Design Agents in 3D Virtual Worlds

Mary Lou Maher, Gregory J Smith and John S Gero
Key Centre of Design Computing and Cognition
University of Sydney
mary@arch.usyd.edu.au, g_smith@arch.usyd.edu.au, john@arch.usyd.edu.au

Abstract

Design agents are rational agents that monitor and modify elements of a designed environment. Special characteristics of design agents include the ability to reason about patterns and concepts, and the ability to act autonomously in modifying or changing the design to achieve their own goals. 3D Virtual Worlds are multi-user distributed systems that provide a designed environment and a closed world environment for studying design agents in a multiagent system. We present a model for a design agent reasoning process and a model for constructing a memory of the agent's knowledge and interaction with a virtual world. The reasoning process includes sensation, perception, conception, hypothesizing, and planning a sequence of actions. Each agent has a constructed memory: a dynamic and changing view of the designed world that is determined by the agents sense data and reasoning. The agents construct and maintain a representation of the relevant objects in the world using a Function-Behavior-Structure formalism in order to reason about the intended and actual functions of the designed objects in the world. We have implemented this agent model by extending the Active Worlds platform so that each object in the 3D world can have agency. We illustrate the model with a door agent and a multi-agent room that reason about the use of the 3D world.

1 Agent-Based Virtual Worlds

3D Virtual Worlds are multi-user distributed systems that provide a closed world environment for studying design agents in a multiagent system. An agent is an encapsulated system that is "situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives" [Jennings, 2000]. Our agents monitor and modify elements of their own environment. The 3D virtual worlds that we have been working with allow multiple users to connect to a server through a client, where the processing is distributed between the server and the client. We use a virtual world platform called Active Worlds [1] (AW) as the software environment for developing our agent societies. In Active Worlds, the users appear as avatars and are able to interact with and change the objects in the world. This environment provides an unbounded closed environment for studying agent reasoning, interaction, and communication. Figure 1 shows a virtual conference room that was designed and used for meetings and seminars. The objects in this room exhibit interactive behaviors preprogrammed in the Active Worlds platform, but they do not have agency. Our research starts with this platform as a basis for developing agent-based virtual worlds.

With the exception of human controlled avatars, each object in a virtual world has both a 3D model that supports the visualisation of the object's function in the world, and is a software object that can have agency to support autonomous behaviour. When each 3D object has agency, the virtual world is composed of design agents in the sense that the world is designed and each object in the world can sense the world and respond by adapting the designed world.

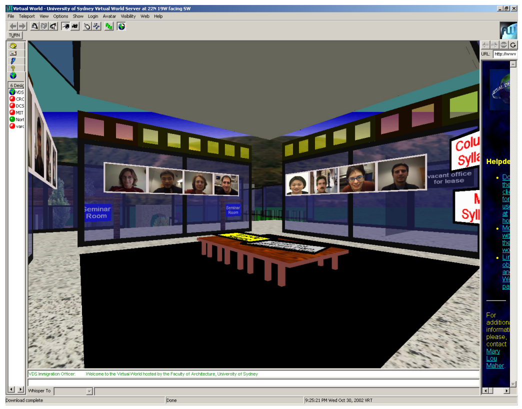
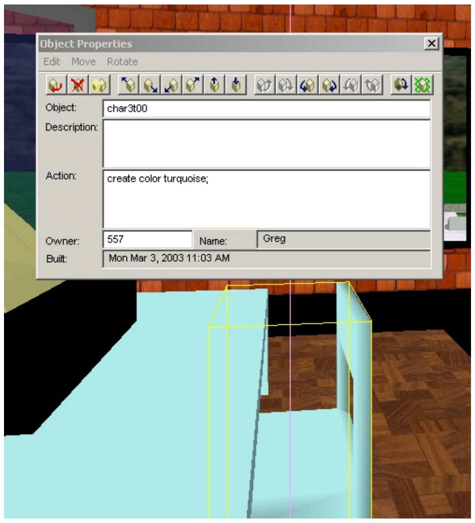


Figure 1: A 3D model of a virtual seminar room.

New 3D objects are added to an AW world by copying an existing object, moving it as required, and editing a dialog box to configure it, as shown in Figure 2. The dialog box allows the world builder to specify a 3D model and a script that describes the behaviour of the object. The 3D models can be

[1] http://www.activeworlds.com

taken from a standard library provided by Active Worlds, or can be generated in a 3D modelling package and added to the library. The behaviour of an object is limited to the preprogrammed behaviours allowed by the scripting language. By making the world agent-based, a person designing the world is able to select or generate an agent model in a way that is similar to selecting or generating a 3D model for the object.



Object: char3t00
Description:
Action: create color turquoise;
Owner: 557 Name: Greg
Built: Mon Mar 3, 2003 11:03 AM

Figure 2: Inserting an object into an AW world.

Consider, by way of example, a virtual conference zone that includes a zone agent, a door agent, and a set of wall agents. A zone is agency applied to a space. The function of the zone is to comfortably contain the avatars of those citizens present, to only allow recognised citizens into the zone, and to manage chat within the zone. The zone has no 3D representation in the world, and so must negotiate with its walls so as to achieve its goals. The crucial point to understand here is that agency is associated with the objects that comprise the design of the places in the world: the majority of agents within the world will not be represented with an avatar.

