Gero JS and Brazier FMT (eds) (2002) *Agents in Design 2002*
Key Centre of Design Computing and Cognition, University of Sydney, pp. 3-22.

CAN DOORS AND WINDOWS BECOME DESIGN TEAM PLAYERS?

SAFWAN ALY
i2Technologies Inc USA

and

RAMESH KRISHNAMURTI
Department of Architecture
Carnegie Mellon University USA

Abstract. In an architectural design session, suppose design objects such as doors, windows and rooms can look after themselves, what kind of recommendations would a designer get? What is the nature of a design environment that facilitates such interactions? Where would a design object acquire the knowledge that allows it to interact intelligently? How would such localized recommendations be aggregated to support global design decisions made by the designer? This paper investigates these questions through the notion of objects as agents in design.

1. Objects and Agent in Design

Computational tools for decision support are typically stand-alone, often designed to provide assistance with respect to a single aspect in the decision-making process. Designers use these tools to generate alternative solutions, to model and simulate the behavior of designed artifacts, and to produce design documents, particularly, where decision-making is integral to the activity. Stand-alone tools provide design assistance, but not without pitfalls: each tool requires designers to commit to a schema of representation. In order to explore various aspects of a design, the same artifact would have to be represented differently according to the schema of the computational tool.

Designers use modeling and generative tools to produce a model of the designed artifact, which other tools simulate the behavior of the model within the same shared representation. Such design environments are typically described as multi-agent decision-making environments. The agents include the designer(s) and computational applications each encompassing a specific domain expertise.

Aly (2000) has investigated enhancements to the design of computational assistants in multi-agent design environments that use shared representation schemes. He proposed expanding the notion of agency to the design objects; these agents then interact with other agents in the execution of design tasks relating to the objects. We call this the *objects-as-agents* approach, where objects are selectively activated to participate in decision-making sessions to execute tasks regarding their immediate design states.

An *object-agent* (OA) is a design object that is activated to perform tasks. In an OA-based design environment, domain applications are global problem solving nodes, OAs are local coordination and management nodes, and, collectively, the designer(s) act as coordinator and final judge. The designer orchestrates this fine-grained agent environment through incremental interactions until the model arrives at an acceptable design state. Vital to the success of an OA-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. This paper explores some of these issues associated with OA-based design environments.

2. Changing the Players Role

An expert application, representing domain knowledge of a real world expert, is an active player, whereas a design object, representing a real world object, is passive. Active players possess pertinent knowledge to manipulate passive players. That is, agents manipulate objects and decision-making is a characteristic of agents. However, design environments made up of active and passive players suffer from some of the following problems, though headway has been claimed towards their resolution, e.g., in the IMMACCS system (Pohl et al 1999).

- Elimination of rich sources of design information from local nodes;
- Difficulties in identifying problem sources in their immediate settings;
- Loss of capability in handling problems at the local level;
- Inability to handle design problems with a high level of abstraction, or the need for relatively excessive information (mostly irrelevant) while dealing with relatively smaller design problems.

3. An OA-Based Design Environment

In a computational design environment, *design objects* (DOs) represent the artifact being designed at various levels of abstraction. To act as agents, the objects should be endowed with task management and problem solving knowledge. DO agentification potentially enriches the design environment with adequate information about the state of each DO with respect to its performance requirements. Further, design tasks can be broken down into

smaller self-regulating sub-tasks that are easier to understand and manage. These sub-tasks can be distributed to the applicable OAs and executed by the OAs as needed.

To illustrate the basic premise of the OA approach, consider a room, a domain-object, that can find its required or prototypical daylighting level from an architectural program (or a prototypical database), which interacts with a daylighting application to evaluate its current daylighting level. During execution of this daylighting evaluation, the room decomposes and delegates the task to its openings (e.g., to determine the amount of daylight coming through each opening) and room surfaces (e.g., for reflection). A window may be found to admit less daylight than anticipated due to its glazing type. Such detailed information can be communicated to the designer to modify the glazing area, or to 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 have to be addressed, for instance, on the degree of agent autonomy granted to an OA. In the above example, one would have to address how, with reference to the room, is a daylight evaluation task initiated: Is it by the room itself (as an OA)? Or is it assigned to the room by another environment agent?

