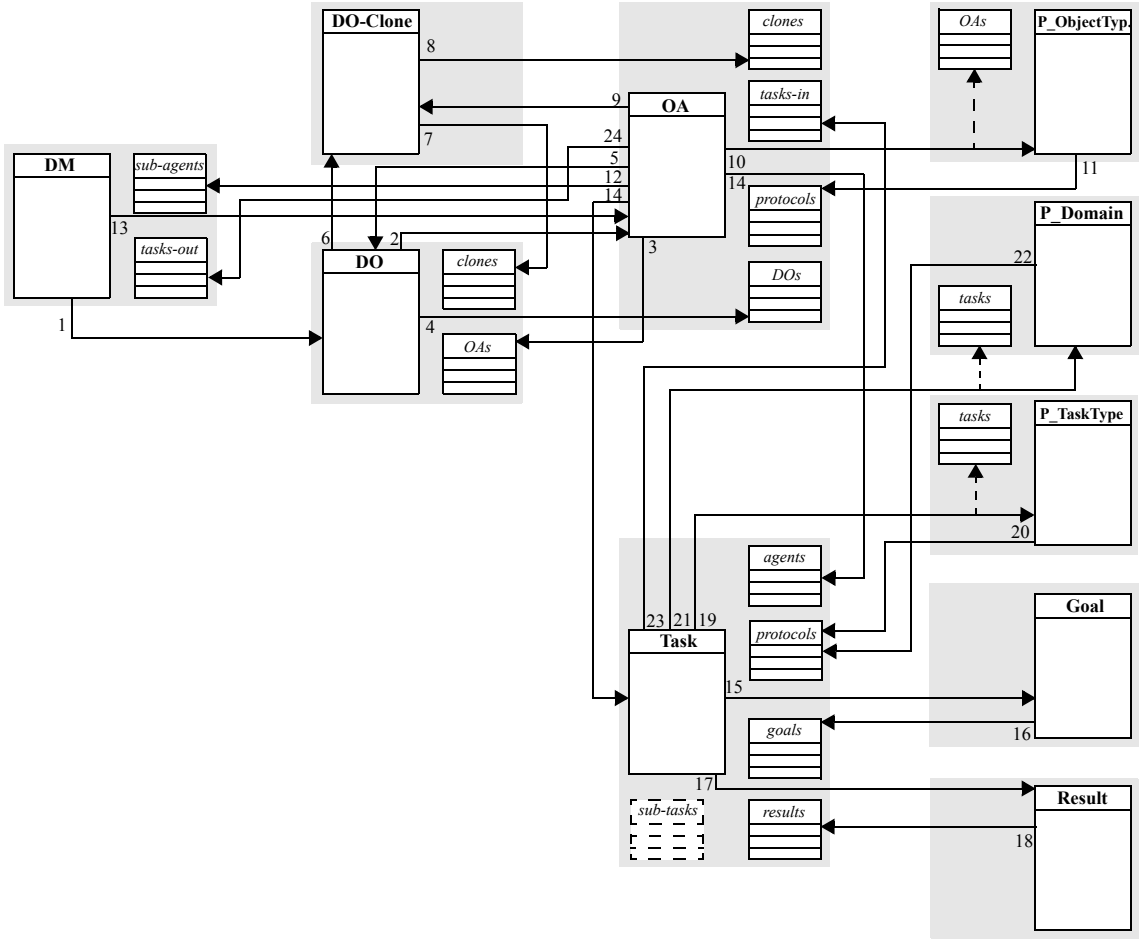


FIGURE 6.6.

The implementation design of the activation process.

5. clone: the OA requests a clone of the DO;
6. new clone: the DO duplicates itself. In addition, the DO registers itself with the DO-Clone);
7. register: clone registers itself in the clones list of the DO;
8. register: DO-Clone registers itself in the clones list of the OA;
9. register: OA registers itself with the DO-Clone;
10. new P_ObjectType: the OA instantiates a new object-type protocol object of the same DO-type (e.g., Wall-DO, Room-DO). In addition, the OA registers itself in the OAs list of the P_ObjectType;
11. register: the P_ObjectType registers itself in the Protocols list of the OA;

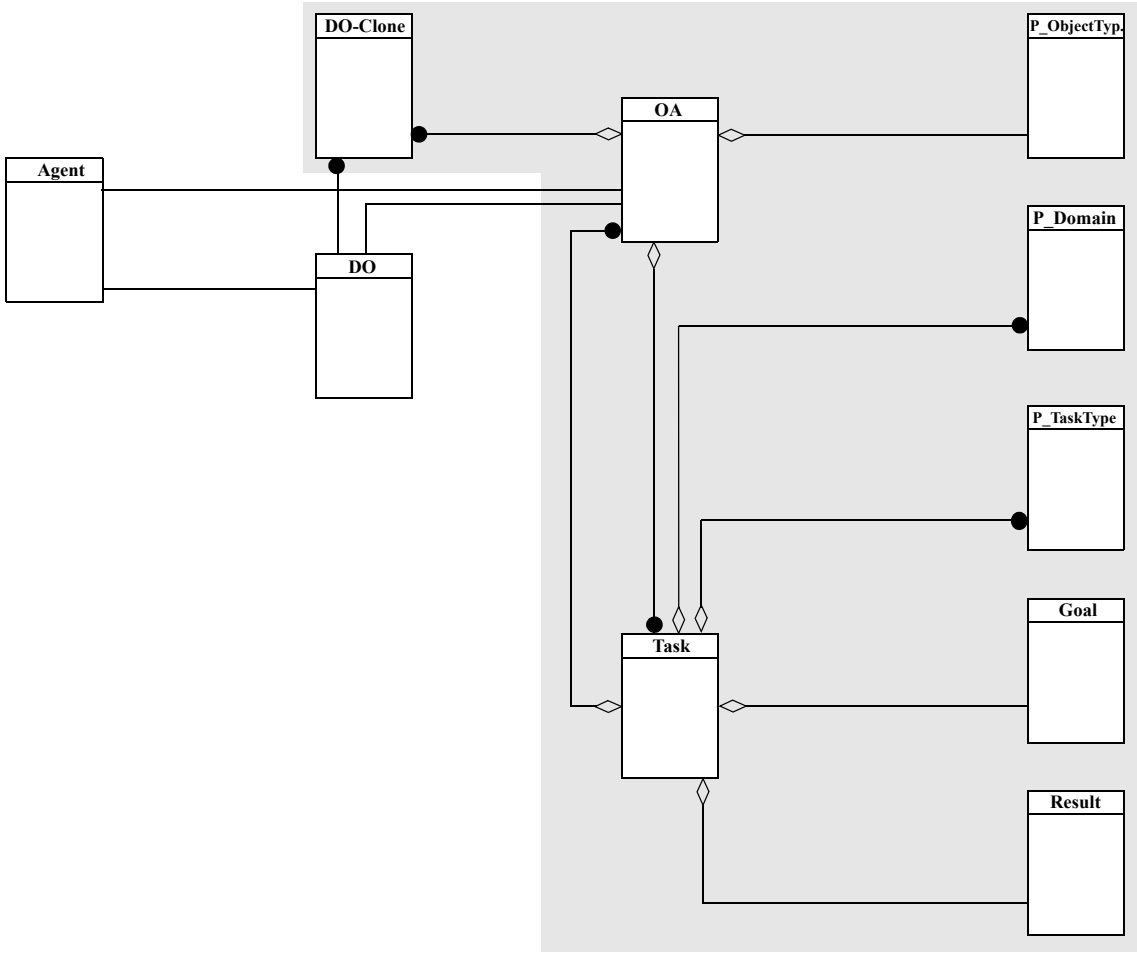


FIGURE 6.7.

Object model of an OA.

.

12. register: the OA registers itself in the sub-agents list of the DM;
13. assign: the DM assigns a task to the OA;
14. new task: the OA instantiates a new Task object. In addition, the OA registers itself in the OAs list of the Task;
15. new goal: the Task instantiates a new Goal object. In addition, the Task registers itself with the Goal;
16. register: the Goal registers itself in the goals list of the Task;
17. new result: the OA instantiates a new Result object. In addition, the Task registers itself with the Result;
18. register: the Result registers itself in the results list of the Task;

19. new P_TaskType: the Task instantiates a task-type protocol object of the same task-type (e.g., evaluation, conflict handling etc.). In addition, the Task registers itself in the tasks list of the P_TaskType;
20. register: the P_TaskType registers itself in the Protocols list of the Task;
21. new P_Domain: the Task instantiates a domain protocol object of the same task-domain and task-focus (e.g., cost, structure). In addition, the Task registers itself in the tasks list of the P_Domain;
22. register: the P_Domain registers itself in the Protocols list of the Task;
23. register: the Task registers itself in the tasks-in list of the OA;
24. register: the OA registers itself in the tasks-out list of the DM;
25. execute task: OA starts executing the assigned task (this step is not included in Figure 6.5).

Figure 6.7 shows an object model of an OA after it is instantiated, assigned a task and loaded the appropriate protocols. The objects in the shaded area belong to the OA. Objects outside of the shaded area have immediate relations to the OA but are not part of the OA (e.g., the super-agent which activated the DO and assigned the task to its OA).

6.5 The Objects Implementation Design

This section presents the design of objects of the model shown in Figure 6.1 to be developed in an OODE. The objects are defined earlier in Section 6.2.1 and are detailed in this section.

6.5.1 The object structure

As in any OODE, each object has three main components; a name, a set of attributes and a set of methods to execute operations that are mostly related to the attribute values of the object. Graphically, each object is represented by a box which is divided into three parts that corresponds with each of the three components. The upper part contains the object name; the middle part contains the object attributes; and the lower part contains the object methods (see Figures 6.8-6.12).

Naming convention

The object names follow a convention that is specific to this framework. Names of objects in the same hierarchy starts with the same letter. The sub-class name is appended to the initials of its super-classes.[7] The initials of a class is either one

or more letter, for instance the initials of the class named 'DataObject' are 'DO'. Therefore, any 'DataObject' subclass name starts with 'DO_'. All types of agents starts with the letter 'A_' since the super-class is named 'Agent'. 'A_System' is an SA and it is a sub-class of the 'Agent' class. A_S_Expert is a domain EA which is a sub-class of the 'A_System' class.

Attributes Object attributes are holders to the necessary information of the object. Each attribute is declared with its type whether it is factual or relational data. Data types such as; 'bool' a boolean value, 'char *' a pointer to an array of characters, 'string' a string, 'int' an integer, 'float' a float; 'SeqCollection *' a pointer to a sequential collection (a list) or 'OrdCollection *' a pointer to an ordered collection (an ordered list), '<object-name> *' a pointer to an object, '<_object-name_>' a specific object type, or exceptional types such as 'void' which may act as wild card for any type provided within the implementation environment.

The attributes are also relation holders. For instance, an attribute of the type '<object-name> *' constitutes a relation to a single object, while an attribute of the type 'SeqCollection *' constitutes a set of relations to a set of objects. In the object models created using OMTool, such as the model shown in Figure 6.1, relations between objects are identified through roles (for more details about roles see Section 6.2.1). Attribute values and relations may also represent constraints. For instance, an attribute may hold a lower and upper values, or may be broken into two attributes each of which holds a value limit. However, the issue of constraint representation, propagation and management for the OAs require further work that is beyond the scope of this thesis.

The objects illustrated on this chapter are produced in OMTool and, therefore, they follow the OMTool symbology. When an attribute name is preceded with a '+' symbol this indicates a **public** attribute, any object can access such information. A '#' symbol indicates a **protected** attribute, only the sub-classes can access such information. A '-' symbol indicates a **private** attribute, no other object can access such information.[8]

In this implementation design most of the attributes are protected. Therefore, accessing the information stored in any object attribute is a function of the object methods associated with such attribute. For example, in Figure 6.10, the 'Agent'

7. In this model no class is a sub-class of two super-classes, there can be only one hierarchy of super-classes for each sub-class.

class has a ‘sub-agent’ attribute which is a list of all current agents that are currently assigned tasks by the agent. The methods associated with the ‘sub-agent’ attribute are the typical add, get and remove; ‘addSubAgent’ to add a new agent on the sub-agent list, ‘removeSubAgent’ to remove an agent from the list, ‘getSubAgents’ to obtain the list.

An attribute may be set to a specified default value upon creation of the object instance. For example, in Figure 6.8, the ‘Data_Object’ class has an ‘activationstatus’ attribute that is set to ‘FALSE’ upon creation of an instance, and the ‘numofclones’ attribute is set to ‘0’. Other attributes are set upon creation or later using its associated ‘set’ method.

Methods As in any OODE, the object methods serves primarily as the interface to its own attributes. External objects may use the object methods to access or manipulate (if permitted) the information stored in the object attributes. Internally, an object may use its methods to manipulate its own attributes, such as setting values or adding external object to its lists when necessary.

Typically, common attributes of the types ‘char *’, ‘bool’, ‘int’, ‘float’ and ‘<object-name> *’are associated with ‘set’ and ‘get’ methods, while attributes of the type ‘SeqCollection’ are associated with ‘add’, ‘remove’ (or ‘delete’ in some cases), and ‘get’ methods. Each class in this model has a ‘name’ attribute of the type ‘char *’, ‘ID’ attribute of the type ‘int’ and in many cases ‘type’ attribute of the type ‘char *’. Both ‘ID’ and ‘name’ attribute has its related ‘get’ and ‘set’ methods. A sub-class of any generalization inherit such attributes and methods from its super-class, therefore, such attributes and methods are not repeated in any sub-class. For any concrete sub-class the value of such attributes (such as ID) are assigned upon creation.