A zone agent is able to sense the number of citizen avatars within it, sense chat by those citizens, and sense where its walls are. Assume for this example that the conference zone contains a maximum 20 citizens before citizen "Greg" enters. The zone's sensor constructs sense-data corresponding to "Greg is in the zone". The zone interprets this as function "zone is to comfortably contain the avatars of those citizens present" not being satisfied. The result of this is a goal to change the zone so as to satisfy function: it wants the zone to get larger. As this is a multi-agent society, it negotiates with its walls to change the size of the zone. The zone also recognises and teleports illegal citizens, traps chat and controls how chat is distributed amongst citizens in the zone, and negotiates to make the walls secure when no meeting is occurring.

Although the focus of this paper is on agent reasoning and agent memory, we intend that these agents communicate and collaborate. In this application, we define a multi-agent system to be an aggregation of agents that share some ontological connection, such as a zone agent plus a set of wall agents that collectively comprise a virtual conference room. The multi-agent system facilitates inter-agent communication and manages resources on behalf of the agents. This includes computational resources, such as a thread pool and a connection to a virtual world.

This paper presents a model for a design agent reasoning process and a model for constructing a memory of the agent's knowledge and interaction with the world and other agents. By establishing a common model for reasoning and agent memory, the world can be designed using a consistent representation and therefore expectation of what it means for the objects in the world to have agency.

2 Reasoning Process Model for Design Agents

We have developed a reasoning process model for design agents that is independent of the type of design or environment in which the agent operates. This model, derived from [Maher and Gero, 2002; Gero and Fujii, 1999; Smith and Gero, 2002], allows the agent to reason about the world through sensation, perception, and conception, and to reasoning about its own behaviour by hypothesising and planning a set of actions. Our agent model comprises sensors, effectors and four reasoning processes: perception, conception, hypothesiser, and action activation. This model is illustrated in Figure 3, showing how the processes interact with each other and the sensor and effectors.

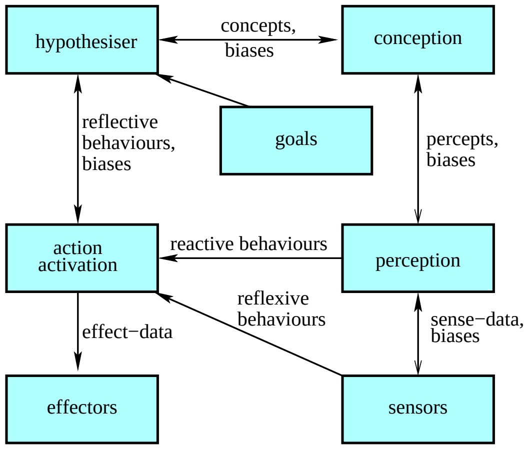


Figure 3: Design Agent Model: each agent has sensors, perceptors, conceptors, a hypothesiser, actions and effectors.

The agent is able to sense and have an effect on the virtual world through its sensors and effectors. Perception interprets sense-data and recognises patterns in the data. Conception associates meaning with percepts or patterns. The hypothesiser monitors the percepts and concepts, and identifies and selects goals that are associated with the agent's view of itself in the world. The action activator reasons about the steps to achieve a goal and triggers the effectors to make changes to the environment. Triggers may be directly off sense-data and percepts, or may be the implementation of partial plans.

In our agent model there are three levels of reasoning:

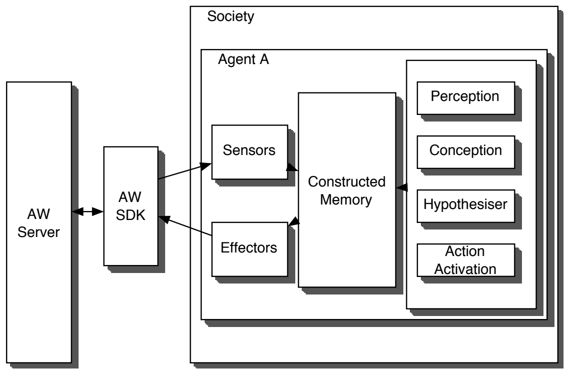


Figure 4: The Agent Model: sensors and effectors are implemented as Java Beans; the constructed memory and agents processes are implemented in Jess.

- Reflexive: where sense-data being placed in the agent's memory from the sensors triggers action activation. Reflexive behaviours are those that do not involve beliefs.
- Reactive: where percepts in the agent's memory triggers action activation. Reactive behaviours are those that do not involve intentions.
- Reflective: where the agent reasons about concepts and alternative goals before action activation is triggered. Reflective behaviours are those that involve intentions and desires.

These levels of reasoning allow the agent to act at different levels of knowledge about the world. The reflexive level of reasoning is similar to the pre-programmed behaviours that the Active Worlds scripting language allows. The reactive and reflective levels of reasoning allow the agent to reason about its understanding of itself in the world before acting.

The Active Worlds platform allows the behaviour of the world to be extended using a software development kit (SDK). This SDK can be used to program bots that can enter the world as software controlled avatars and interact with human controlled avatars and objects. We are using the SDK to implement our agent model. Using the SDK directly means writing a C/C++ program to interface to a dynamic link library. Sensors and effectors are implemented in Java using a Java Native Interface. This encapsulates the SDK within a set of standard configurable components. The agent reasoning is implemented using the rule-based language Jess [2]. The rules in the rule-base are grouped into modules according to perception, conception, hypothesiser, and action activation. The fact base is effectively a constructed memory for each agent, which is modified externally by input from the sensors and internally by the rules. This implementation configuration is illustrated in Figure 4.