4. Creating Object-Agents

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 protocols representing the behavior expected from such DO type. Three types of protocols are loaded; object-type, task domain and task type/focus protocols. The created OA acts on behalf of the DO performing tasks assigned to the DO and reporting to the designer and other interested agents when required. The following scenario illustrates how an OA can be created taking an object-oriented implementation of activation.

- An agent (or a designer) requests the activation of a Room_DO to perform a daylighting evaluation task.
- The request received by the 'Object-Agent' class in turn creates a Room_OA instance.
- The Room_OA requests a clone of the Room_DO (a copy of the exact Room_DO being activated).
- Through the clone, the Room_OA would load interaction protocols related to its DO-type, creating an instance of the 'Room_Object_ Type_Protocol' class.

- The requester agent assigns the task to the created Room_OA, which in turn, loads the related task-type protocols, making an instance of the 'Evaluation_TaskType_Protocol' class. This enables the Room_OA to proceed with the execution of the evaluation task.
- The Room_OA loads the domain specific daylighting protocols, making an instance of the 'Daylighting_Domain_Protocols' class, which provides the daylighting parameters, needed by the Room_OA for decomposing, aggregating and sorting the daylighting task, all of which are subclasses of the 'Daylighting_Domain_Protocol' class.
- The Room_OA is now ready to interact with the environment agents to complete the execution of the assigned task.

5. Performing Tasks

An OA employs a set of general task handling protocols for each task type. The OA type and task domain add an additional layer of specificity to the sequence of actions. General and specific protocols represent short term planning capabilities of an OA. Using such protocols, an OA can obtain services from other agents, distribute tasks to other object-agents, manage other agents, and run conflict handling sessions, about its DO state, that involve multiple agents.

5.1 EVALUATION TASKS

The DOs can be activated to provide various evaluations of their current state upon designer 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 latter requires a search for the DOs performance requirements (either prototypical or designer specified) and computations of the current performance values. In this sense, an OA interacts with the environment agents (e.g., designer, EAs) to obtain performance requirements (prototypical, designer specified or represented in the model, for instance, as constraints networks). The OA then interacts with EAs which assess the OA performance based on the information provided by the OA and according to any specified requirements.

5.2 MODELING WITH OBJECT-AGENTS

The following scenario assumes a designer using a bottom-up approach to generate a design model of an office building. The designer selects rooms from a pool of predefined Room_DO types (or defines a new Room_DO type). The designer 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 of the relationship between them. Three main relationship status are currently identified: *no-relation* (the default status), *constituent-of/contains*; and *associated-with.*

In a "constituent-of/contains" relationship a super_DO contains a sub_DO and the sub_DO is a constituent-of the super_DO. For example, a Wall_DO is logically a sub_DO of a Room_DO. However, it should be permissible for the same Wall_DO to be a super_DO of the same Room_DO when needed. The designer should be able to assign any type of relationship between DO types, the logic behind the relationship being dependent on the designer's views of how the DOs should be linked. There is no reason to suggest 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, this Wall_DO can still be associated with the Room_DO even if it is located in the volume of an adjacent Room_DO. On the other hand, there may not be the need to establish a relationship between a Door_DO and a BuildingFloor_DO if no agent in the environment can utilize such relations. Note that 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 at the same time. A Wall_DO can be, simultaneously, a sub_DO of two Room_DOs in which case the Wall_DO is a joint_DO.

