AN OBJECT-ORIENTED SIMULATION OF AUTONOMOUS AGENTS IN A COMPLEX PHYSICAL ENVIRONMENT

Douglas B. Craig
Graduate Student
Oakland University
Rochester, MI 48309-4401

Abstract
A design for a simulation of autonomous agents acting in a physical environment is outlined. The simulation is concurrent object-oriented in design using a continuum of message passing protocols. Messages, which account for interaction between all simulation objects, are non-symbolic and tightly coupled to the methods used for their interpretation. This coupling is expressed through the use of meta-methods which constrain the relationship of classes of messages with respect to classes of methods. Modeling is endomorphic and support is provided for the construction of dynamic modeling hierarchies internal to the agent. The dispersal of messages through such a hierarchy is discussed with respect to issues of aggregation and subsumption with special attention being given to realization of reflective computation in such an architecture.

1. Introduction
In designing a simulation of multiple independent agents coexisting in some environment, a seemingly natural approach is to treat the physically independent agents as functionally independent objects in the simulation. Further, by also representing independent functional aspects of these agents as simulation objects, a complete object-oriented design can be adopted. One functional aspect typical of any agent operating autonomously, is its ability to reflect upon its situation in the environment. This ability is characteristically thought to derive from the agents ability to internally model its environment (including itself). Implicitly then, any simulation of autonomous agents must not only model agents and the environment within which they reside, but also each agent's ability to model the environment. This recursive form of modeling, termed endomorphic by Zeigler [19], must take into account that these two levels of modelling differ fundamentally in purpose. At the simulation level the model is descriptive; at the agent level the model is functional. In addition, at the simulation level the model is designed complete; whereas, at the agent level it is experientially deduced. Any simulation of autonomous agents must provide a framework which is at once supportive of both levels of modelling.

While both levels are similar in form, as alluded to previously, they originate in disparate fashion. The simulation level is essentially a static description of what is to be simulated; it is necessarily designed in. At the agent level, however, the agent is left to derive this structure in the course of its interaction with the environment. Both levels can be modeled in a hierarchical, modular manner, but any such architecture must extend this modelling methodology to support dynamic (run-time) construction of models. This requirement is crucial if the agent is to maintain a model based on its own changing conception of the environment. An additional constraint on dynamic model building is that it must be sufficient at all times. Sufficiency, in this case, means that the agent can apply the model to all manner of situation. Success is not guaranteed, but some level of "appropriate" action is necessary. This implies that there is continuity, or continuous improvement, in the construction of models. This paper proposes an object-oriented simulation [8,11,14,17,19] design to facilitate dynamic model construction at the agent level. Mechanisms for interpreting the model are proposed which maintain continuity over the course of the model's evolution.
2. Simulation Design

It is the goal of this author's work in simulating autonomous agents to gain some insight into the design of real-world machines. It is, therefore, important that the simulation maintain some level of fidelity in its modeling of the physical environment. The agent is at once an adaptive system making control decisions and a physical object interacting on a very low level with its environment. These two perspectives are realized in a meta-architecture [7,9] with interaction ranging from purely spatial (at the physical level), to purely functional (at the abstract/control level). To accomplish this a range of message passing protocols is available to simulation objects as shown in Figure 1:

Because an agent's internal models are dynamic, messages are subject to changing interpretations. This implies that the methods servicing these messages must be changing too. To accomplish this meta-methods are employed. These methods provide an inheritance mechanism for evolving structures without imposing the details of implementation. This weak inheritance constrains methods to a class of operations on a specific class of messages. Dynamically, this gives rise to hierarchies of methods. An isomorphism exists between hierarchies for physical objects, and hierarchies for abstract/control objects. For physical objects this hierarchy represents levels of aggregation [12,13], whereas for abstract objects it represents levels of abstraction. It is proposed that these hierarchies are especially useful in the application of reflection [9,16] since they provide an implicit abstraction of any internal model.

Various aspects of the simulation design are discussed in more detail in the following sections.