3 Memory Model for Design Agents

Our agents make use of the Function-Behaviour-Structure (FBS) formalism [Gero, 1990]. In this formalism, design objects are represented by three sets of descriptors: function F, behaviour B, and structure S. Function is the teleology ascribed to an object in the world. It is an ascription only, although for humans the relationship between structure and function is learned so that it appears there is a direct connection between them. Consider as an example a door agent, shown in Figure 5. The functions of a door include to allow access, to restrict access, and to provide security.

[2]http://herzberg.ca.sandia.gov/jess/

Figure 5: A door in the world to which agency has been applied.

Structure are the elements and their relationships that go to make up a designed object. It is what is synthesised in a design process, and it is what can be sensed from the world by an agent. Structure of a door includes its location, orientation, name, and dimensions.

The third component of the FBS formalism is behaviour. Behaviour in FBS are attributes that are either derivable from structure or are expected for a required function. Behaviour is inferred from changing structure; it is an interpretation of those changes by the agent. The FBS notion of behaviour is a design computation one; it is closer to the notion from qualitative reasoning than from psychology. Examples of behaviour for the door are `OPEN` and `CLOSED`. These alternative states that a door can be in are implemented by changing the structure of the door in the world. Further, these behaviours can be implemented in different ways. The agent may choose to implement the `OPEN` behaviour by removing the door, by making it transparent, or by moving it up or to the side.

An Active Worlds world will contain a set of 3D objects where some 3D objects have no agency, some 3D objects are added to the world by an agent, and some 3D objects are recognised by an agent and thereafter can be changed by the agent. Furthermore, some agents will correspond to a single 3D object, whereas some will correspond to a set of 3D objects.Our agents react and reason about objects in the world. Conceptually they *are* objects in the world, only imbued with an agency that they would not otherwise have. Each agent maintains a representation of itself as an object or objects in the world, using FBS to categorise the attributes of itself as an object. Each agent also constructs an FBS model of the other objects in the world that are relevant, as described in [Gero and Kannengiesser, 2003]. In addition, an agent constructs a representation of the avatars in the world, its goals, and its plans as a sequence of actions. Structure will therefore comprise the identification and location of a set of 3D objects, sensed and perceived relations between 3D objects and/or the world, plus other sensed data such as uninterpreted messages from other agents.

We demonstrate our design agent model with an example door agent, shown in Figure 6(a). The door is initially a single agent that recognises the 3D representation of itself in an AW world and then maintains itself according to its function.

The agent package currently includes sensors for chat text, avatars, 3D objects (within a configured region of the agent), ACL messages, and virtual reality time. Sense-data are all java beans asserted into Jess working memory. Sensors and effectors are configured from XML, which for the door is as follows:

```
<reteagent name="door">
<parameter name="jess" value="file:newdoor.clp"/>
<parameter name="owner" value="Greg"/>
<parameter name="friendlist" value="Greg Mary"/>
<parameter name="officeXmin" value="-1910"/>
<parameter name="officeXmax" value="-580"/>
<parameter name="officeZmin" value="-5810"/>
<parameter name="officeZmax" value="-4160"/>
<parameter name="radius" value="300"/>
<location x="-1920" y="-250" z="-4065"/>
<behaviours codebase="file:">
<sensor class="kcdcc.awa.base.AW3DObjectSensor">
   <parameter name="width" value="500"/>
   <parameter name="height" value="500"/>
</sensor>
<sensor class="kcdcc.awa.base.AWAvatarSensor"/>
<sensor class="kcdcc.awa.base.AWChatSensor"/>
<effector class="kcdcc.awa.base.AW3DObjectEffector"/>
<effector class="kcdcc.awa.base.AWChatEffector"/>
</behaviours>
</reteagent>
```

The sensor `kcdcc.awa.base.AW3DObjectSensor`, for example, senses 3D objects from the virtual world. For example, when a 3D object is clicked on from an AW browser by a citizen, a java bean is asserted in working memory that looks from Jess like the following:

```
(kcdcc_awa_base_ClickedObject3DSenseData
   (objectNo ?no)
   (locn ?location)
   (OBJECT ?obj)
   (slot class))
```

Similarly, there are effectors for chat, 3D objects, email, URLs, and avatar movement/teleportation.

A visual scene, as an external context for a design agent, is too complex to be used as pre-supplied percepts. So we distinguish sense-data from percepts. This is an example of a situation where a coupled dynamical systems model of agency [Beer, 1995] applies, and it applies generally for embodiment in a complex environment. Sense-data are uninterpreted data collected by a sensor, and percepts are interpreted patterns in that sense-data. Sense-data provide structure, S, as behaviour is derived and function is ascribed. Sense-data is used to construct the agent's structural representation of objects and people (avatars, citizens) in the world.

In agent memory there is an object corresponding to each object of the 3D world that is of relevance to the agent. These objects have properties that are categorized according to FBS. Perception rules have raw sense data on the LHS and then modify agent memories of structure on the RHS. Perception reasons about patterns that the agent finds that produce a representation of behaviours of the objects in the world. Behaviour, therefore, is an interpretation by the agent of changing structure. The agent cannot sense behaviour, it must construct it.

