In Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Mueller, Wooldridge and Jennings (eds.), Lecture Notes in AI, 1193, 49-63, Springer-Verlag, 1997.

From Agent Theory to Agent Construction: A Case Study

Michael Luck* Nathan Griffiths* Mark d'Inverno†

* Department of Computer Science, University of Warwick
Coventry CV4 7AL, U.K.
{mikeluck,neg}@dcs.warwick.ac.uk

† School of Computer Science, University of Westminster
115 New Cavendish Street, London W1M 8JS, U.K.
dinverm@westminster.ac.uk

Abstract. There is a growing body of work that concentrates on theoretical aspects of agents and multi-agent systems, and a complementary body of work concerned with building practical systems. However, the two have typically been unrelated. This gap between the theory and practice of intelligent agents has only relatively recently begun to be addressed. In this paper we describe the construction of an agent simulation environment that is based strongly on a formal theory of agent systems, but which is intended to serve in exactly this way as a basis for practical development. The paper briefly introduces the theory, then describes the system and the simple reactive agents built with it, but most importantly shows how it reflects the theoretical framework and how it facilitates incremental agent design and implementation. Using this example as a case-study, some possibilities for a methodology for the development of agent systems are discussed.

1 Introduction

The field of agent-oriented systems is growing dramatically in many directions. Coupled with its relative youth, however, this has given rise to the concentration of research in distinct niches so that there are very different approaches to essentially similar problem areas with, in some cases, little or no interrelation. Such a fragmentation leads to a lack of consensus regarding such fundamental notions as agents and autonomous agents, and also impedes progress towards integrated approaches to agent theory and agent construction. As the field matures, the broader acceptance of agent-oriented systems will become increasingly tied to the availability and accessibility of well-founded techniques and methodologies for system development.

A major criticism of much formal or theoretical work is that while it is important and contributes to a solid underlying foundation for practical systems, no direction is provided as to how it may be used in the development of these systems. For example, Moreno and Sales [22], Sandu [28] and Dignum and van Linder [6] all describe valuable formal models of aspects of agents and their behaviour but these are unrelated to issues of system development. There are many agent logics, but their rôle in building systems is unclear.

Recently, however, some efforts have been made to provide a greater harmony between these two camps, and to integrate the complementary aspects. Wooldridge and

Jennings have developed a model of cooperative problem solving (CPS) [33] which attempts to capture relevant properties of CPS in a mathematical framework while serving as a top-level specification of a CPS system. Rao has attempted to unite theory and practice in two ways. First, he provided an abstract agent architecture that serves as an idealization of an implemented system and as a means for investigating theoretical properties [27]. A second effort developed an alternative formalization by starting with an implemented system and then formalizing the semantics in an agent language which can be viewed as an abstraction of the implemented system, and which allows agent programs to be written and interpreted [26]. Goodwin has also attempted to bridge the gap by providing a formal description in the formal specification language, Z, of agents, tasks and environments, and then defining agent properties in these terms [13].

We also view our enterprise as one of building programs; our formal models are intended to provide a means of moving down the path towards implemented systems in a principled and grounded fashion. In previous work, we have proposed definitions of *agency* and *autonomy*, and described how autonomy is distinct but is achieved by *motivating* agency. On the basis of these definitions, a formal but accessible framework for agents and autonomous agents has been constructed which specifies the entities and the ways in which they are related, and provides an operational account of their behaviour in an environment [16]. The framework itself has been refined to detail aspects of agent modelling and agent relationships such as cooperation and engagement [17], and it has also been applied to existing systems [9] and theories [8] as a means of *reformulation*.

This paper is concerned with precisely this problem, but in a different way — moving from a formal, theoretical description of generic agents to a practical, implemented computer system. Specifically, it describes how the formal agent framework can be refined and used to support the development of agent systems. Based on the framework, a simple computational environment for reactive agents has been constructed and used for running a range of performance experiments. The environment, and the agents within it, are implemented in C++, and this allows a relatively easy transition from the formal Z specifications to the level of working program.

The next section provides a brief outline of the formal agent framework, giving a selection of Z schemas that describe salient aspects, so that a reasonable context is available within which to situate this work. Section 3 describes the agent simulation environment and typical agent designs, together with a discussion of some experiments for which it has been used. After that, we consider the construction of the implementation in more detail, paying particular attention to the use of object-oriented methods for reflecting the structure of the framework in the development of the system. Section 5 then assesses the possibilities for a methodology of agent system development in the light of this work, and finally, we briefly review related work and present concluding remarks.