The "associated-with" relation specifies a non-hierarchal functional or semantic link in the model. A pair of DOs may be associated by their attributes. For instance, a Wall_DO's thickness attribute may be associated with a Room_DO's thermal and acoustic attributes. The relation "associated-with" is temporarily assigned during a design session (e.g., during the execution of a task). This relation can be used to register a DO or an EA in the list of interested DOs and EAs of a DO attribute. See the "list of interest" in Aly (2000) used for conflict handling.

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 latter is performing a task to modify its proportions. The location of the interior Wall_DO may consequently be changed if the proportions of the facade are modified (even though the interior Wall_DO is not a constituent-of the Facade_DO). The DO hierarchies are specified by their relations according to the task on hand. That is, DOs should only have task dependent hierarchies.

How does the designer assign relations between DOs? Is it necessary that the environment provides the means to assist the designer in assigning relations and establishing the task depend hierarchies among the DOs? It can be argued that if a designer is to assign each single relation among the

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. Thus, the designer needs only to assign those relations that are required for the current tasks on hand. On the other hand, if the environment provides predefined DO hierarchies, a large number of unnecessary relations are produced which may need to be disabled in order to perform certain tasks. This can be a more tedious task. Thus, it is necessary that the environment adopts the designer's mental model of the design state and not force the designer to adopt a pre-defined model imposed by a set of default DO relations and hierarchies. The environment may provide support to the designer in assigning relations amongst DOs in various ways:

- Through interface-agents to provide the designer with multiple techniques of assigning relations amongst single 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).
- Through predefined DO hierarchies to provide the designer with experimental test beds.
- Through domain specific agents that are geared toward establishing hierarchies amongst the DOs of the model.

In summary, DOs do not have relationships to other DOs unless specified by the designer or other supporting agents according to designer preference. Relations between DOs are temporal, and hierarchies established between DOs are task dependent.

5.3 TASK DECOMPOSITION WITH OBJECT-AGENTS

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 decomposes the task to sub-tasks amongst the sub_OAs. The task decomposition is dependent on the relations and hierarchies established between the OA and its sub_OAs. There are two types of decompositions: *flat* and *complex*.

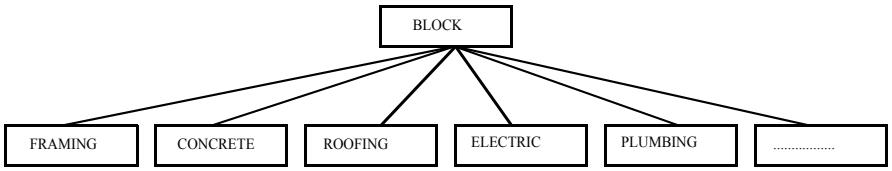
A *flat decomposition* is performed whenever the result of a task assigned to the OA is the aggregation of the results of the sub-tasks assigned to its sub_OAs, and each DO is a constituent of one DO. An example of a flat hierarchy is a cost estimate task for a building block, Figure 1-A. The material cost of a building block is the cost of all its material components represented in a hierarchy as 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 on 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, calculated using the floor area), Figure 1-B.Various flat decompositions can be established around different classifications of the same task.

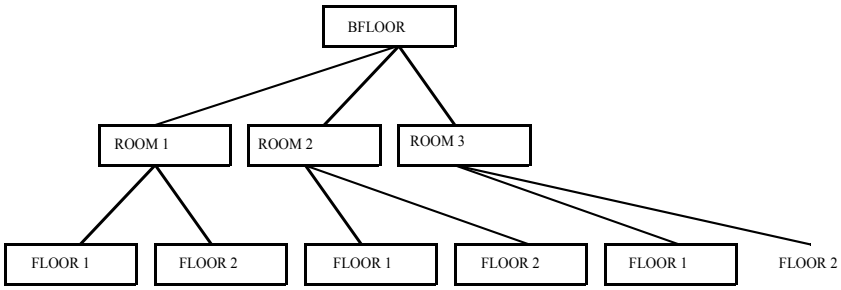
A *complex decomposition* is performed whenever the result of a task assigned to the OA is not necessarily the aggregation of the results of the sub-tasks assigned to its sub_OAs, and at least one DO is a constituent of more than one DO (a joint_DO). For example, consider framing cost estimate classified per building block. The cost of framing material of each building block is the cost of all its frame components. 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 1-C. For any classified quantity take-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 that do not depend entirely on aggregation of sub-results may require complex decompositions as well.