For example, for the door, perception determines structure to recognise the door 3D object and recognise the presence of citizens. The 3D object of the door must be sensed as the citizen that owns the door can move it indendepently of the agent (using an AW browser). The structure of the door is remembered as a Jess fact:

```
(deftemplate STRUCTURE::door
   (slot object-id (type INTEGER) (default 0))
   (slot location (type OBJECT) (default  nil))
   (slot ownerid (type INTEGER) (default  0))
   (slot owner  (type STRING))
   (slot radius (type INTEGER) (default 200))
   (slot model (type STRING) (default "pp16w3.rwx"))
   (slot description (type STRING)
                     (default "Intelligent Door"))
   (slot action (type STRING)
                (default "create color white")))
```

The behaviour of the door is similarly remembered as a Jess fact:

```
(deftemplate BEHAVIOUR::door
   "Behaviour of a door object"
   (slot state (type LEXEME) (default NONE))
   (slot object-id (type INTEGER) (default 0))
   (multislot friendlist))
```

State is an interpretation on the part of the agent. Structures may implement `(BEHAVIOUR::door (state OPEN))` by translating the door object, by making it transparent, or by removing it completely. But the agent cannot determine door state without reasoning on its part.

Beliefs of the agent are modular. So, for example, perceptor and conceptor rules are in module `INTERPRETATION`. When the avatar for an already recognised citizen moves, the following rule fires:

```
(defrule INTERPRETATION::person-interpretation-2
   "Locate a known person"
   ?f <- (MAIN::kcdcc_awa_base_LocatedAvatarSenseData
            (name ?n)
            (locn ?l)
            (session ?s))
   ?ps <- (STRUCTURE::person (name ?n))
   ?pb <- (BEHAVIOUR::person (name ?n))
=>
   (modify ?ps (location ?l))
   (modify ?pb (area NONE))
   (retract ?f))
```

The inference engine is capable of forward and backward chaining but as it is built from java components, any java-implemented reasoning can be used. The focus stack of the inference engine restricts rules to firing from one module at a time. Using the focus stack in this way isolates the control mechanism from any knowledge: all rules on the agenda from the current focus module run, then all rules on the agenda from the next scheduled module run, and so on until the end of the schedule. For this particular example the schedule is `MAIN, INTERPRETATION, HYPOTHESISER, ACTION`.

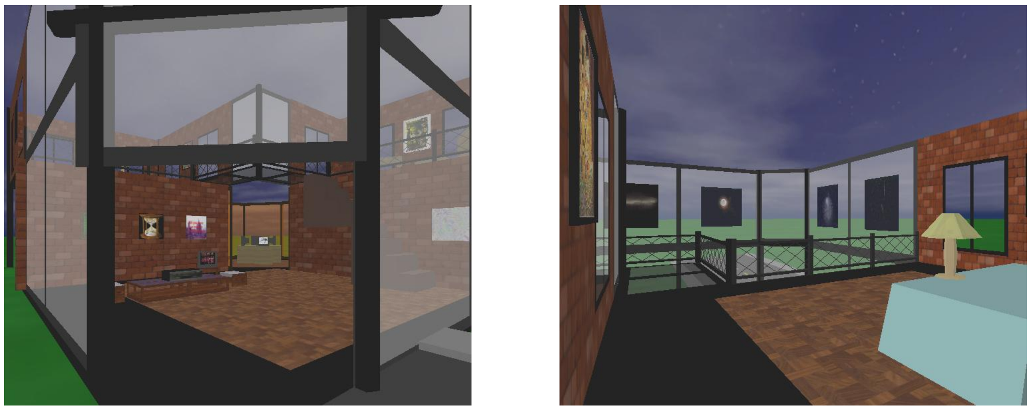


Figure 6: Our virtual office: (a) view from outside, with the door agent opening the door, and (b) the view from upstairs.

Conception rules associate meaningful concepts with the objects in the world that are relevant to the agent. A concept can be a behavior or a function of an object. For example, a person is categorised according to where the avatar is located in the world and whether that avatar is recognised as a friend. The following rule, then, alters the inferred behaviour when an avatar is recognised as being within the radius of the doorway:

```
(defrule INTERPRETATION::classify-person-1
   (declare (salience 10))
   ?pb <- (BEHAVIOUR::person (name ?n) (area NONE))
   ?ps <- (STRUCTURE::person (name ?n) (location ?pl))
   (STRUCTURE::door (radius ?rad)
      (location ?dl&:(neq ?dl nil)
                   &:(< (?pl distance ?dl) ?rad)))
=>
   (modify ?pb (area DOORWAY)))
```

This inferred behaviour is one of the agents beliefs about the world at present. These behaviours are derived from structure. The agent may also hypothesise about what behaviours are expected given its function. In either case the behaviours are the agents beliefs about the world; they are inferences, they are not sensed from the world. An agent can, however, communicate behaviour to another agent providing they share an ontology. Currently agents in our multi-agent systems share an ontology only in that they are encoded similarly. They could, but do not as yet, communicate ontology.