In each object there is a ‘Constructor’ method and a ‘Destructors’ method. A constructor method uses the same object name, and destructor method uses the same object name preceded by a ‘~’. A constructor method main purpose is to create an instance of such class. The initial attribute settings are performed

8. Some of the attributes in the object figures are shown for clarity of object design though they are not needed during the generation of the C++ code by OMTool. Such attributes are necessary and each of them are represented on the object models of Figure 6.1 as a ‘role’. During the generation of the C++ code OMTool translate each role into an attribute automatically, therefore, they should not be represented explicitly in the object attributes during implementation to avoid redundancy.

during the execution of the constructor method. A destructor method is used to eliminate an instance with all its pointers.[9]

Other methods are designed to achieve environment specific functions such as the 'activate' and 'deactivate' methods of the 'DataObject' class, or for manipulating external objects such as the 'assignTask' method of the 'Agent' class. Certainly all methods of agency such as task, goal, result or any protocol objects are environment specific.

6.5.2 Characterized attributes of objects in the OA model

The following section describes few characterized attributes of selected objects from those shown in Figures 7.8-7.12. Each object is represented by a figure which shows the attributes and methods that are necessary to the functionality of that object within the OA model. Other attributes and methods that are not directly related to the functionality of the OA model are not listed neither in the figures nor on the attribute descriptions.

Attributes of the 'Session' class (of Figure 6.8):

- environments: a list of all saved environments

Attributes of the 'Environment' class (of Figure 6.8):

- session: a link to the Session where this Environment is created
- scenarios: a list of all saved Scenarios of this Environment
- agents: a list of Agents currently active in this Environment
- DOs: a list of DOs currently instantiated in this Environment

Attributes of the 'Scenario' class (of Figure 6.8):

- environment: a link to the Environment where this Scenario is recorded

9. In some cases such a destructor method deals with the lower level necessities of object elimination such as memory management. This depends on the OODE. In most C++ environments such allocation and freeing of memory spaces is required, while in JAVA and Eifel [Meyer, 88] this is unnecessary. ET++ provides means to reduce the need for memory management and pointer deletion upon elimination of an object.

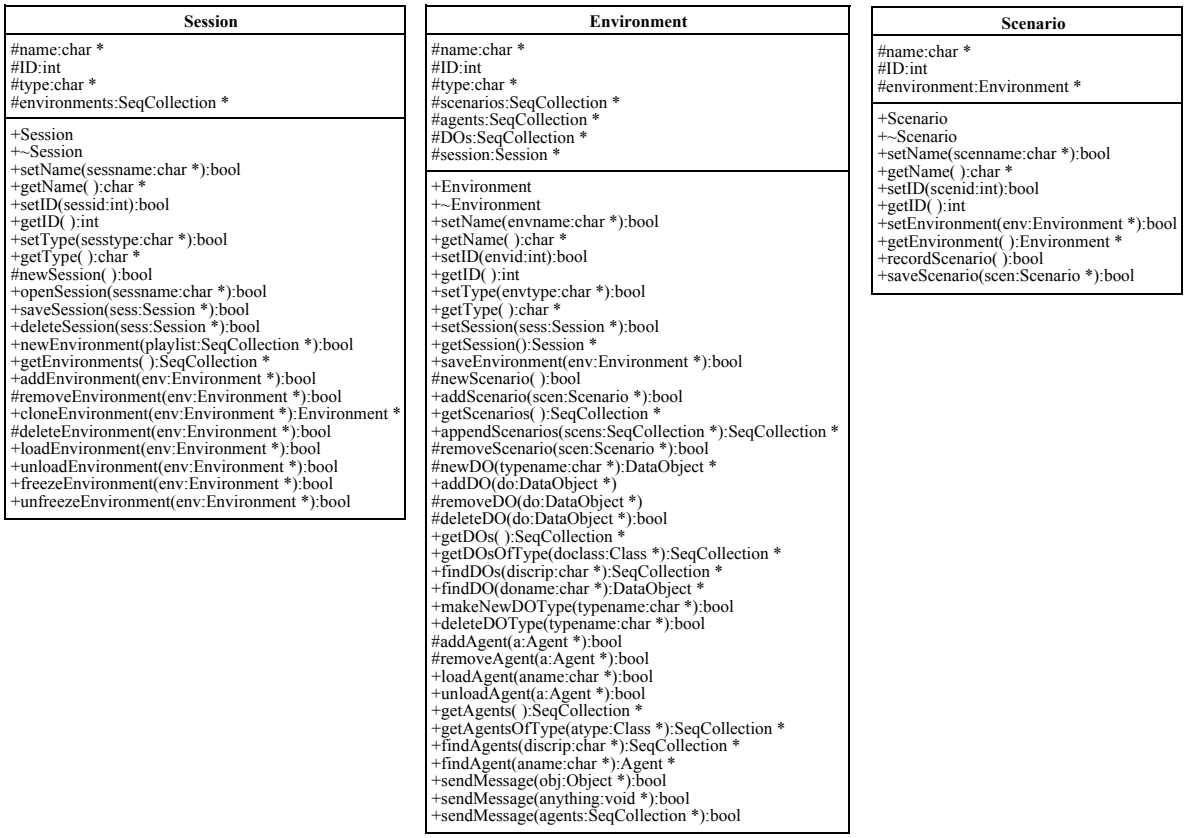


FIGURE 6.8.

Session, Environment and Scenario Objects.

Attributes of the 'DataObject' class (DO) (of Figure 6.9):

- activationstatus: a boolean to indicate whether an OA for this DO is currently active
- clones: a list of all clones of this DO currently used by OAs
- numofclones: number of clones on the previous list
- interestlist: a list of interested DOs and EAs for each attribute of the DO
- shared: a boolean to indicate whether the DO is a shared one
- OAs: a list of OAs that are currently representing the DO
- clonesof: the DO of which this clone is a duplicate of (only when the DO is a cloned instance)
- master: the first OA that is currently representing the DO
- environment: the current environment where the DO exists
- constraints: a list of all constraints on the DO attributes

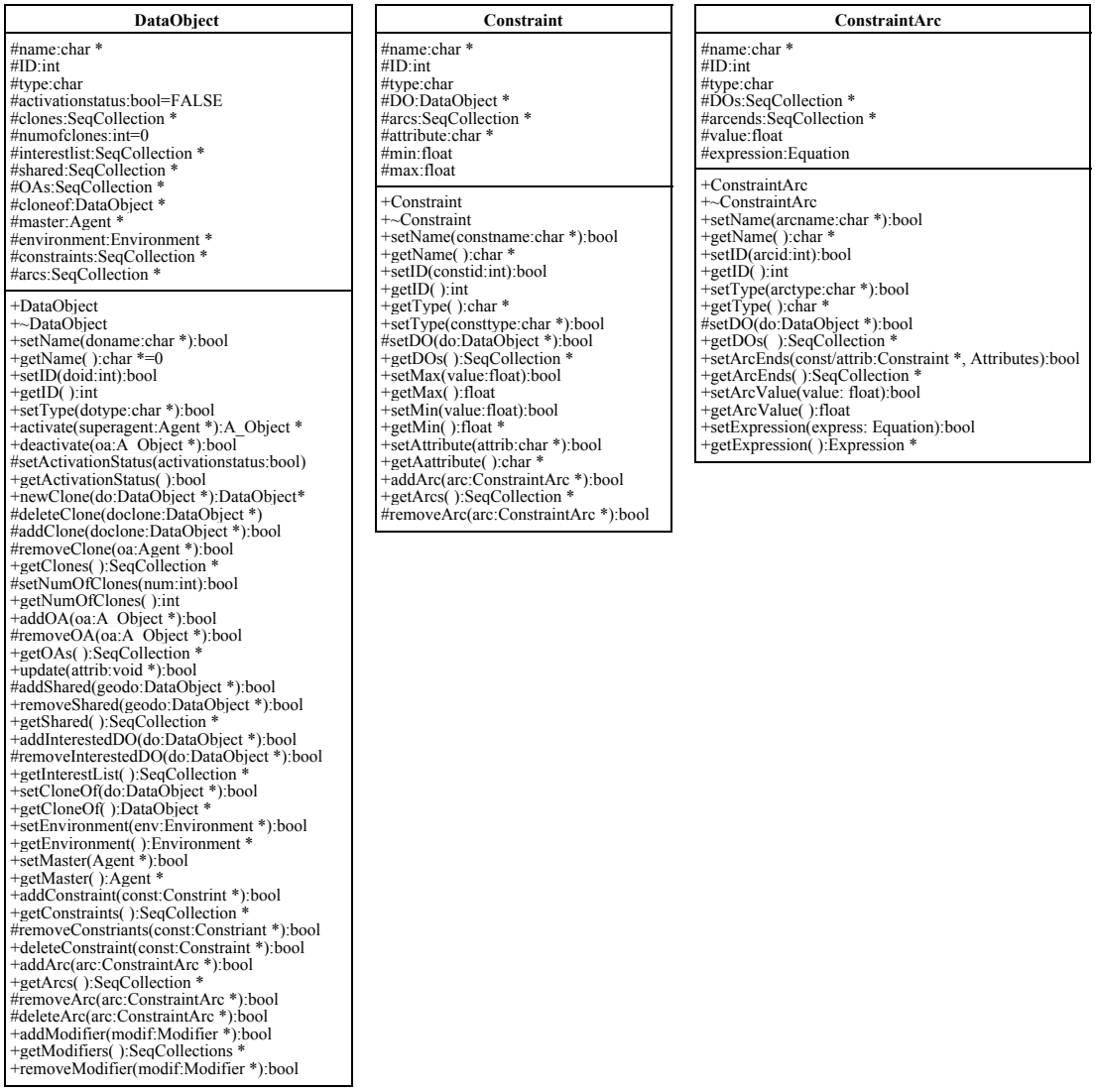


FIGURE 6.9.

DataObject (DO), Constraint and ConstraintArc objects.

- arcs: a list of all constraint arcs linking the DO attributes with other attributes, constraints or constraint arcs.

Attributes of the ‘Constraint’ class (of Figure 6.9):

- DO: the DO of the constrained attribute
- arcs: a list of all constraint arcs linking this constraint with other attributes, constraints or constraint arcs

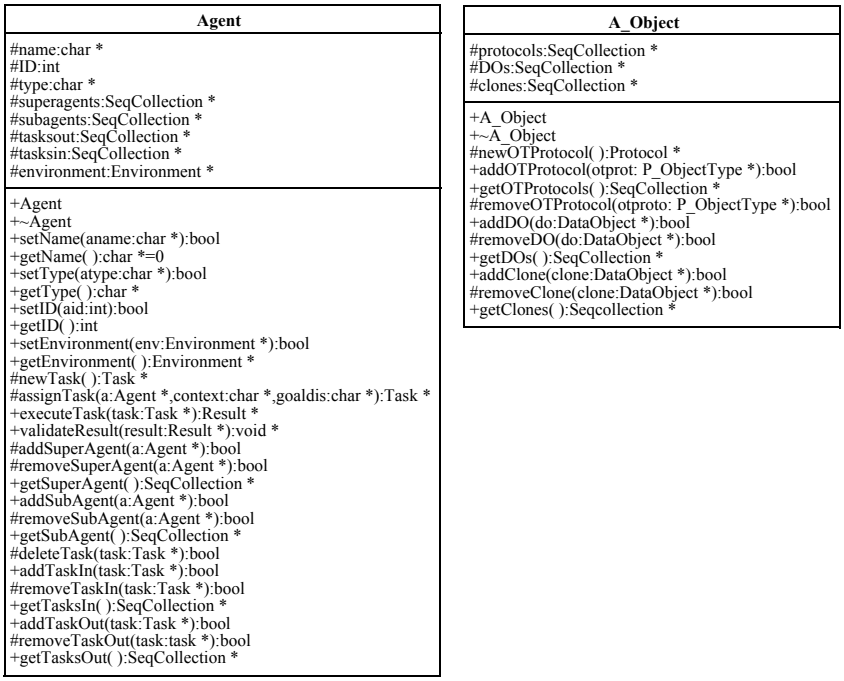


FIGURE 6.10.

Agent and A_Object (OA) Objects.

- attribute: the attribute name
- min: the lower bond of the constraint acceptable range
- max: the upper bond of the constraint acceptable range

Attributes of the 'ConstraintArc' class (of Figure 6.9):

- DOs: a list of DO using this constraint arc
- arcends: a list of attributes, constraints, and constraint arcs linked by this constraint arc
- expression: an expression that represents the link between the arc-ends.
- value: the current evaluation of the constraint arc expression (e.g., mathematical equation)

Attributes of the 'Agent' class (of Figure 6.10):

- superagents: a list of all agents that are currently assigning a task to this agent
- subagents: a list of all agents that are currently assigned tasks by this agent
- tasksout: a list of all assigned tasks by this agent

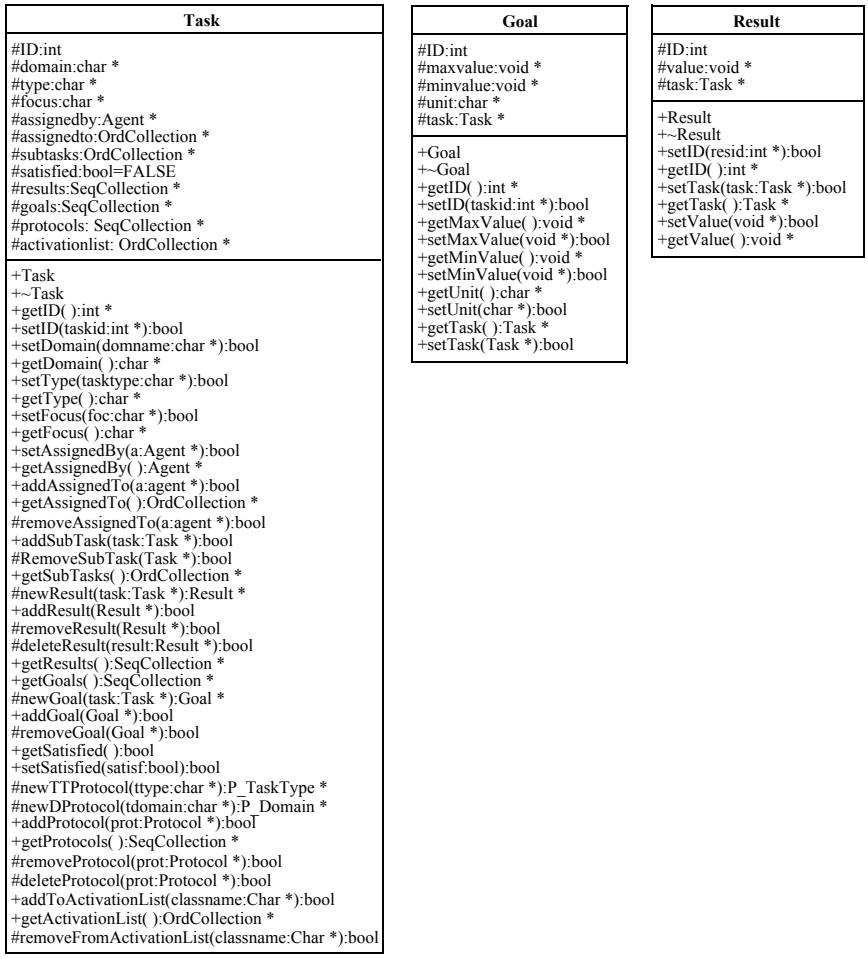


FIGURE 6.11.