2.1 Object Classes

There are three basic classes of objects: physical, abstract (or control), and sensor/actuators. As shown in Figure 1, these objects support a continuum of message passing protocols. Physical objects reside in space, therefore, their interaction (i.e., messages) can be initiated by mediating spatial phenomena such as: contact, proximity, and line-of-sight. This differs significantly from classical object-oriented languages (e.g., Smalltalk, C++) which typically interact only on a functional level. Interactions in the real-world depend more on the fortuitousness of proximity than on functional relationships. It is this type of classical, functional relationship, however, that is typical of abstract/control objects. At both the simulation and agent level these objects model functional relationships between objects and provide the means by which control decisions are made. The bridge between these two types of objects is provided by the sensor/actuator objects. Although they reside in space, they provide the means by which physical interaction can be interpreted, and by which abstraction can initiate action. These objects thus accept and pass messages between abstract objects and physical ones.

2.2 Messaging

With the introduction of spatially instantiated messages, messaging is necessarily asynchronous. The simulation language is thus concurrent by design [1,2,6]. A distinctly non-symbolic approach is also taken to messages. In classical object-oriented languages each message has associated with it an implicit function -- the sender knows what is being asking for. Messages from the environment, however, have no meaning in isolation. These messages are always subject
to the perspective of the agent. Message and method form a dynamic couple in which messages are assimilated into extant methods, while methods accommodate new messages. The two are mutually constrained by the mode (or source) of messages, and inheritance from meta-methods. Message modes serve to define classes of messages. These modes include: from what object type a message originated, whether the message conveys spatial or temporal information, and contextual information about the method which formed the message (see Constructors below).

2.3 Methods

Since messages are assumed to be dynamic (i.e., non-symbolic) all methods must be inherited from meta-methods. These meta-methods constrains methods to operate on classes of messages. As described in the previous section, messages are classed based on their origin, the dimension of their information, and other implicit contextual information. Meta-methods insure that continuity is maintained between message and method as the model evolves. A meta-method does this by employing a set of constructors which are used in the inheritance process to build new methods.

2.3.1 Constructors

Constructors typically operate along four dimensions: assimilation, accommodation, specialization, and generalization. Assimilation and accommodation define how existing methods are to change their response to new messages. Specialization and generalization, on the other hand, specify how new methods are created from old.

2.3.1.1 Assimilation

Assimilation constructors define how messages are to be interpreted in the context of existing methods. Since messages are non-symbolic, their form can vary considerably. Assimilation is perspective from the method end. In this case what the method expects is what it will see.

2.3.1.2 Accommodation

Accommodation constructors represent perspective from the message end. In this case greater weight is placed on the integrity of the message, thus forcing the method to change. For reasons of stability accommodation is typically a much more gradual process; however, mediating conditions may demand immediate accommodation for critical messages.

2.3.1.3 Specialization

The specialization constructors are designed to spawn new methods within the hierarchy. Specifically, messages which distinguish themselves with a few separable cases are candidates for specialization. These constructors specify how messages are to be broken into sub-messages.

2.3.1.4 Generalization

Generalization constructors act as the reciprocal of specialization by spawning super-methods. This type of activity is usually only useful across message modes. In this case methods are designed to look for commonalities between messages resulting in the formation of new message classes (see §3.1.2).

2.3.2 Meta-Method Classes for Naming

As an example of how meta-classes are organized, consider a class of methods for naming other messages. This type of method figures importantly in the functioning of reflection (see §2.5 and §3.4). Functionally, naming amounts to forming a classification set and then generating a new message, the name, to represent this set. In this case, the meta-method constrains the dimensions along which classification can take place for certain types of naming.

2.3.2.1 Naming Physical Objects

Naming physical objects means that classification must occur along dimensions of sense. At its most basic, a physical object must arise as a message from some sensor. Although more complex physical objects may cross multiple sensors and be removed from direct sensing by several levels of hierarchy, they will always be rooted in the sensors.

2.3.2.2 Naming Temporal Patterns

Naming temporal patterns is typically a specialization of other naming methods. Classification occurs along the dimension of time for other names. For physical objects this specialization names changes in state of an
object, or more generally, action. Instantaneously, the physical object's name does not change, but individual members of the classification set may change (see §3.3).

2.3.2.3 Naming Intensional Objects
Naming intensional objects [10] can be thought of as naming names. Functionally, intensional objects serve as place holders for objects yet to be identified. This concept is of particular importance in the service of goal/objective definition (see §3.4). In this case the meta-method constrains naming to occur only on other name messages. Intensional objects are therefore messages serving as abstract concepts.