A goal of a design agent is to transform an intended function into a plan for changing the structure of the world. Generally no direct transformation from function to structure exists [Gero, 1990]. So instead, the hypothesiser transforms a selected function into expectations of behaviour. The action activator then changes the structure of the world so as to try and satisfy those expectations of behaviour. As an example, the following rules select a function `ALLOW-ACCESS` whenever there is a friend in the doorway and there is not also a non-friend. It then asserts a goal to satisfy that selected function.

```
(defrule HYPOTHESISER::evaluate-rule-1
   ?f <- (FUNCTION::door (selected ~ALLOW-ACCESS)
                         (object-id ?id&:(neq ?id 0)))
   (BEHAVIOUR::person (area DOORWAY) (state FRIEND))
   (not (BEHAVIOUR::person (area DOORWAY) (state ~FRIEND)))
=>
   (modify ?f (selected ALLOW-ACCESS)))

(defrule HYPOTHESISER::synthesise-rule-2
   (FUNCTION::door (selected ALLOW-ACCESS)
                   (object-id ?id&:(neq ?id 0)))
   ?f <- (BEHAVIOUR::door (state ~OPEN))
   (not (MAIN::goal (label OPEN)))
=>
   (assert (MAIN::goal (label OPEN))))
```

An agent modifies the structure of a world by adding, deleting or changing the 3D objects that comprise that world. Door action rules encode a partial plan that allows the door to operate. Each action involves one or more effector activations, and each effector is a java bean. For example, to open the door a 3D object effector is used to change the state of the 3D door object, and the state of the door is updated accordingly:

```
(defrule ACTION::open-door-1
   (MAIN::kcdcc_awa_base_ReteAgent (OBJECT ?a))
   ?d <- (BEHAVIOUR::door
           (object-id ?id&:(neq ?id 0))
           (state ~OPEN))
   ?ds <- (STRUCTURE::door (object-id ?id))
   ?df <- (FUNCTION::door (object-id ?id))
   ?f <- (MAIN::goal (label OPEN))
   ?eff <- (MAIN::kcdcc_awa_base_AW3DObjectEffector
             (OBJECT ?reference)
             (objNo ?id&:(neq ?id 0)))
=>
   (modify ?eff
      (command
         (get-member kcdcc.awa.base.AW3DObjectEffector
                     CHANGE))
      (action "create solid off, visible off"))
   (bind ?no (?reference activate))
   (retract ?f)
   (modify ?d (state OPEN)
              (object-id (?reference getObjNo)))
   (modify ?ds (object-id (?reference getObjNo)))
   (modify ?df (object-id (?reference getObjNo))))
```

The agent, then, uses the processes of Figure 3. These processes operate on facts in working memory that represent the agent's beliefs of the world using the FBS formalism. This has been illustrated in Figure 7 by overlaying FBS transformations on the model of Figure 3. Expectations of behaviour are derived from function by the hypothesiser. The action activator then uses the behaviours and knowledge of structure to generate intentions that, if realised, it believes will satisfy the required function. So goals and concepts are used by the hypothesiser to generate a partial plan that the action activator enacts.

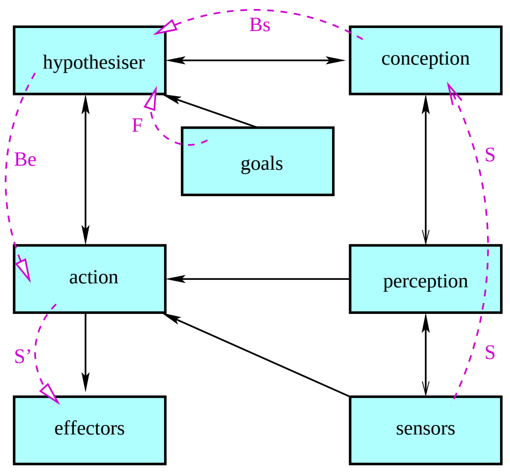


Figure 7: Design synthesis overlayed on Figure 3. To distinguish between current structure from the world and effects to be applied to the world, we denote structure to be effected by S'. Dotted arrows denote FBS transformations.

Some of the tasks of the agent are simple reflexive or reactive ones. For such tasks a sense-data driven, forward chaining of encoded partial plans is sufficient. On the other hand, many of the reflective tasks of agents that we desire of our multi-agent systems are design ones. Such agents may hypothesise the behaviours that are necessary to satisfy its functions, evaluate those expected behaviours against current actual behaviour, and thus desire changes to the structure of the world. Desiring and acting to bring about changes to the world are design acts that, in our view, are best handled with a design formalism like FBS. Such an approach also allows us to build on prior work done in the field of design computation. Our model has been developed to allow such reflective design reasoning to occur without preventing real time reflexive and reactive behaviour.

4 Communication among Design Agents

Although the agents both reason and communicate in FBS terms, the structure of a communication about about an object is not the same as the structure of an object. So prior to considering a multi-agent example we we need to consider what our agents communicate to each other.

Any multi-agent system must address the issue of communication. At a basic level we have indirect communication since each agent can sense and effect the 3D world server data. Indirect communication does not allow the agents to collaborate with each other, but only be aware of the actions of other agents through sensing the changes in the environment. Direct communication would allow inter-agent communication and collaboration. In developing a model for inter-agent communication, we need to address questions related to the content and intention of the communication. For example, should a room say to a wall "I would like an agent to change so that I am larger", or "I would like a specific agent to move such that the position of object X which is now outside becomes inside", or should it say to a particular wall "I would like you to move in direction D"? A crucial difference between an agent and an object is that an object encapsulates state and behaviour realization, but not behaviour activation or action choice [Jennings, 2000]. So a room agent that dictated to a wall agent what it should do would be a room agent with an aggregation of wall objects. It would not be a room agent collaborating with a wall agent. In our inter-agent communication, agents should communicate their intentions to one another and allow for independent choice of action.