Task, Goal, and Result Objects.

- taskin: a list of all tasks assigned to this agent
- environment: the environment where this agent currently exists

Attributes of the 'A_Object' class (of Figure 6.10):

- DOs: a list of all DOs that are currently represented by this OA
- clones: a list of all clones that are currently used by this OA

Attributes of the 'Task' class (of Figure 6.11):

- assignedby: the agent that assigned this task
- focus: the main attribute to be modified or the alternative attribute value to be examined

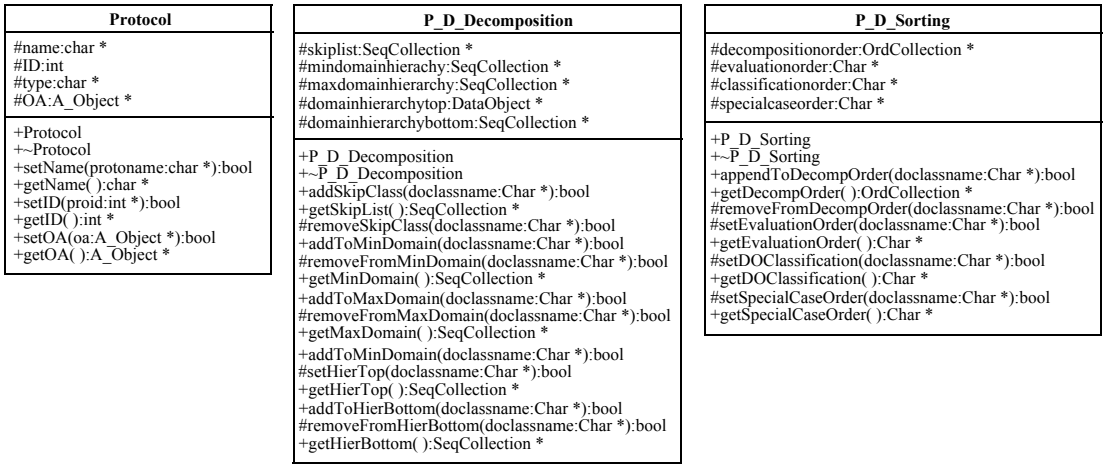


FIGURE 6.12.

Protocol, P_D_Decomposition and P_D_Sorting objects.

- subtasks: a list of all sub-tasks assigned to other agents as a results of executing this task (e.g., decomposition of this task)
- assignedto: a list of all sub-agents that are currently assigned sub-tasks by this task
- satisfied: a boolean to indicate whether this task is executed successfully
- results: a list of all results generated as a result of executing this task
- goals: a list of all goals to be accomplished by the execution of this task
- protocols: a list of all protocols that are currently loaded for the execution of this task

Attributes of the 'Goal' class (of Figure 6.11):

- maxvalue: upper bond of the acceptable value range
- minvalue: lower bond for the acceptable value range
- unit: units of measurement (of the values)
- task: the task of which this goal is related

Attributes of the 'Result' class (of Figure 6.11):

- value: the current result value
- task: the task of which this result is related

Attributes of the 'Protocol' class (of Figure 6.12):

- OA: a list of all OAs that are currently using this protocol

The following two objects are exemplary damian protocols. These are not included in the object model of Figure 6.1:

Attributes of the 'P_D_Decomposition' class (of Figure 6.12):

- skiplist: a list of all DOs to be skipped during the decomposition of a task
- mindomainhierarchy: a set of the minimum DO classes needed for the execution of tasks of this domain
- maxdomainhierarchy: a set of the maximum DO classes that can be included for the execution of tasks of this domain
- domainhierarchytop: a DO class that marks the upper bond of the domain-hierarchy
- domainhierarchybottom: a set of DO classes that mark the lower bond of the domain-hierarchy

Attributes of the 'P_D_Sorting' class (of Figure 6.12):

- evaluationorder: the general orientation of task decomposition
- decompositionorder: an ordered list of DO classes used as a specific guide for task decomposition, activation, and hence, task execution.
- doclassification: a DO class where the task result is classified about (e.g., cost of a BFloor-DO per Room_DO, the Room-DO is a doclassifiction in this case)
- specialcaseorder: an additional variable to accommodate special cases of task decomposition.

7 Conclusions

7.1 Contributions

During the past two decades, several design support tools have been developed for both research and commercial purposes. Most are stand-alone tools, few are comprehensive or collaborative design environments. Such tools encompass a wide range of design activities from simulation and evaluation to generation and recommendation to production and documentation. Most are intended for the early stages of design and for rapid prototyping, few for the later stages of design and for detailed modeling of the artifact being designed. Some tools have adopted the notion of computational agency. In such cases, the domain applications encapsulate the domain expertise and act as expert agents with a degree of autonomy. Nevertheless, none of the tools – that I have surveyed during the course of developing this thesis – employ any kind of representation where agency behavior can be considered as a property of design objects as well. This dissertation, thoroughly investigates the notion of design objects endowed with agency behavior. The engineering of an object-agent based framework for computational design environments, as a decision making environment, is the main contribution of this work.

7.1.1 Specific contributions

This thesis is structured around the development of a framework for an a object-agent-based decision making environment. During the course of developing this framework (Chapters 3-6) the following results were accomplished:

- A general architecture of an object-agent-based environment as a demonstration of how a design tool or a comprehensive design environment can be conceptually structured around such notions was developed (Chapter 3).
- A computational framework for task[1] execution, decomposition, delegation, and management for global and local decision making nodes was developed (Chapter 3, 4, and 5).
- A set of general and domain specific reusable patterns of interaction needed to allow a designer to orchestrate and finally benefit from such a fine grain multi-agent decision making environment was developed (Chapter 4). The developed patterns focused on evaluation tasks and on conflict handling. These patterns can be reused for generation, recommendation and implementation tasks. Each pattern of interaction is graphically represented by an enhanced event-trace chart. Each step in a pattern in the event-trace chart maps to an object method in an object oriented implementation of an OA-based environment (Chapter 4, 5, and 6).
- A set of interaction algorithms (mainly for activation, decomposition and conflict handling) to be used by the object-agents during the course of handling tasks was developed. With minimal modifications this set of algorithms can also be adopted for non-design decision making environments (Chapter 5).
- A mechanism for compiling an activation list to contain the design objects which can (or must) participate in the execution of a task was developed. Such activation lists reflect the valid task decomposition of a design object in a defined hierarchy (Chapter 5).
- A general object oriented implementation design for an object-agent-based environment was engineered (Chapter 6).

7.2 Research Topics and Agenda for Future Work

This dissertation addresses a number of fundamental issues around the notion of agency in design. The main outstanding task is a computer implementation of an object-agent-based design environment built upon the framework presented in this thesis. This effort requires the participation of distinct groups of researchers,

1. This is based on the notion of agency tied primarily to the ability to execute tasks.

and substantial funding. However, the issues listed below are open research topics that can be tackled by individual researchers. Advancements in any one of these issues can contribute to the approach advocated in this dissertation.

7.2.1 Object-agents knowledge

Access to and interpretation of external knowledge beyond the immediate coordination knowledge of an agent affects their role in the environment. This can be enhanced through:

1. A global communication mechanism; where agents can participate in communication systems where they are able to continually monitor and interpret messages of interest which may not be communicated directly to them (see Section 5.3.1).
2. Planning long term activities; where agents can dynamically plan activities considering other agents plans and capabilities (see Section 3.4.2).

7.2.2 Conflict handling mechanism

In conflict handling situations, enabling agents (in general) to conduct direct negotiations with other agents to resolve design conflicts is an area of interest to enhance the abilities of OAs (See sections 3.1 and 5.2.2).

Another area of work that is specific to the OA, is the enhancement of the conflict detection mechanism through the use of the $interest_{lists}$:

1. Sorting the interest list; establishing a weight mechanism to enable object-agents to sort any of their attribute $interest_{list}$ in respect to the degree of relevance to the task in hand (see Section 5.2.2).
2. Controlling conflict dependencies; establishing a mechanism to control the number of conflict handling sessions triggered by the task in hand and to eliminate possible cyclic dependencies among members and non-members of an attribute interest list (see conflict handling in Section 3.1).

7.2.3 Object-agent autonomy in design

Agent autonomy and the ability to self-initiate tasks, plan activities, handle expanded goals can be enhanced through:

1. Self-initiated tasks; in addition to executing assigned tasks object-agents should have the ability to initiate a task when it sees fit (see Section 5.1).

2. Expanding the notion of goals for object-agents; an object-agent executing a task may then encapsulate an entire structure which represents design requirements, functional specification of that task. To consider a goal as accomplished, all requirements of such structure (which are sub-components of the goal) need to be satisfied (see Section 6.2.1).

7.2.4 Interface of an object-agent-based environment

An object-agent-based environment is a highly interactive system. Therefore, interface design plays a major role in its success. In fact, in such environments, it is difficult to draw the line between the interface and the main system functionality. One may consider some of the interface issues listed here out of context. It is my judgement that these issues are more related to designer/agents interactions and accordingly related to interface design. I list four main aspects of interfaces that require further research; building, providing and manipulating task dependent hierarchies, controlling the flow of task executions, managing conflict handling sessions, and facilitating communications among agents.

1. Providing and manipulating task dependent hierarchies; an interface should provide functionality related to whole or sub-parts of hierarchies such as create, save, import, freeze or related to relations among objects such as new, duplicate, add, remove (see Section 3.3.3). Such functionality should be provided to the designer in various forms especially graphical. Established hierarchies and relations should be validated and should generate errors, warnings and notes as necessary. In addition, the ability to reuse and modify hierarchies should be supported. The interface should provide libraries of previously established or typically used domain hierarchies.
2. Controlling the flow of task executions; through the interface the designer should be aware (upon request) of each task being executed and who is executing such task. The interface should provide textual and graphical representation of the tasks execution and delegation among the agents at any point in time. The designer should be able to interact by assigning, eliminating, replicating, freezing tasks, providing alternative values for re-evaluation, validating and implementing task execution results through any provided form of interface representation.

3. Managing conflict handling sessions; the interface should provide the designer with the means to access and re-sort $interest_{lists}$ of any design object (see Section 5.2.2). Any cross dependencies among $interest_{list}$ members should be graphically displayed as a warning mechanism. The interface should provide the designer with a mechanism to control the number of participants in a conflict handling session and the depth conflict checking to be generated from a task.
4. Facilitating communications among agents; to control the interactions among agents of the environment the interface should provide the designer with the ability to suspend and establish communication among single and groups of agents. A message classification mechanism provided by the interface (or agents of the interface) may also help agents of the environment to participate more efficiently in a global messaging system.

Bibliography

[Aarhus 95] Denmark Aarhus (1995). "Object and Agents Love at first Sight or Shotgun Wedding?". ECOOP 95, Workshops, Copenhagen, Denmark.

[Abiteboul 91] Abiteboul, Serge and Bonner, Anthony (1991). "Objects and Views". In Proc. of the ACM SIGMOD International Conference on Management of Data. Denver, Colorado, May, 1991.

[Acharya 92] Acharya, A. Tambe, M. and Gupta, A. (1992). "Implementation of Production Systems on Message-Passing Computers". In IEEE Transactions on Parallel and Distributed Systems, 3(4), July, pp. 477-488.

[Agha 86] Agha, G. A. (1986). "Actors: A Model of Concurrent Computation in Distributed Systems", The MIT Press, Cambridge, Massachusetts.

[Agre 95] Philip E. Agre (1995). "Computational Research on Interacbion and Agency", Artificial Intelligence 72(1-2), 1995, Pages 1-52.

[Ahmed 91] Ahmed, S., Wong, A., Sriram, D. and Logcher R. A (1991). "Comparison of Object-Oriented Database Management Systems for Engineering Applications", Intelligent Engineering Systems Lab, Research Report R91-12, Department of Civil Engineering, M.I.T., May 1991.

[Akin 89] Akin, Ö., Dave, B. and Pithavadian, S. (1989). "A Paradigm for Problem Restructuring in Design". In Proc. of the NSF Engineering Design Research Conf., University of Massachusetts, Amherst, MA.

[Akin 88a] Akin, Ö. (1988). "Architectural Design as Task for Complex Problem Solving Systems", Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.