Accordingly, 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: DO type, task domain, task type, and task focus. It is, therefore, more appropriate to allow the designer to establish the hierarchies according to the nature of task on hand. In such case, task decomposition among sub_DOs can be a direct reflection of the established hierarchy.

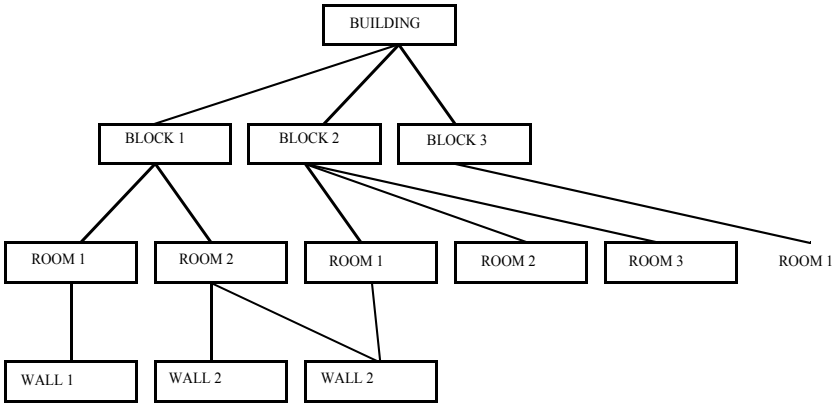
How does an OA decompose a task among it is sub_DOs? An OA's knowledge of how to handle a task on hand is embedded within its problem solving protocols. These protocols are general guidelines of either how a task can be decomposed and how sub-tasks are delegated to the sub_OAs or how a task can be directly executed by the OA when no decomposition is needed. The protocols are therefore specific to DO-type and task domain, type and focus. 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.



(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 1. Task and decomposition.

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 designer with a structural analysis based solely on the building floor own geometry and attributes. If the designer 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. In another words, the OA needs a hierarchy to apply the more sophisticated problem solving protocols. Using the information embedded in the protocols and interacting with the EA the OA may assist the designer to establish the right task dependent hierarchy.

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

When new DO-types are added to the environment a set of protocols applicable to each such type must be made available to its OA. Note that if multiple DOs of different DO-types are to be activated as a single composite_OA (e.g., a Corner_OA which may be a composition of walls floors and ceilings), a new set of protocols need to be made available to such composite_OAs. It should be noted that 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.

5.4 EVALUATING THE MODEL

When a designer adds a new DO to the environment, the DO remains in a passive state until the designer 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 then activated and a newly created Room_OA starts interacting with the appropriate agents that will assist in executing the assigned task.

The problem solving protocols of the Room_DO type (which are loaded into the Room_OA during creation) prompts it to first try to identify the daylighting requirements for this room (e.g., bedroom, reading space). 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, or finally, from the designer 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 on hand, namely the Daylighting_EA in this case). The Daylighting_EA requests information about the Room_OA, for example, about its dimensions, orientation, location with 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 reside in its original DO, or are obtained through interaction with a geometry-agent (typically, using the functionalities of the CAD system). Its geometric relations to other DOs are calculable. To obtain this information the Room_OA would assign a task to a geometry-agent to find specific information relating to its adjacencies. The geometry-agent performs the necessary calculations, and provides the results back to the room_OA which, in turn, provided to the requester.

If the lighting level is found to be below the required value, the Room_OA notifies the designer that current lighting level are below the required value, which, in turn, necessitates further modifications to the current state in order to meet the performance requirements.