2.4 Hierarchies
As mentioned previously, an isomorphism exists between the interpretation of hierarchies of physical objects and of hierarchies of abstract objects. For physical objects the hierarchy can be viewed as increasing levels of aggregation, whereas for abstract object it can be viewed as increasing levels of abstraction. When viewed in the context of control, abstract objects in the hierarchy exhibit a subsumption architecture [4]. Levels of abstract objects thus show increasing specificity of function. This is isomorphic to how levels of physical objects show increasing specificity of form.

2.4.1 Aggregation in Physical Objects
Hierarchies of physical objects, whether a product of the simulation level or the agent level, represents a physical description of objects with increasing specificity in form, composition, and extent. Roughly, form can be equated with resolution; resolution, both of appearance and mechanics. For example, at low resolution a human is a torso with a head, two arms, and two legs. Mechanically, the arms and legs alternate in a swinging fashion as the human moves. These descriptions represent aggregate forms of the object and are subsumed by increasing detail about appearance and mechanics (e.g., the arm is composed of multiple segments with varying degrees of freedom). At the simulation level levels of aggregation can serve to control the focus of attention on the model [12].

At the agent level aggregation provides a framework upon which to refine dynamic models.

Because messaging between physical objects is mediated by spatial conditions, aggregation can also provide a framework for handling extended physical objects [12]. Objects which span large spatial areas can be made to distribute spatially disparate messages down the hierarchy. This allows for more efficient interaction with extended objects without sacrificing the aggregate properties of the whole (see §3.1.1).

2.4.2 Aggregation in Abstract Objects
In hierarchies of abstract objects aggregation denotes increasing specificity of function or behavior. Because abstract objects operate in a control capacity, their hierarchical organization closely resembles a subsumption architecture [3,4]. A distinguishing feature of this architecture is its support of messages operating at multiple levels of the hierarchy. In this manner, previously subsumed layers are actively used to abstract levels of performance (a.k.a. competence). This type of information is crucial to the operation of reflection in dynamic model building.

2.5 Reflection
Reflection is the process by which an agent reasons about itself [9,15,16]. Since reasoning generally entails action at some point, reflection directly affects the way in which the agent itself reasons. Endomorphic systems [19] represent a form of reflection since models internal to the simulated agent can reason about the agent itself (Figure 2).

Specifically, there are two distinct levels of modelling: the simulation level which maintains a description of the entire environment, and the agent level which develops a functional model of its own environment. While both levels model the same environment, their origin and purpose differ wholly. Mechanisms for realising reflection at the agent level are discussed in §3.4.
3. Autonomous-Agent Model

Ziegler [18] proposes three levels of achievement for autonomous systems:

- **Level 1**: achieve prespecified objectives
- **Level 2**: adapt to major environment changes
- **Level 3**: develop its own objectives

With respect to the simulation design described thus far, these levels of achievement can be interpreted as follows. Level 1 assumes that the agent has some static hierarchical model of its environment. This model is exercised when incoming messages from the environment are interpreted with respect to some goal resulting in some action taken by the agent. In this case the hierarchy is fixed and the system operates in much the same way as a classical object-oriented language. Level 2 competency requires that the model become dynamic to some extent. Methods are no longer constant; instead, they accommodate changes in message content. This change need not take place at the hierarchy level, only through the interpretation of messages moving about the hierarchy (i.e., assimilation and accommodation). Finally, with Level 3 competency the hierarchy itself must become fully dynamic. This gives the agent complete control of the modeling process. The mechanisms by which an agent achieves these increasing levels of competency is presented in the following sections.

### 3.1 Hierarchical Dispersal of Messages

At both the simulation level and the agent level messages are dispersed in parallel along all layers of a hierarchy. Mechanisms of assimilation assure that only those portions of a message of interest to a particular object are received. This general scheme is qualified by the fact that contextual cues may instigate aggregation (e.g., at the simulation level this may be done to control the focus of attention). More details are given below.

#### 3.1.1 Physical Objects

When physical objects interact there are mediating contextual cues. Specifically, resolution of the interaction defines the layer at which messages are interchanged. For example, if a complex physical object is viewed at a great distance by a medium resolution sensor, messages querying the form of the object should not extend below the level perceivable by the sensor. Also true of sensors is the fact that resolution can extend along the time dimension (i.e., sampling). For agent level models this implies that attention can play a role in whether a sensor is receptive to stimuli from the environment. Similarly, sensors can be modeled such that they respond only to changes in input messages (i.e., stimuli).