[Akin 88b] Akin, Ö. (1988). "Expertise of the Architect", Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.

[Alberts 92] Alberts, L., Mars, N. and Wognwm, P. (1992). "Structuring Design Knowledge on the Basis of Generic Components". In Artificial Intelligence in Design '92, Gero, J. S. (Ed.), Kluwer Academic Publishers, Dordrecht, The Netherlands.

[Assal 94] Assal, H. and Eastman, C. (1994). "An Object-based Information Model for Design Supporting partial Integrity", University of California at Los Angeles.

[Augenbroe 92] Augenbroe, G. (1992). "Integrated building performance evaluation in the early design stages". In Building and Environment 27 (2), pp. 149 - 161.

[Aygen 99] Z. Aygen (1999). "A Hybrid Model for Case Indexing and Retrieval in Building Design", School of Architecture, Canegie Mellon University, Pittsburgh, PA.

[Bahler 92] Bahler, D. and Bowen, J. (1992). "Supporting Multiple Perspectives: A Constraint-Based Approach to Concurrent Engineering". In Artificial Intelligence in Design '92, Gero, J. S. (Ed.), Kluwer Academic Publishers, Dordrecht, The Netherlands.

[Banerjee 87] Banerjee, Jay; Kim, Won; Kim, H.J.; and Korth, Henry F. (1987). "Semantic and Implementation of Schema Evolution in Object-Oriented Databases". In Proc., ACM SIGMOD Annual Conference, 16:3, pp. 311-322.

[Basye 95] Basye, K., Dean, T. and L. P. Kaelbling (1995). "Learning Dynamics: System Identification for Perceptually Challenged Agents", Artificial Intelligence 72(1-2), Pages 139-171.

[Bates 92] Bates, J., Loyall, A. B. and Reilly, W. S. (1992). "An Architecture for Action, Emotion and Social Behaviour". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems. Castelfranchi, C. and Werner, E. (Eds.), Selected Papers of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 1992, Springer-Verlag, Berlin, Germany.

[Batory 85] Batory, D.S. and Kim, Won. (1985). "Modeling Concepts for VLSI CAD Objects", ACM Transactions on Database Systems, Vol. 10, No. 3, pp.322-346, September 1985.

[Batory 84] Batory, D.S. and Buchmann, Alejandro P. Molecular (1984). "Objects, Abstract Data Types and Data Models: A Framework". In Proc. of the 10th International Conference on Very Large Data Bases. Singapore, August, 1984.

[Baykan 92] Baykan C. and Flemming U. (1992). "Constraint Generation vs. Generate and Test", EDRC Tech Report 92, Carnegie Mellon University, Pittsburgh, PA.

[Barsalou 91] Barsalou, Thierry, Keller, Arthur M., Siambela, Niki and Wiederhold Gio. (1991). "Updating Relational Databases Through Object-Based Views". In Proc. of the ACM SIGMOD International Conference on Management of Data. Denver, Colorado, May, 1991.

[Beer 95] Randall D. Beer (1995). "A Dynamical Systems Perspective on Agent-Enviroment Interaction", Artificial Intelligence 72(1-2), Pages 173-215.

[Berker 95] Berker, I., and D.C.Brown (1995). "Conflicts and Negotiations in Single Function Agent Based Design Systems", Submitted to Concurrent Engineering: Research and Applications.

[Bertino 91] Bertino, Elisa and Martino, Lorenzo. (1991). "Object-Oriented Database Management Systems: Concepts and Issues", IEEE Journal of Computer, April 1991.

[Biliris 89a] Biliris, Alexandros (1989) . “A Data Model for Engineering Design Objects", IEEE, August, 1989.

[Biliris 89b] Biliris, Alexandros (1989) . "Management of Objects in Engineering Design Applications", Technical Report, BU-CS TR 89-005. Computer Science Department, Boston University.

[Birmingham 95] Birmingham, W. P., E. H. Durfee, T. Mullen, and M.P. Wellman (1995). "The Distributed Agent Architecture of the University of Michigan Digital Library" (extended abstract), AAAI Spring Symposium on Information Gathering in Distributed, Heterogeneous Environments.

[Boden 96] Boden, Margaret A. (Ed.) 1996. "Artificial Intelligence: Handbook of Perception and Cognition", 2nd Edition, Academic Press, INC.

[Bond 89] Bond, A. H. (1989). "The Cooperation of Experts in Engineering Design". In Distributed Artificial Intelligence, Gasser, L. and Huhns, M. (Eds.), Vol. II, pp. 463-484, Pitman Publishing, Los Altos, CA.

[Booch 94] Grady Booch (1994). "Object-Oriented Analysis and Design with Applications", Booch, G., Jacobson, I.and Rumbaugh, J., (Eds.) Addison-Wesley, Reading, MA.

[Brafman 96] Brafman, R. I., Tennenholtz, M. (1996). "On Partially Controlled Multi-Agent Systems". In Journal of Artificial Intelligence Research 4, AI Access Foundation and Morgan Kaufmann Publishers.

[Brooks 86] Brooks, R. A. (1986). "Asynchronous Distributed Control System for a Mobile Robot". In Proc. SPIE's Cambridge, MA, Oct 86.

[Brown 89] Brown, D. and Chandrasekaran, B. (1989). "Design Problem Solving: Knowledge Structures and Control Strategies", Pitman Publishing/Morgan Kaufmann Publishers, Inc., San Mateo, CA.

[Buchmann 91] Buchmann, A.P., R.S. Carrera, M.A. Vasquez-Galindo (1991). "Handling constraints and their exceptions: an attached constraint handler for object-oriented CAD databases". In On Object-Oriented Database Systems, K. Dittrich, U. Dayal, and A. Buchmann (Eds.), Springer-Verlag, New York, pp. 65-83.

[Burkhard 95] Burkhard, H. (1995). "How to Define Agent Properties - Or: What is a Fair Agent?". In Lecture Notes in Artificial Intelligence 957: From Reaction to Cognition. Castelfranchi, C. and Muller, J. (Eds.), Selected Papers of 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW ’93, Neuchatel, Switzerland, August 1993, Springer-Verlag, Berlin, Germany.

[Burmeister 95] Burmeister, B., Haddadi, A. and Sundermeyer, K. (1995). "Generic, Configurable, Cooperation Protocols for Multi-Agent Systems". In Lecture Notes in Artificial Intelligence 957: From Reaction to Cognition. Castelfranchi, C. and Muller, J. (Eds.), Selected Papers of 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '93, Neuchatel, Switzerland, August 1993, Springer-Verlag, Berlin, Germany.

[Cardelli 85] Cardelli, Luca and Wegner, Peter. On Understanding (1985). "Types, Data Abstraction, and Polymorphism", Computing Surveys, Vol. 17, No. 4, December.

[Castelfranchi 97] Castelfranchi, C. and Falcone, R. (1997). "Delegation Conflicts". In Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Boman, M. and Van de Velde, W.(Eds.), Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer, NY.

[Cavedon 97] Cavedon, Lawrence, Rao, Anand & Wobcke, Wayne (EDS) (1997). "Intelligent Agent Systems: Theoretical and Practical Issues". In Lecture Notes in Artificial Intelligence 1209, Based on workshop held at PRICAI '96, Cairns, Australia, August 1996, Springer

[Chaib-Draa 96] Chaib-draa, B. (1996). "Interaction Between Agents in Routine, Familiar and Unfamiliar Situations", International Journal of Intelligent & Cooperative Information Systems (issue unkown).

[Chaib-Draa 94] Chaib-draa, B. and Levesque, P. (1994). "Hierarchical Model and Communication by Signs, Signals and Symbols in Multi-Agent Enviroments". In Lecture Notes in Artificial Intelligence 1069: Distributed Software Agents and Applications, Perram, J. W. and Muller, J. P. (Eds.), Proc. of 6th European Workshop on Modeling Autonomous Agents in a Multi- Agent World, MAAMAW '94, Odense, Denmark, August 1994, Springer, NY.

[Chaib-Draa 92] Chaib-draa, B., Mandiau, R. and Millot, P. (1992). "Distributed Artificial Intelligence: An Annotated Bibliography", Sigart Bulletin, ACM Press, Vol. 3, 3, August.

[Christen 95] Christen Krogh (1995). "The Rights of Agents" [IJCAI'95 Workshop on Agent Theories, Architectures, and Languages, Montreal - Quebec, August 19-20 1995].

[Cohen 94] Cohen, P., M. Wang, and SC Baeg (1994). "OAA: An Open Agent Architecture" [AAAI Spring Symposium, 1994].

[Conrey 88] Conrey, S., Meyer, R and Lesser (1988). "Multistage Negotiation in Distributed Planning". In Readings in Distributed Artificial Intelligence, Bond, A. and Gasser, L. (Eds.), pp. 367-386, Morgan Kaufmann Publishers, Los Altos, CA.

[Correa 95] Correa, M. and Coelho, h. (1995). "Around the Architectural Agent Approach to model Conversations". In Lecture Notes in Artificial Intelligence 957: From Reaction to Cognition. Castelfranchi, C. and Muller, J. (Eds.), Selected Papers of 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '93, Neuchatel, Switzerland, August 1993, Springer-Verlag, Berlin, Germany.

[Coull 93] Coull, T. and Rothman P. (1993). "Virtual Reality for Decision Support Systems". In AI Expert, August 93.

[Craig 94] Craig, I.D. (1994). "A Perspective on Multi-Agent Systems" [CS-RR-273], Coventry, UK.

[Danforth 88] Danforth, Scott, and Tomlinson, Chris (1988). "Type Theories and Object-Oriented Programming. ACM Computing Surveys, Vol. 20, No. 1, March 1988.

[Dave 95] Dave, B. (1995). "Towards Distributed Computer-Aided Design Environments", CAAD, Futures '95, Singapore, Sept. 24-26 1995.

[David 95] Kurmann David (1995). "Sculptor - A Tool for Intuitive Architectural Design", CAAD Futures '95, Singapore, Vol 1, 6/32, Univ. of Singapore.

[Davis 78] Davis, C. T. Jr. (1978). "Data Processing Spheres of Control". IBM Systems Journal 17(2) pp.179-198.

[Davis 83] Davis, R. and Smith, R. (1983). "Negotiation as Metaphor for Distributed Problem Solving". In Artificial Intelligence, Vol. 20, pp. 63-109, Jan.

[Decker 95a] Decker, K., and Lesser, V. (1995). "Coordination Assistance for Mixed Human and Computational Agents", UMASS Computer Science Technical Report 95-31. Submitted to the Second International Conference on Concurrent Engineering Research and Applications. Mclean,VA.

[Decker 90] Decker, K. and Lesser, V. (1990). "A Cooperative Distributed Problem Solving". In Proc. 10th International Workshop on Distributed Artificial Intelligence, Huhns, M. N., (Ed.) Bandera, TX.

[Deitel 94] Deitel, H.M. and Deitel, P.J. (1994). "C++ How to Program", Prentice-Hall, Inc., Englewood, NJ.

[D'Inverno 96] D'Inverno, M. and Luck, M. (1996). "Formalising the Contract Net as a Goal-Directed System". In Lecture Notes in Artificial Intelligence 1038: Agents Breaking Away, Van de Velde, W. and Perram, J.W. (Eds.), Proc. of 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Eindhoven, The Netherlands, January 1996, Springer, NY.

[Dittrich 91] Dittrich, Klaus R. (1991). "Object-Oriented Database Systems: The Notion and the Issues". In Dittrich, Dayal and Buchmann (Eds.) On Object-Oriented Database Systems. Springer Verlag, Berlin.

[Dodhiawala 86] Dodhiawala, R., Jagannathan, V. and Baum, L. (1986). "Integrating Architecture for Complex System Design". In Proc. of ROBEXS, Houston, TX.

[Durfee 94] Durfee, E. H., and Rosenschein, J. S. (1994). "Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples". In Proc. of the Thirteenth International Distributed Artificial Intelligence Workshop, July 1994.], pp. 94-104.

[Durfee 92] Durfee, E. H., Damouth, D., Huber, M., Montgomery, T. A., and Sen, S. (1992). "The Search for Coordination: knowledge-Guided Abstraction and Search in a Hierarchical Behavior Space". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems, Castelfranchi, C. and Werner, E. (Eds.), Proc. of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'92, S.Martino al Cimino, Italy, July 1992, Springer, NY.

[Durfee 90] Durfee, E. H. and Motogomery, T. A. (1990). "A Hierarchical Protocol for Coordination of Multiagent Behavior". In Proc. of 8th National Conf. on Artificial Intelligence, pp. 86-93, Boston, MA.

[Durfee 88] Durfee, Edmund H. (1988). "Coordination of Distributed Problem Solvers", Kluwer Academic Publishers, Boston, MA.

[Durfee 87] Durfee, E., Lesser, V. and Corkill, D. (1987). "Cooperation Through Communication in a Distributed Problem Solving Network". In Distributed Artificial Intelligence, Huhns, M. N. (Ed.), pp. 29-58, Pitman Publishing/Morgan Kaufmann Publishers, Inc., San Mateo, CA.

[Eastman 94] Eastman, Charles (1994). "Survey of Object Oriented Models". Informal, Design and Computation Research Report, G.S.A.U.P., University of California at Los Angeles, CA.

[Eastman 92] Eastman, C. M., Chase S. C. and Assal, H. H. (1992). "System Architecture for Computer Integration of Design and Construction Knowledge", Graduate School of Architecture and Urban Planning, University of California, Los Angeles.

[El-Attar 97] Mohammad Sherif El-Attar (1997). "Application of Artificial Intelligence in Architectural Design", Ph.D. dissertation, School of Architecture, Al-Azhar University, Cairo, Egypt.