5.5 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 simple 2D geometric configurations to solid complex objects with attributes, constraints and so forth. 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, and 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 task 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 room dimensions and location with respect to 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.

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 further information from a query-agent or from the designer, or may inform the designer that the information provided is inadequate and not compatible with the assigned task.

An alternative to changing the design of the EAs is to make the interface-agents (which facilitate the interactions between the EA and 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 interface-agents and 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 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.

6. Advancing a Design State with Multiple Object-Agents

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: i) the various degrees of autonomy of an agent, and the planning capabilities that each type of agent may incorporate according to the advocated design environment; and ii) presents our approach on how cooperation of agents with different capabilities can support design activities.

6.1 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, to 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 (Myers et al 1993).

Quadrel (1991) 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. When applied to design tasks, the coordination of such an organization is complex even when limited to SAs. In an OA-based design environment with a large number of agents (SAs and OAs), coordination becomes even more complex, especially if the OAs are to be fully autonomous.

Agency behavior implies that an OA, as an agent, should have the ability to self-activate when it sees fit. This requires that an OA should have the ability to interpret other agent actions and to coordinate its actions accordingly. The coordination of 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 designer. Rothman et al (1993) suggests that agents can be classified at many levels of complexity, but they can only be considered intelligent when they possess planning capabilities. An agent creates ‘plans’ based on ‘models’ of itself, and of 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 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 interactions.

We consider OAs as semi-autonomous agents. That is, these should be activated only when there is a task to be performed. Accordingly, there are no fully autonomous agents apart from the designer(s).

6.2 SHORT TERM VERSUS LONG TERM PLANNING IN DESIGN

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*. Agents follow rules to map states to actions without a long-term view of how such actions will lead to achieving goals. Durfee (1988) describes this type of planning as 'reactive planning.' He considers it important for any problem solving environment to adopt what he terms 'strategic planning' which 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 long term global goals even if a group of the local and sub-goals are modified or changed during the execution of the plan. Designers 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.

Accordingly, we suggest that the designer should be responsible for long term planning and for collective evaluation of the different states of design. 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 tasks among small entities, such as OAs, make it feasible to set an acceptance criterion for each task.

According to our approach, OAs deal with small and immediate tasks; these are more applicable to short term planning strategies.

The change of an OA status depends on the support and response of other agents in the environment. Each change in the OA status is an incremental change for the entire design state. It is up to the designer 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 designer may not be aware of the individual activities of the agents. Further, it is not intended that the designer guide each event conducted by each agent. However, it is the designer's responsibility to guide the efforts of agents toward a goal state and to force conversions when seen fit. While agents in the environment may not be aware of the designer's intentions, it is the designer who should recognize solution opportunities and orchestrate the agents to an acceptable state.

The designer'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 DOs (i.e., introducing new DOs to the CAD

environment, modifying attributes of current OAs, etc.), and by modifying design goals. More importantly, the designer is required to direct and guide the efforts of the other agents to advance the current state towards an acceptable design.

7. Details of a Daylight Evaluation Task Executed by Object-Agents

7.1 THE ASSIGNED TASK

Here, the designer is interested in evaluating daylighting of a BFloor_DO. We list the assigned task, identify the main players, major events and expected results. The details of the interactions are given in Aly (2000, pp. 58-62). The evaluation includes daylighting levels classified according to each Room_DO of the BFloor_DO during a specified range of hours of the day. The evaluation also includes daylighting levels of each individual opening within each Room_DO. The BFloor-evaluation includes statistical informat-ion about the number of Room_DOs within the BFloor_DO satisfying the daylighting constrains and prototypical values. The scenario involves three players: the designer, a BFloor_OA and a Daylighting_EA.