#### 3.1.2 Abstract Objects

Dispersal of messages in abstract object hierarchies can have varying effects depending on the level at which the hierarchy resides. At the simulation level the hierarchy is descriptive; at the agent level it is adaptive. This endomorphic relationship (see §2.5) implies reflective use of the hierarchy. For this purpose the abstract object hierarchy at the agent level is best viewed as a subsumption architecture. Because it is evolving, higher layers represent cruder methods of handling messages. Although crude, these layers can potentially be viewed as abstractions of subsequent subsuming layers. In fact, by applying generalisation, parent layers can be made to better represent abstractions of subsuming layers. In this case, generalisations
are propagated back up the hierarchy in a bottom-up refinement process.

3.2 Perspective

One example of perspective is the way in which a viewed object inquires about the resolution of the viewer's sensor. In this case the viewer gets exactly what is expected (in terms of resolution). This type of interchange between the sensed and the sensor make possible a clean handling of extended objects. Perspective thus has a literal meaning when applied to physical objects.

Perspective as applied to abstract objects takes on a slightly different character. It is more useful to think of perspective as a refinement of the message for a particular receiver. This refinement may include filtering, supplemental information, and transformation. As previously described, perspective relies heavily on the relationship between meta-methods and their associated message classes. At Level 1 competency this relies almost exclusively on assimilation. In this case methods are assumed fixed for a particular model; assimilation acts to fit messages to the correct method. A first step toward Level 2 competency is through the use of accommodation.

Accommodation represents the first point at which the model is considered dynamic. In this case messages are assimilated to existing methods but have the side effect of making adjustments to the way in which future messages are interpreted. A key to accommodation is the dynamic classification of messages. Rather than assume that messages fall into fixed classes, accommodation reacts to new messages by adjusting the classes. Because messages and methods are coupled through meta-methods, this results in concomitant changes to the method.

3.3 Dynamic Classification

In Level 2 competency messages are dynamically classified. One method of classification is through the use of polymorphous sets [5]. Using this method a class is constructed based on some set of n features discernible in messages. For membership a message must possess any m (or more) of these features. By dynamically classifying messages, feedback can be provided to the method. This information can subsequently be used to exercise: accommodation (by discarding or adding membership features), specialization (by determining separable sets of messages), and generalization (by showing common features of heretofore different message classes).

To further enhance the effectiveness of dynamic classification, reflective methods can be employed. This entails gathering other information about messages (e.g., their origin, use, frequency). The means by which this type of information can be gathered is call reflective interpretation.

3.4 Reflective Interpretation

Reflective interpretation describes a class of methods which are capable of traversing model hierarchies. In the process of traversal, these methods generate a class of reflective messages which can subsequently be used by the model. Figure 3 shows an example of reflective interpretation:

![Figure 3](image)

In this example message activity over time is shown with by the activity arrow. Each object along this path is shown situated in its respective hierarchy. In this fragment activity shows "putting" something. Reflectively the agent can know that something has been put by traversing up each hierarchy from an active node to its root. Associated with each root node is a name message which describes the abstract activity of each respective hierarchy. This highlights two keys to reflective interpretation: traversal of active objects, and the naming of message classes (see §2.3.2).
Reflective interpretation is proposed as an enabling mechanism for Level 3 competency. Specifically, as alluded to previously, it can be used as a means to define new goals. Abstractly, a goal defines a slot that is to be filled. The slot is described as part of the goal, but the means by which it is to be filled is not. First, an intensional object (see §2.3.2.3) can be generated in the service of this goal. The generation of this object is based on the idea that some class of message (i.e., an abstraction) has been defined, but the method by which to generate a message of this class must be discovered. Reflection can then be employed to determine the method(s) best suited for generating a message to instantiate the intensional object.

4. Further Thoughts

Currently a prototype of the design given here is being implemented in C++ as part of the author's dissertation work at Oakland University. Specific attention is being given to run-time instantiation of methods based on meta-methods. Further details are also being worked out on just how reflexive interpretation might be used for goal selection, the evaluation of cause and effect, and the evaluation of plans as descent through the aggregation hierarchy.

References