[Engeli 95] Engeli, M., Kurmann, D., Schmitt, G. (1995). "A New Design Studio - Intelligent Objects and Personal Agents in a Virtual Environment".In Proc. of ACADIA '95, Seattle, USA, October 1995, pp. 155-170.

[Engeli 96] Engeli, M., Kurmann, D. (1996). "A Virtual Reality Design Environment with Intelligent Objects and Autonomous Agents". In Proc. of Design and Decision Support Systems Conference, Spa Belgium, 1996.

[Ephrati 95] Ephrati, E. and Rosenschein, J. S. (1995). "A Framework for the Interleaving of Execution and Planning for Dynamic Tasks by Multiple Agents". In Lecture Notes in Artificial Intelligence 957: From Reaction to Cognition. Castelfranchi, C. and Muller, J. (Eds.), Selected Papers of 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '93, Neuchatel, Switzerland, August 1993, Springer-Verlag, Berlin, Germany.

[Ephrati 94] Ephrati, E. and Rosenschein, J. S. (1994). "Multi-Agent Planning as Search for a Consensus that Maximizes Social Welfare". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems. Castelfranchi, C. and Werner, E. (Eds.), Selected Papers of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 1992, Springer-Verlag, Berlin, Germany.

[Falcone 94] Falcone, R. and Castelfranchi, C. (1994). "Plan Recognition: From Single-Agent to Multi-Agent Plans". In Lecture Notes in Artificial Intelligence 1069: Distributed Software Agents and Applications, Perram, J. W. and Muller, J. P. (Eds.), Proc. of 6th European Workshop on Modeling Autonomous Agents in a Multi- Agent World, MAAMAW '94, Odense, Denmark, August 1994, Springer, NY.

[Fenves 89] Fenves, S., Flemming, U., Hendrickson, C., Maher, M.L., and Shmitt, G. (1989). "A Prototype Environment for Integrated Design and Construction Planning of Buildings". In Proc. CIFE Symposium, Stanford University, CA.

[Ferber 90] Ferber, J. and Carle, p. (1990). "Actors and Agents Reflective Concurrent Objects: A MERING IV Perspective". In Proc. 10th International Workshop on Distributed Artificial Intelligence, Huhns (Ed.), M. N., Bandera, TX.

[Fischer 92] Fischer, G. and Nakakoji, K. (1992). "Making Design Objects Relevant to the Task at Hand", Department of COmputer Science and Institue of Cognitive Science, University of Colorado, Boulder, CO.

[Finger 88] Finger, S., Fox, M., Navinchandra, D., Printz, F., and Rinderle, J. (1988). "The Design Fusion Project: A Product Life-Cycle View for Engineering Designs", Tech. Report, Carnegie Mellon University, PA.

[Flanagan 91] Flanagan, Owen (1991). "The Science of The Mind", The MIT press, Massachusetts Institute of Technology, Cambridge, Massachusetts.

[Flemming 93] Flemming, Ulrich (1993). "Artificial Intelligence and Design: A Mid-Term Review", Department of Architecture and Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

[Flemming 95] Flemming, U. and Woodbury, R. (1995). "Software Environment to Support Early Phases in Building Design (SEED): Overview". In Journal of Architectural Engineering, Dec. 95, Vol.1 No. 4.

[Foner 93] Foner, Leonard N. (1993). "Whats an Agent Anyway: A Sociological Case Study", Agents Memo 93-01. The agents Group, MIT Media Lab, Boston, MA. May 93.

[Gary 93] Gary, J. and Reuter, A. (1993). "Transaction Processing: Concepts and Techniques". Morgan Kaufmann.

[Gaines 93a] Gaines, Brian R, and Mildred L G Shaw (1993). "Eliciting Knowledge and Transferring it Effectively to a Knowledge-Based System", IEEE Transactions on Knowledge and Data Engineering, 5(1), pp. 4-14.

[Gaines 93b] Gaines, Brian R, and Mildred L G Shaw (1993). "Knowledge Acquisition Tools based on Personal Construct Psychology", Knowledge Engineering Review, 8(1), pp. 49-85.

[Gamma 95] Gamma, E., Helm, R., Johnson, R. and J. Vlissides (1995). "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Menlo Park, CA.

[Gauchel 92] Gauchel, J., Van Wyk, S., Baht, R., and Hovestadt, L., (1992). "Building Modeling Based on Concepts of Autonomy". In Artificial Intelligence in Design '92, Gero, J. S. (Ed.), Kluwer Academic Publishers, Dordrecht, The Netherlands.

[Ghedira 94] Ghedira, K. (1994). "A Distributed Approach to Partial Constraint Satisfaction Problems". In Lecture Notes in Artificial Intelligence 1069: Distributed Software Agents and Applications, Perram, J. W. and Muller, J. P. (Eds.), Proc. of 6th European Workshop on Modeling Autonomous Agents in a Multi- Agent World, MAAMAW '94, Odense, Denmark, August 1994, Springer, NY.

[Gmytrasiewicz 93] Gmytrasiewicz, Piotr J., and Edmund H. Durfee (1993). "Reasoning about Other Agents: Philosophy, Theory, and Implementation". In Proc. of The Twelfth International Workshop on Distributed Artificial Intelligence, May 1993.

[Gmytrasiewicz 92] Gmytrasiewicz, P., Durfee, E., and Wehe, D. (1992). "The Utility of Communication in Coordinating Intelligent Agents", Department of Nuclear Engineering and Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI.

[Goodwin 93] Goodwin, R. (1993). "Formalizing Properties of Agents". Technical Report CMU-CS-93-159, Carnegie-Mellon University.

[Gruber 92] Gruber, T., Tenenbaum, J. and Weber, J. (1992). "Toward a Knowledge Medium for Collaborative Product Development". In Artificial Intelligence in Design '92, Gero, J. S. (Ed.), Kluwer Academic Publishers, Dordrecht, The Netherlands.

[Gu 96] Gu, C. and Ishida, T. (1996). "Analyzing the Social Behavior of Contract Net Protocol". In Lecture Notes in Artificial Intelligence 1038: Agents Breaking Away, Van de Velde, W. and Perram, J.W. (Eds.), Proc. of 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Eindhoven, The Netherlands, January 1996, Springer, NY.

[Guessoum 96] Guessoum, Z. and Dojat, M. (1996). "A real-Time Agent Model in an Asynchronous-Object Enviroment". In Lecture Notes in Artificial Intelligence 1038: Agents Breaking Away, Van de Velde, W. and Perram, J.W. (Eds.), Proc of 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Eindhoven, The Netherlands, January 1996, Springer, NY.

[Hall 92] Hall, L. E., Macaulay, L. and O'Hare, G. (1992). "User Role in Problem Solving with Distributed Artificial Intelligent Systems". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems. Castelfranchi, C. and Werner, E. (Eds.), Proc. of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 1992, Springer-Verlag, Berlin, Germany.

[Hayes-Roth 95a] Hayes-Roth, B., L. Brownston, and R. v. Gent (1995). "Multiagent Collaboration in Directed Improvisation. " [First International Conference on Multi-Agent Systems, San Francisco CA], Knowledge Systems Laboratory.

[Hayes-Roth 95b] Hayes-Roth, B. (1995). "Agents on Stage: Advancing the State of the Art of AI. Knowledge", [Systems Laboratory, KSL-95-50].

[Hayes-Roth 95c] Hayes-Roth, B., K. Pfleger, P. Morignot, and P. Lalanda (1995). "Plans and Behavior in Intelligent Agents" [Knowledge Systems Laboratory, KSL-95-35].

[Heiler 90] Heiler, Sandra and Zdonik, Stanley (1990). "Object Views: Extending the Vision". In Proc. of the sixth International Conference on Data Engineering. Los Angeles, CA.

[Hewitt 80] Hewitt, C. and Kornfeld, B. (1980). "Message Passing Semantics", SIGART Newsletter, p. 48, Oct.

[Hewitt 77] Hewitt, C. and Yonezawa, A. (1977). "Modeling Distributed Systems". In Proc. 5th international Joint Conf. on Artificial Intelligence, Cambridge, MA.

[Holland 89] Holland, J. H., Holyoak, K. J., Nisbett, R. E. and P. R. Thagard (1989). "Induction: Processes of Inference, Learning and Discovery", The MIT press, Cambridge, MA.

[Horswill 95] Ian Horswill (1995). "Analysis of Adaptation and Enviroment". Artificial Intelligence 73(1-2), Pages 1-30.

[Jennings 98] Jennings, N. R., Sycara, K. and Wooldridge, M. (1998). "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems 1, pp. 7-38. Kluwer Academic Publishers, Boston.

[Jennings 95a] Jennings, N. R. (1995). "Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions", Artificial Intelligence, 75(2), pp. 195-240.

[Jennings 95b] Jennings, N.R., and M. Wooldridge (1995) "Applying Agent Technology". Applied Artificial Intelligence: An International Journal (issue number unkown).?

[Jennings 93] Jennings, N. R. (1993). "Commitments and Conventions: The Foundation of Coordination in Multi-Agent Systems", The Knowledge Engineering Review, 8 (3), pp. 223-250.

[Jonker 97] Jonker, C. M. and Treur, J. (1997). "Modeling an Agent's Mind and Matter". In Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Boman, M. and Van de

Velde, W.(Eds.), Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer, NY.

[Jossen 97] Joosen, W., Bijnens, S., Matthijs, F., Robben, B., Van Oeyen, J. and P. Verbaeten (1997). "Building Multi-Agent Systems with CORRELATE". In Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Boman, M. and Van de Velde, W.(Eds.), Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer, NY.

[Kalay 89] Kalay, Y. E., (1989). "Principles of Computer-Aided Design: Modeling Objects and Environments", John Wiley & Sons, Inc. New York, NY.

[Katz 91] Katz, Randy H. and Chang, Ellis E-Li (1991)."Inheritance Issues in Computer-Aided Design Databases". In Dittrich, Dayal and Buchmann (Eds.) On Object-Oriented Database Systems. Springer Verlag, Berlin.

[Kautz 94] Kautz, Henry, Selman, Bart, Coen, Michael, Ketchpel, Stevenl, and Chris Ramming (1994). "An Experiment in the Design of Software Agents" [Proc. AAAI94].

[Kearney 92] Kearney, P. J. (1992). "Experiments in Multi-Agent System Dynamics". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems. Castelfranchi, C. and Werner, E. (Eds.), Selected Papers of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 1992, Springer-Verlag, Berlin, Germany.

[Khoshafian 86] Khoshafian, S. And Copeland, G. (1986). "Object Identity". In OOPSLA Proc., 1986.

[Khedro 93] Khedro, T., Genesereth, M., and Teicholz, P (1993). "Federation of Collaborative Design Agents", CIFE projects, Stanford University, CA.

[Kinny 92] Kinny, D., Ljungberg, M., Rao, A., Sonenberg, E., Tidhar, G. and Werner, E. (1992). "Planned Team Activity". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems. Castelfranchi, C. and Werner, E. (Eds.), Proc. of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 1992, Springer-Verlag, Berlin, Germany.

[Kirsh 95] David Kirsh (1995). "The Intelligent Use of Space". Artificial Intelligence 73(1-2),Pages 31-68.

[Klein 90] Klein, M. (1990). "Mechanisms for Cooperative Problem Solving and Multi-Agent Learning in Distributed Artificial Intelligence Systems". In Proc. 10th International Workshop on Distributed Artificial Intelligence, Huhns, M. N. (Ed.), Bandera, TX.

[Kraus 90] Kraus, S. and Wilkenfeld, J. (1990). "The Function of Time in Cooperative Negotiations". In Proc. 10th International Workshop on Distributed Artificial Intelligence, Huhns, M. N., (Ed.) Bandera, TX.

[Krishnamurti 92] Krishnamurti, R. and Earl, C. (1992). "Shape Recognition in Three Dimensions", Environment and Planning B: Planning and Design, Vol. 19. pp. 585-603.

[Krishnamurti 86] R. Krishnamurti (1986). "The Mole Picture Book: On a Logic for Design". In Design Computing, Vol. 1, pp. 171-188., John Wiley & Sons, Inc.

[Kuhn 70] Thomas S. Kuhn (1970). "The Structure of Scientific Revolutions", University of Chicago, Chicago.

[Kurihara 97] Kurihara, S., Aoyagi, S. and R. Onai (1997). "Adaptive Selection of Reactive/Deliberate Planning for the Dynamic Enviroment". In Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Boman, M. and Van de Velde, W. (Eds.), Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer, NY.

[Larsi 90] Larsi, B. Larsi, H. and Lesser, V. (1990). "Negotiation and its Role in a Cooperative Distributed Problem Solving". In Proc. 10th International Workshop on Distributed Artificial Intelligence, Huhns, M. N. (Ed.), Bandera, TX.

[Lee 93] Lee, K., Mansfiels Jr., W. and Bellcore, A. (1993). "A Framework for Controlling Cooperative Agents". In IEEE, Computer 93.

[Lesser 92] Lesser, V., Durfee, E., and Corkill, D. (1992). "Cooperative Distributed Problem Solving". In The Handbook of Artificial Intelligence Vol. IV., Barr, A., Cohen, P. and Feigenbaum, E. (Eds.), Addison-Wesley Publishing Company, Inc. NY.

[Lesser 91] Lesser, V., Carven, N., and Cuetanovic, Z. (1991). "Sophisticated Cooperation in FA/C Distributed Problem Solving". In Proc. 9th National Conf. on Artificial Intelligence, Anaheim, CA, pp. 191-197.

[Lesser 88] Lesser, V. and Erman, L. (1988). "Distributed Interpretation: A Model and Experiment". In Readings in Distributed Artificial Intelligence, Bond, A. and Gasser, L. (Ed.), pp. 120-139, Morgan Kaufmann Publishers, San Matio, CA.

[Levine 88] Marvin Levine (1988). "Effective Problem Solving", Prentice Hall, Englewood Cliffs, NJ.