7.2 THE MAJOR EVENTS

There are five of which two are optional:

- The designer activates the BFloor_DO and assigns it a daylighting evaluation task, which triggers a chain of activation by the BFloor_OA down the hierarchy to its Room_DOs (the last nodes in the daylighting task dependent hierarchy).
- Interactions with the Daylighting_EA. Each activated OA runs a daylighting evaluation session regarding its own state. The sequence of evaluation sessions starts from leaf OAs, where the results are provided to their super-agents. Each super-agent aggregates the results provided by its sub_OAs, and so forth up to the designer level.
- The OAs update the information of its DOs and terminate themselves.
- (Optional) The designer triggers a conflict handling session.
- (Optional) The designer requests the implementation of new values.

7.3 THE EXPECTED RESULTS

For each evaluation session, the assigned BFloor_OA provides the designer with information that may contain at least one of the following:

- Daylighting levels of each Room_DO;
- Prototypical daylighting levels of each of the Room_DO types (e.g., living room, bedroom);

- A warning issued to each DO when the specified or prototypical daylighting levels are below requirement.
- The designer examines either new DO attribute values for DOs in the task dependent hierarchy or new daylighting constraints.
- (Optional) The designer examines new DO attribute values for DOs interested in the attribute values being evaluated.
- (Optional) Implementation of examined DO attribute values and updating of the DO relations.

8. Object-Agent Task Execution

Once an OA is activated (i.e., a class instance is created and registered with its super-agent), the super-agent assigns a task to the OA. The super-agent provides the task type, domain and focus (e.g., the value of an attribute to be examined. This sets the context for the task. The assigned task is classified by the OA and handled by appropriate algorithms (evaluate, implement etc.).

8.1 HANDLING EVALUATION TASKS

An OA handles an evaluation task according to the task domain and 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 represents a leaf_DO in a task hierarchy, 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 in interaction with the EA of the task-domain. Figure 2 shows a decomposition of a Block_DO for a cost evaluation task. The decomposition is centered about its construction categories (such as framing, roofing, etc.) instead of its spatial elements (i.e., floors, zones, rooms, etc.). This is a flat decomposition which corresponds to the example in Figure 1-A. The activated DO (e.g., Foundation_DO) uses (and activates) related DOs (e.g., Slab_DOs, Footing_DOs) during the course of evaluating its own cost.

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.

- *Block_DO total cost classified per BFloor_DO.*

As shown in Figure 3 the 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 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 and then aggregated to provide the cost of the Block_DO.

- *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 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 aggregation of sub-results.

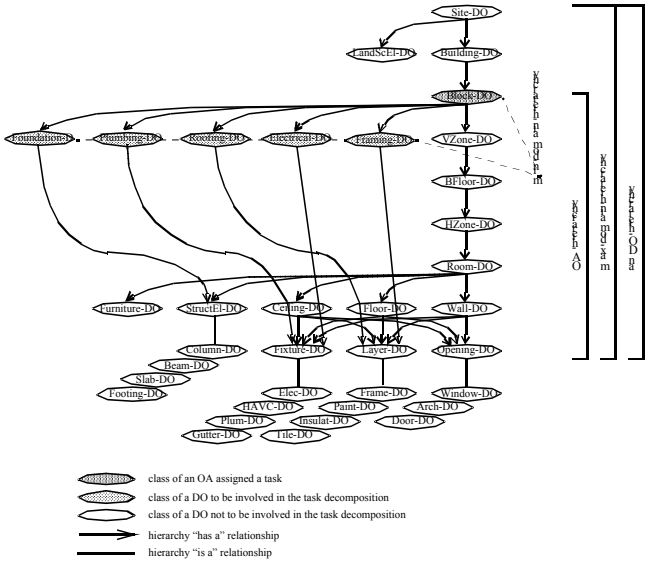


Figure 2. Decomposition of a Block-DO cost evaluation task.