Communication requires a common language, ontology and protocol. The language should be powerful, expressive, not unnecessarily constrain agent communications, and be sensible to all agents. The theory of speech acts [Cohen and Perrault, 1979] is the conceptual basis of agent communication languages (ACL). The ACL from the Foundation for Intelligent Physical Agents (FIPA) and KQML [Huhns and Stephens, 1999] are probably the best known. They are similar, with the FIPA ACL having a well defined semantics [Wooldridge, 2002].

In our multi-agent system we have encapsulated the FIPA ACL within sensors and effectors. Agents that communicate directly use XML and our own simple XML grammar. We adopt the FIPA ACL abstract message structure but not any of the existing implementations because our agents are deliberately not distributed. Indeed, our intention is to design the agent framework such that they can eventually be part of the AW server. XML allows agent developers to extend sensor or effector classes with their own ACL if they so desire, and the text basis of XML minimises functional coupling between sensors and effectors of different agents.

The FIPA ACL describes a set of performatives that constitute the communicative acts [FIPA, 2001]. Examples include the Call-for-Proposal, Inform, Propose, and Request. Each FIPA communicative act should be implemented in accordance with the semantic preconditions defined in terms of belief and intentions. We intend to use this FIPA ACL semantics [FIPA, 2001] only descriptively as a guide for agent designers, not formally.

We do not explicitly represent the ontology an agent uses when communicating. Each XML message contains a public identifier for an associated document type descriptor. This public identifier is taken as identifying the ontology. Agents are encoded to look for particular public identifiers. They assume that they understand any such message. The content of the messages will be reified predicates in order that they have a standard form that other agents can query. Such uninterpreted, reified content that is transmitted is structure and interpreted content is behaviour. These interpreted behaviours are what the receiving agent believes that the sending agent believes.

5 Design of a Multi-Agent Office

Our office is a multi-agent system that includes agents for the zone, walls, a door, a listener, a concierge, and a novelty implementation of the classic "Eliza" chatterbot. The office as it appears in the virtual world is shown in Figure 6. The function of this multi-agent system is to provide an intelligent,

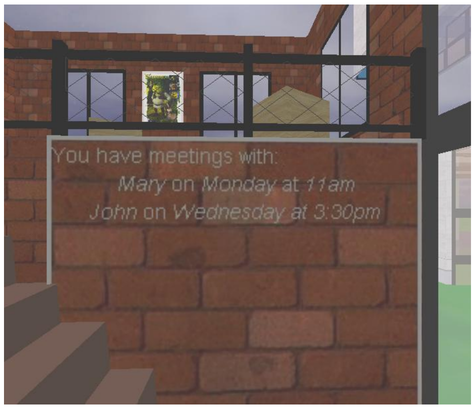


Figure 8: $ProvideInformationDisplay$ realised using wall graffiti.

virtual meeting place. In this section we shall describe the design of the zone agent and describe functions provided by the other agents.

Function is the result of behaviour from the viewpoint of the agent, so each agent desires that its 3D objects achieve their function. If we ascribe function using the terms **ToMake**, **ToMaintain**, **ToPrevent**, **ToControl** [Chandrasekaran, 1994] then the function of a wall in the multi-agent office is threefold:

Wall 1. **Function** $EnsurePrivacy$
ToControl visibility of the room during meetings by adjusting the transparency of the wall and included objects such as windows

Wall 2. **Function** $ProvideVisualBoundary$
ToMaintain an appropriate visual boundary by changing design and appearance of the wall and included objects

Wall 3. **Function** $ProvideInformationDisplay$
ToMake display on wall as graffiti of requested data

Consider function $ProvideInformationDisplay$. A wall in normal circumstances is just a 3D object with a texture mapped onto it. In a virtual world, though, that need not be the case. There is no reason that the texture should not change, and if that wall is an agent the the texture could change according to its goals. So Figure 8 shows a realisation of $ProvideInformationDisplay$ using HTML graffiti. Another agent sends a message to an available wall agent with a request to display meeting slides, minutes of meetings, schedules and so as HTML. The wall agent formats the HTML to a raster image, and changes the wall texture to that image.

What we conceive of as a room is not a 3D object but is a zone. A zone represents agency applied to the space bounded by the office walls, door, and so on. As it is a space it is not represented in the world by a 3D object. It is, though, the entity with which avatars act. The function of a zone is also threefold:

Zone 1. **Function** $MaintainSpace$
ToMake zone space and ambience

Zone 2. **Function** $MaintainSecurity$
ToMaintain security appropriate to current situation

Zone 3. **Function** $ManageMeeting$
ToMaintain participant seating and chat

The functions of the concierge are to answer questions, schedule meetings, and teleport citizens to selected locations. The functions of the listener are to record chat during meetings, restrict who can chat during a meeting to those scheduled to attend, and telegram those recordings as meeting minutes. The functions of the door are to ensure privacy during meetings, and to control access at other times.