[Liu 95] Liu, J. and Sycara, K. (1995). "Emergent Constraint Satisfaction Through Multi-Agent Coordinated Interaction". In Lecture Notes in Artificial Intelligence 957: From Reaction to Cognition. Castelfranchi, C. and Muller, J. (Eds.), Selected Papers of 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '93, Neuchatel, Switzerland, August 1993, Springer-Verlag, Berlin, Germany.

[Lyons 95] Lyons, d. and Hendriks, A.J.(1995). " Exploiting Patterns of Interaction to Achieve Reactive Behavior", Artifical Intelligence 73(1-2), pp 117-148.

[Maes 91] Maes, P. (1991). "Situated Agents Can Have Goals". In Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, Maes, P. (Ed.), pages 49-70, MIT press, Cambridge, MA.

[Mahdavi 97] Mahdavi, A., G. Suter (1997). " On implementing a computational facade design support tool". In Environment and Planning B 24 pp. 493 - 508.

[Mahdavi 96] A. Mahdavi. (1996). " SEMPER: a new computational environment for simulation-based building design assistance". In Proc. of the 1996 International Symposium of CIB W67 Energy and Mass Flows in the life Cycle of Buildings), Vienna, Austria.

[Mamou 91] Mamou, Jean-Claude and Medeiros, Claudia Bauzer (1991) . "Interactive Manipulation of Object-Oriented Views". In Proc. of the International Conference on Data Engineering.

[Minsky 86] M. Minsky (1986) . "The Society of Mind", Simon and Schuster, New York.

[Mitchel 89] W. J. Mitchell (1989) . "A Computational View of Design Creativity", Preprints: Modeling Creativity and Knowledge-Based Creative Design, J.S. Gero and M.L Maher, (Eds.), University of Sydney, Australia.

[Monk 94] Monk, Simon (1994). "View Definition in an Object-Oriented Database", Information and Software Technology, Vol. 36, No. 9, pp. 549-554, September, 1994.

[Monk 93] Monk, Simon and Sommerville, Ian (1993). "Schema Evolution in OODBs Using Class Versioning", SIGMOD Record Vol. 22, No. 3, September 1993.

[Muller 97] Muller, Jorg P., Wooldridge, Michael J. & Jennings, Nicholas R. (EDS) 1997. "Intelligent Agents III: agent theories, architectures, and languages". In Lecture Notes in Artificial Intelligence 1193, Proc. of ECAI '96 workshop (ATAL), Budapest, Hungary, August 1996, Springer.

[Myers 93] Myers, L., Pohl, J., Aly, S., Chien, S., Cotton, J., Pohl, K., Rodriguez, T. and Snyder, J. (1993). "Object Representation and the ICADS-Kernel Design", Design Institute Report, CADRC-08-93, California Polytechnic State University, San Luis Obispo, CA.

[Nagao 95] Nagao, K., Hasida, K. and Miyata, T. (1995). "Emergent Planning: A Computational Architecture for Situated Behaviour". In Lecture Notes in Artificial Intelligence 957: From Reaction to Cognition. Castelfranchi, C. and Muller, J. (Eds.), Selected Papers of 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '93, Neuchatel, Switzerland, August 1993, Springer-Verlag, Berlin, Germany.

[Neumann 83] Neumann, Thomas (1983). "On Representing the Design Information in a Common Database". In SIGMOD Conf. on Engineering DB, IEEE, pp.81-87.

[Noh 97] Noh, S., Gmytrasiewicz, P. (1997). "Multiagent Coordination in Antiair Defense: A Case Study". In Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Boman, M. and Van de Velde, W. (Eds.), Proc. of 8th European Workshop on Modeling

Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer, NY.

[Osborn 91] Osborn, S.L. Design (1991). "Issues for Object-Oriented Database Systems". In Dittrich, Dayal and Buchmann (Eds.) On Object-Oriented Database Systems. Springer Verlag, Berlin.

[Pan 91] Pan, J. and Tenenbaum, J. (1991). "An Intelligent Agent Framework for Enterprise Integration". In IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Artificial Intelligence.

[Papamichael 96] Papamichael, K., J. La Porta, H. Chauvet, D. Collins, T. Trzcinski, J. Thorpe, S. Selkowitz (1996). "The building design advisor". In Proc. of the 1996 ACADIA conference, Tucson, Arizona, pp. 85 - 97.

[Pernici 90] Pernici, Barbara (1990). "Objects with Roles". Dipartimento di Elettronica, Politecnico di Milano, Italy.

[Pohl 99] Pohl, J.; Porczak, M.; Pohl, K.J.; Leighton, R.; Assal, H.; Davis, A.; Vempati, L.; Wood, A.; and McVittie, T. (1999). "IMMACCS: A Multi-Agent Decision Support System", Design Institute Report, CADRU-12-99, CAD Research Center, Cal Poly State University, San Luis Obispo, California.

[Pohl 97a] Pohl J., A, Chapman, K. Pohl, J. Primrose and A. Wozniak (1997). "Decision Support Systems: Notions, Prototypes, and In-Use Applications". In Technical Report, CADRU 11-94, CAD Research Center, Design and Construction Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California, September.

[Pohl 96a] Pohl, J. (1996). "Agents and their Role in Computer-Based Decision Support Systems", In Advances in Cooperative Environmental Design Systems, Pohl, J. (Ed.), focus symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 14-18 (pp. 41-54)

[Pohl 96b] Pohl, K. (1996). "Koala: An Object-Agent Architectural Design System", Master Thesis, College of Architecture; Cal Poly, San Luis Obispo, California.

[Pohl 96c] Pohl, K. (1996). "Koala: An Object-Agent Architectural Design System". In Advances in Cooperative Environmental Design Systems Pohl, J. (Ed.), focus symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 14-18 (pp. 81-92).

[Pohl 94] Pohl J., L. Myers and A, Chapman (1994). "Thoughts on the evolution of Computer-Assisted Design". In Technical Report, CADRU-09-94, CAD Research Center, Design and Construction Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, California, September.

[Pohl 93a] Pohl, J. and Myers, L. (1993). "A Distributed Cooperative Model for Architectural Design". CAD Research Center, California Polytechnic State University, San Luis Obispo, CA.

[Pohl 93b] Pohl, K. J. (1993). "MERCURY, A Real-Time Message Management Facility for Distributed Cooperative Computing Environments". In Proc. of the 4th International Symposium on Systems Research, Informatics and Cybernetics Pohl, J. (Ed.), Focus Symposium, Advances in Computer-Assisted Building Design Systems, August 2-5, Baden-Baden, Germany, pp. 155-164.

[Pohl 92] Pohl, J., Myers, L., Cotton, J., Chapman, A., Pohl, K., Chauvet, H., Snyder, J. and La Porta, J. (1992). "A Computer-Based Design Environment", Design Institute Report, CADRU-06-92, California Polytechnic State University, San Luis Obispo, CA.

[Polat 92] Polat, F. and Guvenir, H. A. (1992). "A Conflict Resolution-Based Decentralized Multi-Agent Problem Solving Model". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems. Castelfranchi, C. and Werner, E. (Eds.), Selected Papers of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 1992, Springer-Verlag, Berlin, Germany.

[Pree 95] Pree, Wolfgang (1995). "Design Patterns for Object-Oriented Software Development". Addison-Wesley, Menlo Park, CA.

[Pressman 87] Pressman, Roger S. (1987). "Software Engineering: A Practitioner's Approach". McGraw-Hill Book Company, New York, NY.

[Quadrel 91] Quadrel, R. W. (1991). "Asynchronous Design Environments: Architecture and Behavior", Ph.D. Thesis, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.

[Rasmus 95] Rasmus, Daniel W. (1995). "What's the Deal with Agents". In Object Magazine, May, pp. 71-76.

[Rasmussen 86] Rasmussen, J. (1986). "Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering", North Holland.

[Ramani 92] Ramani, A., Chande, P. and Sharama (1992). "A General Model for Performance Investigations of Priority Based Multiprocessor Systems". In IEEE Transactions on Computers, 41(6), June, pp. 747-755.

[Rowe 87] Rowe, P. G. (1987). "Design Thinking", The MIT Press, Cambridge, MA.

[Rumbaugh 91] Rumbaugh, J., Blaha, M., Premerlani,W., Eddy, F. and Lorenson, W. (1991). "Object-Oriented Modeling and Design". Prentice Hall, Englewood Cliffs, NJ.

[Russel 95] Russel, S. J. and Norvig, P. (1995). "Artificial Intelligence: A Modern Approach". Prentice-Hall Inc., Upper Saddle River, NJ.

[Schmitt 94] G. Schmitt (1994). "Scene Animation using Intelligent Objects in a Virtual Design Environment". Speedup Journal, Volume 8, Nr. 1, June 1994, pp. 14-20, CSCS, Manno, Schweiz.

[Schon 88] Schon, D. (1988). "Designing: Rules, Types and Worlds", Design Studies, 9(3), July, pp. 181-190.

[Schon 83] Schon, D. (1983). "The Reflective Practitioner: How Professionals Think in Action", Basic Books.

[Seghrouchni 96] Seghrouchni, A. E. and Haddad, s. (1996). "A Coodination Algorithm for Multi-Agent Planning". In Lecture Notes in Artificial Intelligence 1038: Agents Breaking Away, Van de Velde, W. and Perram, J.W. (Eds.), Proc. of 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Eindhoven, The Netherlands, January 1996, Springer, NY.

[Show 89] Show, M., and Whinston, A., (1989). "Learning and Adaptation in Distributed Artificial Intelligence Systems",In Distributed Artificial Intelligence, Gasser, L. and Huhns, M. (Eds.), Vol. II, pp. 413-429, Pitman Publishing, Los Altos, CA.

[Shriver 87] Shriver, Bruce and Wegner, Peter (Eds.) (1987). "Research Directions in Object-Oriented Programming", Cambridge, Massachusetts: The MIT Press.

[Sierra 97] Sierra, C., Faratin, P. and N. r. Jennings (1997). "A Service-Oriented Negotiation Model between Autonomous Agents". In Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Boman, M. and Van de Velde, W. (Eds.), Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer, NY.

[Simon 69] Simon, H. (1969). "The Science of Artificial", MIT Press, Boston, MA.

[Singh 98] Singh, Munindar P., Rao, Anand & Wooldridge, Michael J. (EDS.) 1998. "Intelligent Agents IV: Agent Theories, Architectures, and Languages". In Lecture Notes in Artificial Intelligence 1365, Proc. of the 4th International Workshop, ATAL' 97, Providence, Rhode Island, USA, July 1997, Springer.

[Singh 97] Singh, Munindar P. (1997). "Commitments Among Autonomous Agents in Information-Rich Enviroments". In Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Boman, M. and Van de Velde, W. (Eds.), Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer, NY.

[Shoham 93] Shoham, Y. (1993). "Agent_Oriented Programming", Artificial Intelligence 60, pp. 51-92.

[Smith 96] Smith, Ian F. C., Kurmann, D. and Schmitt, G. (1996). "Case Combination and Adaptation of Building Spaces". Federal Institute of Technology (ETH), CAAD, Zurich, Switzerland.

[Smith 94] Smith, Faltings B.(1994). "Spatial Design of Artifacts using Cases". In Proc. of the 10th International Conference on Artificial Intelligence for Applications, IEEE, San Antonio, 1994, pp. 70-76.

[Smith 77] Smith, John Miles and Smith, Diane C.P. (1977). "Database Abstractions: Aggregations and Generalizations", ACM Transactions on Database Systems. Vol. 2, No. 2, June 1977, pp. 105-133.

[Steels 90] Steels, L., (1990). "Cooperation Between Distributed agents Through Self-Organization". In Proc. 1st. European Workshop on Modeling Autonomous Agents. In Multiagent World, pp. 175-196., London, England.

[Steiner 93] Steiner, D., Alastair, B., Kolb, M. and Leri, c. (1993). "The conceptual Framework of MAI^{2}L". In Lecture Notes in Artificial Intelligence 957: From Reaction to Cognition. Castelfranchi, C. and Muller, J. (Eds.), Selected Papers of 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '93, Neuchatel, Switzerland, August 1993, Springer-Verlag, Berlin, Germany.

[Stirling 92] Stirling, W. (1992). "Multi-Agent Coordinated Decision-Making using Epistemic Utility Theory". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems. Castelfranchi, C. and Werner, E. (Eds.), Selected Papers of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 1992, Springer-Verlag, Berlin, Germany.

[Stroustrup 97] Bjarne Stroustrup (1997). "The C++ Programming Language". Third Edition, Bjarne Stroustrup, Addison Wesley, Reading, MA.

[Sunderam 90] Sunderam, V. (1990). "PVM: A Framework for Parallel Distributed Computing". Concurrency: Practice & Experience, 2(4) Dec.

[Sycara 89] Sycara, K. (1989). "Multi agent Compromise via Negotiation". In Distributed Artificial Intelligence, Gasser, L. and Huhns, M. (Eds.), Vol. II, pp. 119-138. Pitman Publishing, Los Altos, CA.

[Talukdar 90] Talukdar, S., and deSouza, P. (1990). "Asynchronous Teams". In Proc. Second SAIM Conf. on Linear Algebra: Signals, Systems and Control, San Francisco, CA, Nov.

[Tecuci 98] Tecuci, G. (1998). "Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies". Academic Press, San Diego, CA.

[Timothy 97] Timothy, J. N. and Jennings, R. N. (1997). "Generating States of Joint Commitment between Autonomous Agents". In Lecture Notes in Artificial Intelligence 1441: Agents and Multi-agent System: Formalizations, Methodologies, and Applications, Wobcke, W., Pagnucco, M. and Zhang C. (Eds.). Based on the AI'97 Workshop on Commonsense Reasoning, Intelligent Agents, and Distributed Artificial Intelligence. Perth, Australia, December 1997, Springer NY.