- *Total cost of a Block_DO when the DO of classification is not a super_DO of the leaf_DOs.*

DO classes at the same level but in different branches of the OA hierarchy may act as super_OAs to one another. For example, in Figure 3, 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 at 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 the cost task decomposition assigned by the Block_OA to the construction OAs goes through the VZone_OA. The VZone_OA acts as a super_OA to the construction OAs even though these 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.

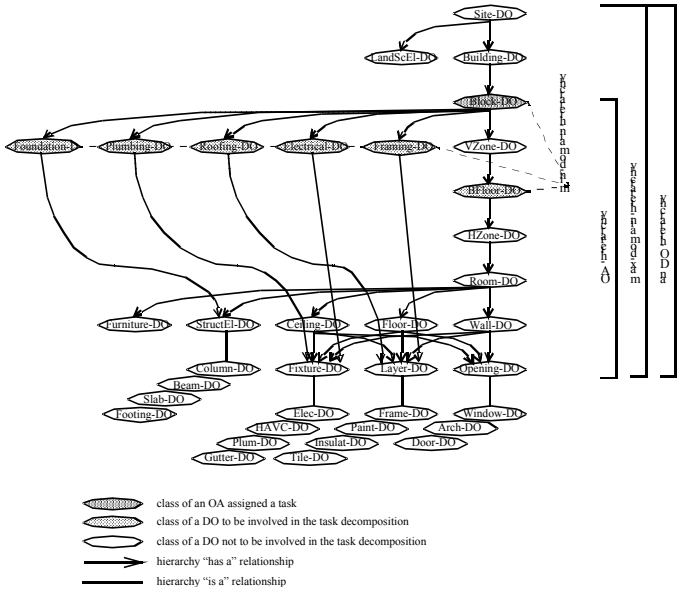


Figure 3. Decomposition of a Block-DO cost evaluation task (classified per BFloor-DO).

To summarize, evaluation tasks fall into one of three cases: (a) *task decomposition is not required*, for example, when the task is assigned to a leaf_OA in the DO-hierarchy or to an OA that hold attributes which substitute for leaf_OAs); (b) *task decomposition is required*, for example, when the task is assigned to an OA that neither is a leaf_OA nor holds any attributes which substitute for leaf_OAs); and (c) *task requires decomposition and classification of sub-results*.

8.2 MAKING THE ACTIVATION LIST

In an OA-hierarchy (i.e., all constituent DOs and 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. With respect to a task being executed by an OA, a DO that is a member of an OA-hierarchy is: i) related to the task and crucial to the execution of the task; ii) related to the task but not crucial to the execution of the task; or iii) not related to the task. Thus, an activation list of the sub_DOs participating in the task decomposition must be compiled. This list is compiled using the parameters provided by the Domain_Decomposition_ Protocol (used by the OA to execute a task). The decomposition protocol provides the DOs that belong to the first category; these DOs constitute the min-domain-hierarchy.

An *activation order* may be required for executing 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. 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:

The set of all DO classes (of the existing hierarchy) which are eligible for activation according to the task domain must be defined 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 Domain_ Decomposition_Protocol of the task on hand. In the hierarchy of Figure 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).

The DO classes that should not to be activated are compiled in a "skip$_{list}$." The making of the skip list is explained in the sequel. The "activation$_{list}$" is then compiled as the difference between the max-domain-hierarchy$_{list}$ and skip$_{list}$. The designer may elect to insert additional DO classes from the max-domain-hierarchy$_{list}$ to the activation$_{list}$. Any class added by the designer must conform to the constraints of making the skip$_{list}$.