Behaviour and structure are represented explicitly by properties constructed during perception, conception and the hypothesiser. They are also represented implicitly by relationships encoded in the rules of these processes. For zone the following are behaviour facts:

```
(deftemplate meeting
   (slot securityState)
   (multislot participants)
   (slot zoneSpace)
   (slot doorSpace)
)
(deftemplate seating
   (slot occupant)
   (multislot neighbourSeats)
)
(deftemplate properties
   "meta-facts of which properties are F, B and S"
   (slot fbs)
   (multislot property)
)
```

Actual behaviour B_s for zone includes how participants are currently seated or not seated. Expected behaviour B_e includes constraints that each participant is currently seated and how they can be seated. All zone sense-data plus the following facts are zone structure:

```
(deftemplate component
   (slot category) ;DOOR,WALL,...
   (slot objectNo)
   (slot model)
   (slot location)
   (slot ownerNo)
   (slot owner)
   (slot timestamp)
   (slot description)
   (slot action)
)

(deftemplate participant
   (slot name)
   (slot session)
   (slot locn)
   (slot citizenNo)
   (slot category) ;from avatar type
                   ;and meeting partipants
   (slot dontLike)
   (slot seat)
)
```

As the current zone space (the 3D region of the world bounded by the zone's component 3D objects) and the current doorway space are dependent variables, they are behaviour not structure.

Perception for zone provides higher level, interpreted structure S. It employs rules to recognise components of this zone (walls, doors and seats), recognise citizens, recognise seats, recognise which seats are neighbours of which other seats, and to interpret inter-agent ACL messages.

Conception for zone performs an FBS analysis transformation $S \rightarrow B_s$. It employs rules to derive current actual behaviour for the zone and door spaces, security state and meeting participants, classification of citizens (participant/not-participant, male/female, doorway/outside/inside), the set of current seats that the seat for a newly arrived citizen is nil, and that set of seats categorised as vacant.

The hypothesiser reasons over zone function by performing an FBS formulation transformation $F \rightarrow B_e$. It recognises currently satisfied functions F, decides on function changes, and which functions to select. The hypothesiser then constructs expected behaviours B_e that include that the set of possible seats for an already seated participant is restricted to their current seat, that the set of possible seats for an unseated participant includes all vacant seats, and a set of constraints. For example, each citizen is classified; one constraint is that two citizens seated next to each other should not be of the same category.

A zone agent includes both reactive and reflective behaviours. Synthesis of a seating plan is reflective and currently uses a constraint satisfaction algorithm. We are investigating an alternative, self-organising method. Other zone actions, such as the door trigger, are reactive. A partial UML activity diagram for the zone is shown in Figure 9. This diagram shows the reasoning process of the agent from sensor data, for example, an avatar enters the zone, to the actions of the zone agent, for example, teleport the avatar from the zone.

The reflective rules synthesise a plan to seat meeting participants. If an action fails to seat a participant and there is at least one free seat, the expected behaviours are reset such that all seats are deemed vacant. If it still fails and there is at least one free seat, the constraints are relaxed. If it still fails then zone will return to the original expected behaviours but with the additional expectation of added seating.

If adding more seating results in crowding, then the zone will communicate with the walls to asked for more room. Functionality required of the room is thus distributed amongst the agents. Some functions are achievable by agents in isolation, others require cooperation. The reason for this is both to simplify the design of the agents and to make them component based. Just as a 3D model of a table can be inserted into a world with minimal influence on objects already in the world, using a multi-agent approach means that we can add or remove agency for objects as we wish. Additionally, for agents which design and alter their world this means that design knowledge of particular virtual artifacts is restricted to agents of that type.

6 Conclusion

In this paper we described a design agent model as the basis for all objects in a virtual world. The agent model serves as a formalism for the intelligent behaviour of the objects in the world in a similar manner to the 3D model as a visualisation of the objects in the world. Our agent model for a virtual world is based on the design agent model introduced in [Maher and Gero, 2002; Gero and Fujii, 1999; Smith and Gero, 2002]. The virtual world provides an opportunity to study the design agent model and communication within a multi-agent system using a multi-user, interactive place that supports professional and educational activities.

We demonstrate the agent model with an example of a virtual conference room in which each of the building blocks of the conference room have both a 3D model and an agent model. The agents are able to reason about the avatars within the zone space of the room and are able to reflect on their own behaviors in response to the activities and needs of the avatars. The examples included automatically changing the behavior of the door by opening and closing it depending on the specific person that is operating a nearby avatar, and by changing the function of a wall in the room depending on the needs of the avatars within the room. These examples demonstrate the use of the FBS formalism in reasoning about a designed virtual world, and in the use of three kinds of agent reasoning: reflexive, reactive, and reflective.

7 Acknowledgements

This work is supported by the Australian Research Council and Active Worlds, Inc.

References

[Beer, 1995] Randall D Beer. A dynamical systems perspective on agent-environment interaction. *Artificial Intelligence*, 72:173–215, 1995.

[Chandrasekaran, 1994] B. Chandrasekaran. Functional representation and causal processes. *Advances in Computers*, 38:73–143, 1994.

[Cohen and Perrault, 1979] P R Cohen and C R Perrault. Elements of a plan based theory of speech acts. *Cognitive Science*, 3(3):177–212, 1979.