[Tokoro 94] Tokoro, Mario (1994). "Agents: Towards a Society in Which Humans and Computers Cohabitate". In Lecture Notes in Artificial Intelligence 1069: Distributed Software Agents and Applications, Perram, J. W. and Muller, J. P. (Eds.), Proc. of 6th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '94, Odense, Denmark, August 1994, Springer, NY.

[Urzeli 92] Urzelai, K. and Garijo, F. J. (1992). "MAKILA: A Tool for the Development of Cooperative Societies". In Lecture Notes in Artificial Intelligence 830: Artificial Social Systems. Castelfranchi, C. and Werner, E. (Eds.), Selected Papers of 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW '92, S. Martino al Cimino, Italy, July 1992, Springer-Verlag, Berlin, Germany.

[Wade 77] Wade, J. W. (1977). "Architecture, Problems and Process: Architectural design as Basic Problem-Solving Process", John Wiley & Sons, Inc., New York, NY.

[Wagner 96] Wagner, Gerd (1996). "A Logical and Operational Model of Scalable Knowledge- and Perception-Based Agents". In Lecture Notes in Artificial Intelligence 1038: Agents Breaking Away, Van de Velde, W. and Perram, J.W. (Eds.), Proc. of 7th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'96, Eindhoven, The Netherlands, January 1996, Springer, NY.

[Watson 90] Watson, A.S. (1990). "CAD Data Exchange in Construction". In Proc. of the Institution of Civil Engineers Part 1 - Design and Construction, Vol. 88, pp. 955-969, Dec. 1990.

[Webster 94] Webster, S. (1994). "An Annotated Bibliography for Object-Oriented Analysis and Design". Information and Software Technology, Vol. 36, No. 9, pp. 569-582, September, 1994.

[Werkman 92] Werkman, K. J. (1992). "Multiple Agent Cooperative Design Evaluation Using Negotiation". In Artificial Intelligence in Design '92, Gero, J. S. (Ed.), Kluwer Academic Publishers, Dordrecht, The Netherlands.

[Werner 94] Werner, Eric (1994). "What Ants Cannot Do". In Lecture Notes in Artificial Intelligence 1069: Distributed Software Agents and Applications, Perram, J. W. and Muller, J. P. (Eds.), Proc. of 6th European Workshop on Modeling Autonomous Agents in a Multi- Agent World, MAAMAW '94, Odense, Denmark, August 1994, Springer, NY.

[Wiederhold 86] Wiederhold, G. (1986). "Views, Objects and Databases", IEEE Computer, December 1986.

[Wileden 90] Wileden, Jack C., Clarke, Lori A., and Wolf, Alexander L. A Comparative (1990). "Evaluation of Object Definition Techniques for Large Prototype Systems". ACM Transactions on Programming Languages and Systems, Vol. 12, No. 4, October, 1990.

[Wobcke 97] Wobcke, Wayne (1997). "Agency and Logic of Ability". In Lecture Notes in Artificial Intelligence 1441: Agents and Multi-agent System: Formalizations, Methodologies, and Applications, Wobcke, W., Pagnucco, M. and Zhang C. (Eds.). Based on the AI'97

Workshop on Commonsense Reasoning, Intelligent Agents, and Distributed Artificial Intelligence. Perth, Australia, December 1997, Springer NY.

[Woodbury 89] Woodbury, R. F. (1989). "Searching for Designs: Paradigm and Practice", Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.

[Wooldridge 97] Wooldridge, M. and Haddadi, A. (1997). "Making it up as they Go Along: A Theory of Reactive Cooperation". In Lecture Notes in Artificial Intelligence 1441: Agents and Multi-agent System: Formalizations, Methodologies, and Applications, Wobcke, W., Pagnucco, M. and Zhang C. (Eds.). Based on the AI'97 Workshop on Commonsense Reasoning, Intelligent Agents, and Distributed Artificial Intelligence. Perth, Australia, December 1997, Springer NY.

[Wooldridge 96] Wooldridge, Michael, Muller, Jorg p. & Tambe, milind (EDS) 1996. "Intelligent Agents II: agent theories, architectures, and languages". In Lecture Notes in Artificial Intelligence 1037, Proc. of IJCAI '95 workshop (ATAL), Montreal, Canada, August 1995, Springer

[Wooldridge 95] Wooldridge, Michael J., Jennings, Nicholas R. (EDS.) 1995. " Intelligent Agents". In Lecture Notes in Artificial Intelligence 890, Proc. of ECAI-94 workshop on Agent Theories, Architectures, and Languges, Amsterdam, The Netherlands, August 1994, Springer.

[Wooldridge 95b] Wooldridge, M. and Jennings, N. (1995). "Towards a Theory of Cooperative Problem Solving". In Lecture Notes in Artificial Intelligence 1069: Distributed Software Agents and Applications, Perram, J. W. and Muller, J. P. (Eds.), Proc. of 6th European Workshop on Modeling Autonomous Agents in a Multi- Agent World, MAAMAW '94, Odense, Denmark, August 1994, Springer, NY.

[Ygge 97] Ygge, Fredrik and Akkermans, Hans (1997). "Making a Case for Multi-Agent Systems". In Lecture Notes in Artificial Intelligence 1237: Multi-Agent Rationality, Boman, M. and Van de Velde, W. (Eds.), Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW'97, Ronneby, Sweden, May 1997, Springer, NY.

[Yoko 90] Yoko, M. Ishida, T. and Kuwabara, K. (1990). "Distributed Constraint Satisfaction for DAI Problem Problems". In Proc. 10th International Workshop on Distributed Artificial Intelligence, Huhns, M. N. (Ed.), Bandera, TX.

[Yokoyama 91] Yokoyama, T. (1991). "A Parallel Cooperative Problem-Solving System with Intelligent Blackboard". Tech. Report, Institute for New Generation Computer Technology, Tokyo.

[Zhang 96] Zhang, Chengqi, Lukose, Dickson (EDS.) 1996. "Distributed Artificial Intelligence: Architecture and Modeling". In Lecture Notes in Artificial Intelligence 1087, Proc. of the first Australian workshop on DAI, Canberra, ACT, Australia, November 1995, Springer.

[Zucker 92] Zucker, J. and Demaid, A. (1992). "Modeling Heterogeneous Engineering Knowledge as Transactions Between Delegating Objects". In Artificial Intelligence in Design '92, Gero, J. S., (Ed.) Kluwer Academic Publishers, Dordrecht, The Netherlands.

A Appendix: Terms and Definitions

For ease of reference, I have collected, into this appendix, the definitions that are pertinent to a computational decision making environment based on an object-agent model. The validity of these definitions are limited to the scope of this thesis. Other definitions local to the context in which they arise are given in the appropriate sections of this dissertation.

The following terms are general and are used in all chapters.

A.1 Decision Makers, Designers and Artifacts.

Def. A.1.1 Decision Maker

Is a principal **agent**[1] in an environment.

The decision maker manipulates representations of a problem state and guides the other agents efforts to incrementally changing a current problem state toward the **goal state**.

The distinction between a decision maker and other agents lies in the fact that the decision maker possesses both intention and ability to exercise judgement beyond heuristics.

Def. A.1.2 Designer

Is a decision maker who manipulates representations of an **artifact** being designed to reach an acceptable **design state**.

1. In any definition through Appendix A., the first appearance of a term that refers to a successor definition is in bold.

Def. A.1.3 Artifact

A real world product.

Example. In an architectural context, a building is an artifact.

The following terms are used in Chapters 3 through 7.

A.2 Data-Objects.

Def. A.2.1 Data-object

A data-object is a representation of an artifact or some part of it**.**

A data-object generally contains geometric and non-geometric information about the artifact. The information contained in a data-object is accessible by any agent, but data-objects cannot access external information.

Def. A.2.2 Data-Object Type

A collection of data-objects that identify a relationship.

Example. In an architectural context, a wall type is a data-object type.

Def. A.2.3 Sub-Data-Object

A data-object that is a constituent of another data-object in a **data-object hierarchy**. It is a sub-data-object of this particular data-object.

Def. A.2.4 Super-Data-Object

A data-object that is a container of a another data-object in a data-object hierarchy. It is a super-data-object of this particular data-object.

Def. A.2.5 Joint Data-Object

A data-object that is a sub-data-object of more than one data-object in the data-object hierarchy.

Example. A shared wall between two spaces can be a joint data-object.

Def. A.2.6 Active and Passive Data-Object

A data-object is active if there is an **object-agent** for it; and passive otherwise.

Def. A.2.7 Data-Object State

The values, at a any given time, of the attributes and relationships (to other data-objects) of a data-object.

Only agents can alter the data-object state.

Def. A.2.8 Data-Object Goal State

A final data-object state.

Def. A.2.9 Design State

The collective state of all data-objects in an environment.

This is distinct from the 'state' of the environment in that it reflects the data values as can be seen by the agents at any given time. Note that the environment may have object-agents that may be in the process of changing data values. These changing data values are not part of the design state.

Def. A.2.10 Goal State

A final design state.

A.3 Agents

Def. A.3.1 Agent

An entity with the ability to initiate **action**s, perform tasks, and interact with other agents in the environment.

Agents are executable. Any agent executing a task is bound to return execution results or error messages. Therefore, agency incorporates a degree of liability. The object-agent model distinguishes three types of agents: decision maker, system-agent and object-agent.

Def. A.3.2 Object-Agent

An object-agent is an agent that represents a data-object.

An object-agent is a temporal version of its data-object; it contains prototypical and domain specific knowledge of its data-object type.

Note that the distinction between object agency and methods is that the object-agent has the ability to alter the state of any data-object, whereas methods may only alter the state of their associated data-objects.

Def. A.3.3 Object-Agent Type

A collection of object-agents of the same data-object type.

Example. In an architectural context, a wall object-agent type is an object-agent type.

Def. A.3.4 Sub-Agent

An agent that is assigned a task by another agent.[2] The agent is a sub-agent within the context of the assigned task.[3]

Def. A.3.5 Super-Agent

An agent that has assigned a task to another agent. The agent is a super-agent within the context of the assigned task.[4]

Def. A.3.6 Composite-Agent

A composite-agent is an object-agent that is created from more than one data-object. It has information that belongs to more than one data-object.[5]

Def. A.3.7 Joint-Agent

A joint-agent is an object-agent of a joint data-object.

Def. A.3.8 System-Agent

A system-agent is an agent that performs a set of related domain specific tasks for other agents. A system-agent has local coordination knowledge, and internal procedures.[6]

---

2. Notice that being a sub-agent is task dependent while being a sub-data-object is representation dependent (based on the data-object hierarchy). This, as well, applies for super-agent and super-data-object.
3. Note that an agent can simultaneously be a sub-agent of more than one agent, that is, it may concurrently perform different tasks assigned to it by different agents.
4. Note that an agent can simultaneously be a super-agent to more than one agent, that is, an agent may have sub-agents that are concurrently performing different tasks.
5. The initial set of data-object types that a decision maker uses may not be sufficient for each situation. Composite-agents represent an important concept in this regard. New data-objects or new features of existing data-objects may emerge during the decision making process. Combining information from multiple data-objects (or from the decision maker) might allow the decision maker to progressively realize new data-objects that are more suitable to the current stage. This suggests that it might be possible to create new data-object types based on the information collected in a composite-agent.

Def. A.3.9 System-Agent Type

A set of domain related system-agents.[7]

Def. A.3.10 Activation

Activation is a task assigned by an agent to bring an entity to participate in a current session.

To activate a system-agent is to load the system-agent into the current session. To activate a data-object, when passive, is to create an object-agent.[8]

Def. A.3.11 Deactivation

Deactivation is a task assigned by an agent to remove an agent from the current decision making session.

To deactivate a system-agent is to unload the system-agent from the session. To deactivate an object-agent is to destroy the object-agent.[9]

6. A an expert-agent can be an expert system or a procedural program, etc. The internal knowledge in an expert system, for instance, is its inference engine and knowledge-base which contains an initial set of facts and rules. Though the facts in working memory may change (to a different set of facts), the inference engine, and the initial set of facts and rules typically do not change after the termination of the execution state. Some expert systems are designed to generate new rules using different techniques such as machine leaning. An object-agent-based model does not impose any restrictions on the possible use of these types of expert systems as system-agents. In a procedural program, the internal knowledge is the algorithms and information structures; these do not change after the termination of the execution state. In the object-agent-based model, system-agents do not change their internal procedures.
7. A system-agent is typically an expert-agent or a utility-agent. An expert-agent, for instance, evaluates, generates, synthesizes, analyzes, recognizes, criticizes, recommends, explains, modifies, and optimizes. A utility-agent for instance, searches for requested information; provides interface facilities between the user, data-objects and agents.
8. A system-agent can be loaded into a session even if it is not necessarily engaged in the current activity by the other agents. The system-agent in this case is considered activated. In other words, for a system-agent, it is activated whenever it is loaded . For an object-agent if it is activated it is engaged in performing tasks, otherwise it should not exist. The data-object can have the capability to observe the environment, therefore, there is no need for an idle object-agent. Activation of a data-object is, therefore, only associated with task assignment (self assignment or by other agents).
9. A data-object state must be updated before the deactivation of its object-agent.

A.4 Task Execution

Def. A.4.1 Action

Acts executed by an **agent**.

Def. A.4.2 Action Type

A set of related actions. An action type is *simple* whenever an agent executes a single act, and *complex*, otherwise.

Def. A.4.3 Task

An assignment of service to be performed by an agent. A task can be *simple* or *complex* depending upon its action type.

Note that tasks are assigned, actions are not.

Def. A.4.4 Task Type

A set of related tasks.

Typically, task are related by *context*. Examples of context include evaluation, generation and implementation.