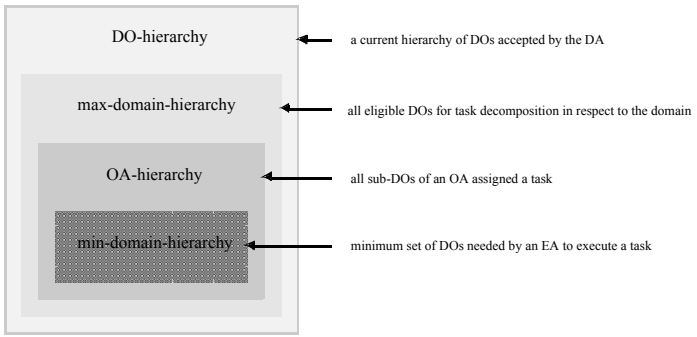


Figure 4. Relation between hierarchies (general case): min-domain-hierarchy < OA-hierarchy < max-domain-hierarchy.

We can conclude that an OA-hierarchy is a subset of a provided DO-hierarchy, typically a subset of the *max-domain-hierarchy* (the set of all eligible classes for task decomposition in respect to this particular domain).

We note that i) the min-domain-hierarchy is a subset of the max-domain-hierarchy, must be a subset in the $activation_{list}$, which is a subset of the max-domain-hierarchy; ii) 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 it is disjoint from the min-domain-hierarchy.

8.3 AGGREGATION

Any decomposition of a task into sub-tasks is counterbalanced by an aggregation of the sub-results of the execution of the sub-tasks. Similar to decomposition, aggregation is domain dependent and is necessary for the execution of a task by an OA. For instance, the cost of an OA is the aggregation of the costs of the leaf_DOs of the OA-hierarchy (provided that all components of cost are represented as leaf_DOs in the OA-hierarchy). In addition, the cost of an OA may include the cost of its sub_DO attributes (typically, such attributes substitute for sub_DOs that are not explicitly represented as leaf_DOs in the OA-hierarchy).

9. Example of Domain Protocols

The OA protocols are mainly domain dependent. We provide an example of the structural analyses protocols for decomposition, sorting and aggregation, Table 1.

TABLE 1 Example protocols

Structural Analysis Protocols		
Decomposition	*Sorting*	*Aggregation*
$skip_{list}$: Site_DO, all leaf_DO classes excluding the StructElement_DO class *min-domain-hierarchy*: [BFloor_DO, StructElement_DO] *max-domain-hierarchy*: [domain-$hierarchy_{top}$ -- domain-$hierarchy_{bottom}$] *$domain\text{-}hierarchy_{top}$*: Building_DO; *$domain\text{-}hierarchy_{bottom}$*: StructElement_DO	*decomposition order*: [Building_DO, Block_DO, VZone_DO BFloor_DO, HZone_DO, StructElement_DO] *typical evaluation order*: carried loads (top-down, higher loads are added to the lower ones, e.g., higher BFloor_DOs must be analyzed first) *special evaluation order*: suspended loads (bottom-up, lower loads are added to the higher ones, e.g., lower loads	*aggregation-type*: Site_DO □ listing (e.g., table) Building (and below) □ request service from structural_EA `

Note For structural recommendation min-domain-hierarchy does not include StructElement_DO	of a suspended bridge must be analyzed first)	

10. Conclusion

For 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 through 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 designed artifact. 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 we have surveyed employ any kind of representation where agency behavior can be considered as an aspect of design objects as well (Aly (2000) for an extensive bibliography). The engineering of a framework for design decision-making environments wherein design objects are endowed with agency behavior is the main contribution of this paper.

References

Aly, S:2000, *A Framework for Interaction and Task Decomposition for Objects Emulating Agency Behavior,* Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, PA.

Durfee, EH: 1988, *Coordination of Distributed Problem Solvers,* Kluwer Academic Publishers, Boston, MA.

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 93, California Polytechnic State University, San Luis Obispo, CA.

Pohl, J, Porczak, M, Pohl, KJ, 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.

Quadrel, RW: 1991, *Asynchronous Design Environments: Architecture and Behavior,* Ph.D. Thesis, Department of Architecture, Carnegie Mellon University, Pittsburgh, PA.

Rothman, P and Coull, T: 1993, Virtual reality for decision support systems, *AI Expert* **Aug**.