[FIPA, 2001] FIPA. FIPA content language library specification communicative act library specification. Document number XC00037H, http://www.fipa.org, 2001.

[Gero and Fujii, 1999] John S. Gero and Haruyuki Fujii. A computational framework for concept formation for a situated design agent. In K. Hori and L. Candy, editors, *Proc Second International Workshop on Strategic Knowledge and Concept Formation*, pages 59–71. Iwate Prefectural University, Iwate, Japan, 1999.

[Gero and Kannengiesser, 2003] John S Gero and Udo Kannengiesser. A function-behaviour-structure view of social situated agents. In *Proceedings of CAADRIA03*. CAADRIA, 2003.

[Gero, 1990] John S Gero. Design prototypes: A knowledge representation schema for design. *AI Magazine*, 11(4):26–36, 1990.

[Huhns and Stephens, 1999] Michael B Huhns and L M Stephens. Multiagent systems and societies of agents. In G. Weiss, editor, *Multiagent Systems: A Modern Approach*

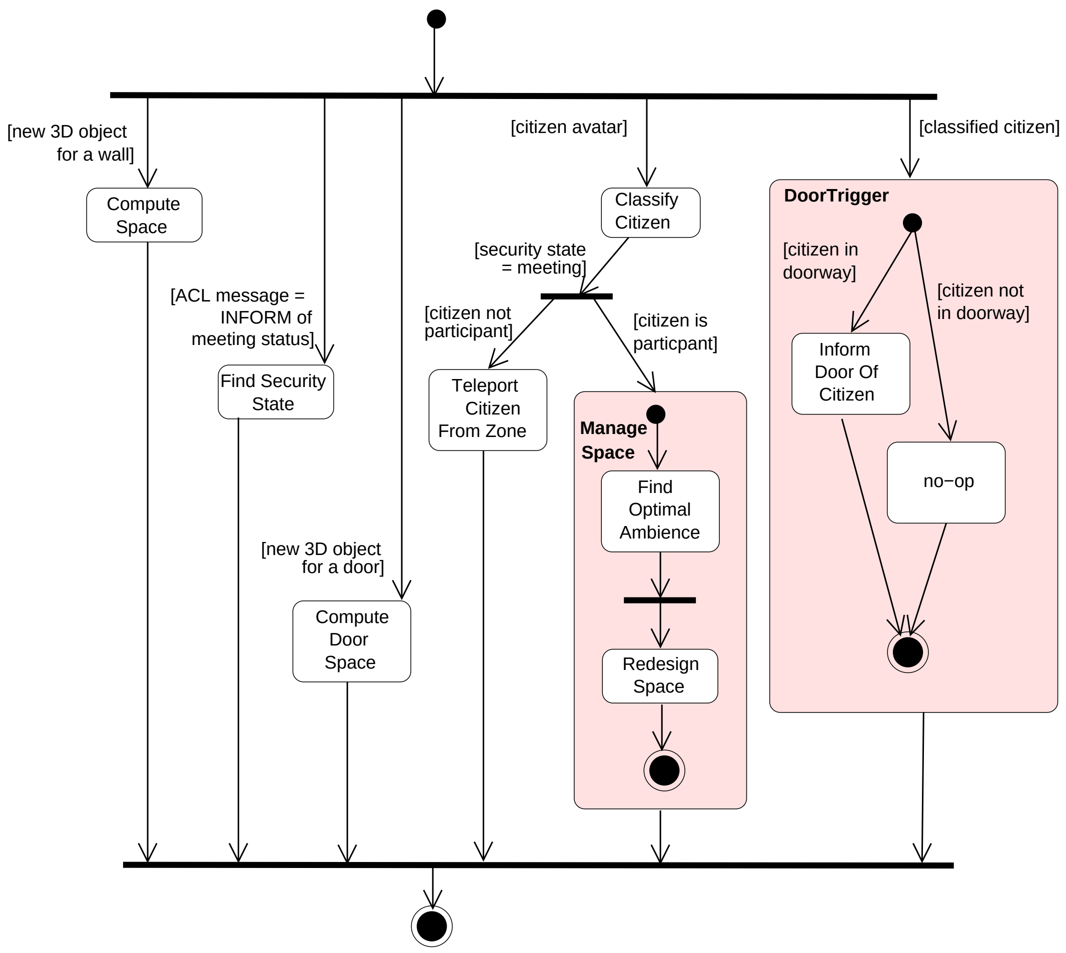


Figure 9: Partial UML activity diagram of zone action.

to Distributed Artifcial Intelligence, pages 79–120. MIT Press, Cambridge MA, 1999.

[Jennings, 2000] Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[Maher and Gero, 2002] Mary Lou Maher and John S Gero. Agent models of 3D virtual worlds. In G Proctor, editor, *ACADIA 2002: Thresholds, Pamona, CA*, pages 127–138. ACADIA, 2002.

[Smith and Gero, 2002] Gregory J Smith and John S Gero. Interaction and experience: Situated agents and sketching. In J S Gero and F Brazier, editors, *Agents in Design 2002*, pages 115–132. Key Centre of Design Computing and Cognition, University of Sydney, 2002.

[Wooldridge, 2002] M Wooldridge. *An Introduction to MultiAgent Systems*. Chichester, England: John Wiley and Sons, 2002.