Def. A.4.5 Direct Task

A task performed by an assigned agent (i.e., without the need to decompose and distribute the task to sub-agents).

Def. A.4.6 Indirect Task

A task that is performed by a sub-agent as a result of decomposition or distributing a another task.

Def. A.4.7 Plan

An ordered sequence of actions[10] towards a state.

Def. A.4.8 Task Handling Protocol

A plan executed by one or more agents to perform a task.

Task handling protocols may be general or specific. General protocols are independent of both object-agent types and the task domain. Specific

10. The execution of a plan may be sequential or concurrent, and may involve one or more agents. Typically, the plan of an object-agent is to attain a desired data-object state.

protocols are, typically, sets of parameters to be used by the task type protocols during the execution of a task (and mainly represents the domain effect on the task in hand (see Chapter 5 for details).

Performing tasks

An object-agent employs a set of general task handling protocols for each task type. Its object-agent type and the task domain add an additional layer of specificity to the sequence of actions. General and specific protocols represent short term planning capabilities of an object-agent. Using such protocols an object-agent can obtain services from other agents, distribute tasks to other object-agents, manage other agents,[11] and run conflict handling sessions, about its data-object state, that involve multiple agents.

Def. A.4.9 Interaction protocol

Is a set of data-object type-specific instructions that enables an object-agent (of this data-object-type) to interact with other agents of the environment in during the execution of a task.

A.5 Task Decomposition

Def. A.5.1 Data-Object-Hierarchy

The global data-object class hierarchy used by the designer at any point in time.

The data-object-hierarchy may be compiled by the designer or provided as an exemplar hierarchy in the environment data-base, and possibly modified by the designer and saved for later use.

Def. A.5.2 Object-Agent-Hierarchy

A set of data-object classes each of which is a constituent of the object-agent or a constituent of a sub-data-object of the object-agent.

11. Within the context of a task assigned by one agent to another, some hierarchies may require the object-agents executing a task to follow certain order of execution. This hierarchy is task dependent and not class dependent. The task hierarchy is established when an object-agent is assigned a task (see Chapter 5.2, Making the Activation List). An object-agent may be involved in more than one task hierarchy at any given time. That occurs when an object-agent is involved in performing multiple tasks concurrently.

Def. A.5.3 Max-Domain-Hierarchy

The set of all eligible classes for task decomposition with respect to this particular domain. This set is defined by the P_Domain decomposition protocol (see Chapter 5 for details and Section 5.2 for specific examples) of the task in hand.

Def. A.5.4 Min-Domain-Hierarchy

The minimum set of data-object-classes necessary to execute an assigned task. This set is defined by the P_Domain decomposition protocol of the task in hand.

Def. A.5.5 Domain-Hierarchy$_{bottom}$

A class or a set of classes which represent the lower boundary of a max-domain-hierarchy .

Def. A.5.6 Domain-Hierarchy$_{top}$

A data-object class which represent the top boundary of the max-domain-hierarchy.

Def. A.5.7 Leaf-Data-Object

A data-object class at the lower end of each branch of a data-object-hierarchy

Def. A.5.8 Data-Object$_{classification}$

A data-object class used for classifying the results of executing an assigned task.

Example. In evaluating the cost of a BFloor-data-object per Room-data-object (see the example in chapter 4), the Room-data-object class is the data-object$_{\text{classification}}$ of this task.

Def. A.5.9 Activation$_{list}$

A set of data-object classes were their instances are to be activated to execute sub-tasks during the executing of a task. The activation$_{\text{list}}$ is always a subset of the max-domain-hierarchy$_{\text{list}}$.

Def. A.5.10 $Skip_{list}$

A list of data-object classes to be skipped during the activation of sub-data-objects of an object-agent-hierarchy. This too is a subset of the max-domain-hierarchy.

Def. A.5.11 Activation Order

The order of activating data-objects in an object-agent-hierarchy during the execution of a task.

Def. A.5.12 Task Dependent Hierarchy

The hierarchy of the data-objects in the $activation_{list}$.

This is the hierarchy of data-objects that participate in the decomposition with respect to the task in hand.

A.6 Conflict Handling

Def. A.6.1 Conflict

An attribute value that is being modified to a new value causing an interested data-object or expert-agent not to be satisfied.

Def. A.6.2 Conflict handling

The process by which a conflict is detected and resolved.

Def. A.6.3 Conflict detection

The process by which the decision maker becomes aware of a conflict.

Def. A.6.4 Conflict Resolution

The process by which a decision maker is able to arrive at a set of acceptable values for all interested parties as well as for the attribute being modified.

In a sense, conflict resolution is a series of local bilateral evaluation sessions involving the interested data-objects and expert-agents where the decision maker examines various data-object attribute values to either resolve the conflict or reach an acceptable state of all the parties involved. Each evaluation session involves the decision maker and one of the interested parties. Evaluation results are communicated to the decision maker, no direct communication amongst the interested parties regarding the conflict is

permitted. Validating the conflict resolution results is the sole responsibility of the decision maker.

Def. A.6.5 Conflict Prevention/Control

The process by which a decision maker is able to avoid or reduce the number of conflict handling sessions about an attribute value that is being modified.

Def. A.6.6 Attribute Interest$_{list}$

A list of data-object attributes and expert-agents that are interested in this attribute value. Each member of the list is paired with an **Interest Context**.

Def. A.6.7 Interest Context

The reasons of which an agent, or a data-object attribute is interested in another data-object attribute (e.g., Wall-data-object width for Room-data-object acoustics). The context may also include the recommended expert-agent to interact with in respect to the focus of this context.

Def. A.6.8 Interested Attribute

An data-object attribute that is registered in the interest$_{list}$ of another data-object attribute.

Def. A.6.9 Interested Data-Object

A data-object with at least one attribute registered in the interest$_{list}$ of an another data-object attribute.

Note that, within the same data-object, an attribute may be registered in the interest$_{list}$ of another attribute.

Def. A.6.10 Interested Expert-Agent

An expert-agent that is registered in the interest$_{list}$ of data-object attribute.

Def. A.6.11 Conflict Focus

An object-agent that is currently providing an interest$_{list}$ for a conflict check**.**

Def. A.6.12 Conflict Zone

The two object-agents involved in a conflict handling session.

Def. A.6.13 Direct Conflict Handling

A conflict handling session involving two object-agents one of which is the conflict focus.

Def. A.6.14 Indirect Conflict Handling

A conflict handling session involving two object-agents none of which is the conflict focus.

Accordingly, the conflict zone does not necessarily include the conflict focus.

A.7 Abbreviations

In this dissertation, the following abbreviations are employed:

Data-Object $\rightarrow$ DO

Object-Agent $\rightarrow$ OA

System-Agent $\rightarrow$ SA

Expert-Agent $\rightarrow$ EA (a specialization of a System-Agent)

Utility-Agent $\rightarrow$ UA (a specialization of a System-Agent)

B Appendix: Actions, Tasks and Interactions

Agents interact whenever one communicates to another. They trigger others to take actions. The DA may trigger a chain of actions to perform certain task. The following action types are applicable to the various agent types:

Action Types

Actions are either simple or complex;

Simple actions require the execution of a single act by sending a message (or simply use an object method of another DO). Simple actions may include but not limited to:

TABLE B.1. Simple Actions

Action	*Description*
activate	a DO or load an agent (i.e., EA or UA)
assign	a task, or *request* service from an agent
query	information, or *request* data.
provide	results (evaluation, query results, etc.), alternative values for task reassignment, an $interest_{list}$, a DO clone, a confirmation or validation.

'Activate', 'assign', 'query' and 'request' are examples of simple actions initiated by an agent. 'Provide' is an action taken by an agent in response to an action by another agent. In other words, it is a **simple reaction**.

Complex actions require the execution of a sequence of simple or complex actions. Such actions can be initiated by the agent when needed or can be a reaction to one or more actions by other agents (a **complex reaction**). An agent

executes an action using its problem solving protocols (see Chapter 6). Complex actions may include but not limited to:

TABLE B.2. Complex Actions

Action	*Description*
generate	geometric/non-geometric information (layouts, recommendations etc.)
search	for requested information; *recognize* geometric and non-geometric patterns or configurations.
detect	inadequacy of an attribute value; conflicted or interfered attribute values.
modify	attribute values of an existing DO, or add/delete DOs; evaluation criteria used by an EA.
plan	activities conduct by the agent itself; activities conduct by the other agents (as the case in task decomposition scenario.)
validate	results of tasks executed by other agents.
update	attribute values and relations of DOs in respect to validated task results.

OA Task Types

An OA can be assigned one of five task types (all of which are complex actions since they require the implementation of a sequence of simple and complex actions):

- evaluation
- recommendation
- generation
- conflict handling
- implementation

The OA performs a set of actions to accomplish an assigned task:

For an **evaluation** task, an OA may take actions of any of the following kind.[1]

TABLE B.3. Evaluation Task Actions

Action	*Description*
assign	a task to a query-agent to search for certain values; a task to an EA to evaluate the current DO state.
detect	inadequacy of a current attribute value. candidates of conflict about an attribute value being modified.
provide	a requester with evaluation results; a requester with query results; a DA with a warning about a detected inadequacy (of DO attribute values or relations) or a list of candidates of conflict.
plan	a sequence of task assignments (e.g., in a task that requires decomposition).

For a **recommendation or generation**, an OA may take one of the following actions.

TABLE B.4. Recommendation and Generation Task Actions

Action	*Description*
assign	a task to an EAs to evaluate or provide alternatives.
query	information from other DOs.
generate	proposals for alternative attribute values, relations, DOs or arrangement of DOs (including new DOs).
provide	a requester of recommendation generated; a requester with query results.

1. These actions are not listed in order, the order by which these actions are executed is specified in the problem solving protocols of each DO type. These sequence of actions may be changed or repeated according to the task being performed.

For a **conflict handling**, an OA may take one of the following actions.

TABLE B.5. Conflict Handling Task Actions

Action	*Description*
provide	a DA with an $interest_{list}$ (DOs and EAs); a DA with a warning about a detected conflict.
plan	a conflict handling session regarding its own attributes (which may be initiated as a result of the new values generated during the current conflict handling session).
detect	a conflict between a recommended and a current DO attribute values (or relation).

For an **implementation**, an OA may take actions of the following kind.

TABLE B.6. Implementation Task Actions

Action	*Description*
assign	a task to an EA (e.g. CAD-agent) to **modify** the OA current state according to the recommendation validated by the DA;[a]
request	validation from the DA before the implementation.
provide	a confirmation of implementation.

a. The modifications may include, adding/removing DOs or changing attribute values of existing ones, and updating the necessary relations of such attributes.

Agent Interaction Types

The OA-based environment has three types of agents; DAs, SAs (which includes EAs and UAs), and OAs. An agent may interact with any type of agents. The types of agents involved in an interaction define the type of interaction. Interaction types constitute the action types that can be taken by the agents involved in an interaction (directly or via an interface-agent).

Five interaction types among agents are identified in Figure B.1. The arrow indicates the sense of the interaction where the first agent type takes action in interacting with the second agent type. The interaction types are listed below.

- **OA-OA** interactions
- **OA-SA** interactions
- **SA-SA** interactions
- **DA-OA** interactions.

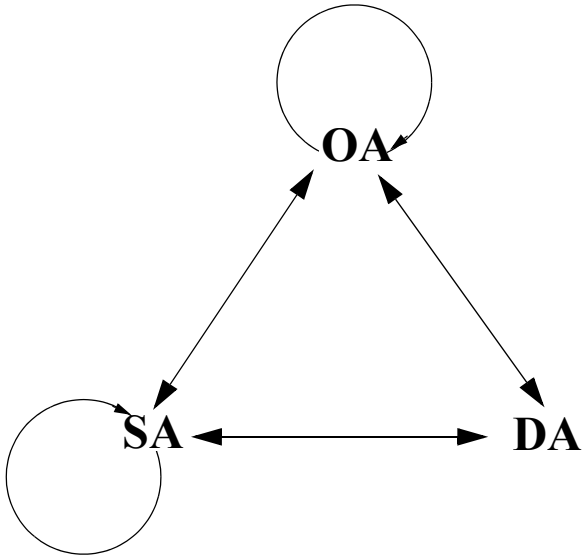


FIGURE B.1.
The Interaction Types.

- **DA-SA** interactions

The following table illustrate the possible agent interaction types.

TABLE B.7. Types of Agent Interaction

Interaction	*Action*	*Description*
OA $\rightarrow$ OA	assign	query requests, evaluation, recommendation, generation, conflict handling or implementation tasks
	provide	query results
	validate	results of directly assigned tasks
OA $\rightarrow$ SA	assign	query requests, evaluation, generation, recommendation, implementation or activation/deactivation tasks
	provide	query results
SA $\rightarrow$ OA	assign	query requests
	provide	query, evaluation, recommendation, generation, and implementation results

TABLE B.7. Types of Agent Interaction

Interaction	*Action*	*Description*
SA → SA	assign	query requests, or activation/deactivation tasks
	provide	query results
DA → OA	assign	query requests, an evaluation, generation, recommendation, conflict handling, implementation and activation/deactivation tasks
	provide	requested information (not available in the environment)
	validate	results of directly or indirectly assigned tasks
	modify	attribute values, performance criteria and OA authorizations
	plan	activities (synchronize, freeze, resume)
OA → DA	assign	query requests
	provide	query and task results and $interest_{lists}$
	request	validation of task results
DA → SA	assign	query requests, evaluation, generation, recommendation, implementation, or activation/deactivation tasks
	provide	query results,
	validate	assigned task results
	modify	evaluation criteria (used by an EA), and activation status
	plan	activities (synchronize, freeze, resume)
SA → DA	assign	query requests
	request	validation of task results
	provide	query, task results