2 A Framework for Agency and Autonomy

While rapid progress continues to be made with agent-oriented systems, there is still an on-going debate as to what constitutes an agent. This is excellently illustrated in the variety of existing definitions surveyed and discussed by Franklin and Graesser, before

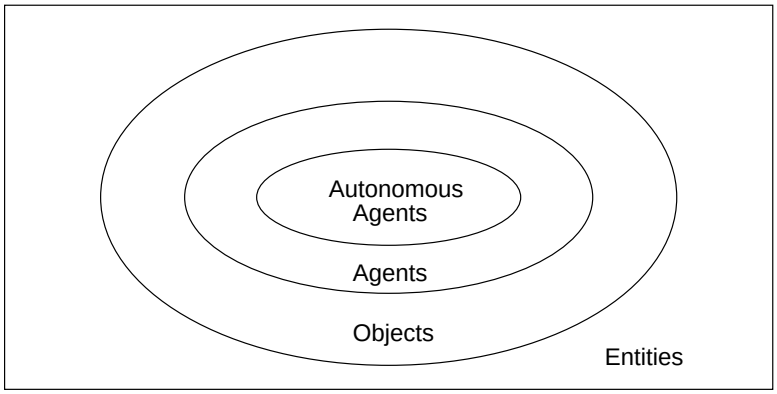

Fig. 1. The Entity Hierarchy

they provide their own encompassing definition and classification of agent properties [11]. Our position is similar; the definitions we use are inclusive and encompass many different kinds of agents with different properties or *dimensions*, though we do not distinguish between them [16]. The critical distinction that we do make, however, is that between agents and autonomous agents, an important aspect often overlooked.

In short, we propose a four-tiered hierarchy of entities comprising *entities*, *objects*, *agents* and *autonomous agents*. The basic idea underlying this hierarchy is that all components of the world are entities. Of these entities, some are objects, of which some, in turn, are agents and of these, some are autonomous agents. This is shown as a Venn diagram in Figure 1. In this section, we briefly outline the agent hierarchy. Many details are omitted — a more complete treatment can be found in [16].

Entities can be used to group together attributes for any useful purpose without adding a layer of *functionality* on top. They serve as a useful abstraction mechanism by which they are regarded as distinct from the remainder of the environment, and which can organise perception. An object is just something with abilities and attributes and has no further defining characteristics. An agent is an object that is useful to (typically another) agent where this usefulness is defined in terms of satisfying that agent's goals. In other words, an agent is an object with an associated set of goals. One object may give rise to different instantiations of agents which are created in response to another agent. This definition of agency relies upon the existence of other agents which provide goals that are adopted in order to instantiate an agent. In order to escape an infinite regress of goal adoption, we can define autonomous agents which are just agents that can generate their own goals from motivations.

For example, a table can be an object. It has attributes specifying that it is stable, made of wood, is brown in colour and has a flat surface. Its capabilities specify that it can support things. If I support my computer on a table, however, then the table is my agent for supporting the computer. The table may not actually possess the goal, but it is certainly satisfying, or can be *ascribed*, my goal to support the computer. A robot which rivets a panel onto a hull is also an agent, and if it has motivations such as hunger and achievement, then it is an autonomous agent.

Mathematically, we can describe this view of agents and provide a complete formal specification of it using the Z specification language [31]. This, and other specification languages such as VDM, are increasingly being used in the AI community (e.g. [4, 13, 32]). Below, we present the basic components of the framework. The presentation is kept as simple as possible, and our use of the notation should therefore be self-explanatory. We will not give excessive detail here, either of the notation or of the way in which these very simple structures have been elaborated to detail, for example, mechanisms for action-selection, perception, motivation evaluation and goal generation and adoption, and to formalise existing systems and theories [8, 9, 10, 17].

An entity is just something with a non-empty set of attributes that is, typically, used as a template for more sophisticated components. An object is an entity with the added constraint that it has a non-empty set of capabilities. Similarly, an agent is an object with a non-empty set of goals, and an autonomous agent is an agent with a non-empty set of motivations. Note the use of schema inclusion for incremental definition.

$$
\begin{array}{|l}
\hline
\mathit{Entity} \\
\mathit{attributes} : \mathbb{P}\ \mathit{Attribute} \\
\hline
\mathit{attributes} \neq \{\,\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\mathit{Object} \\
\mathit{Entity} \\
\mathit{capableof} : \mathbb{P}\ \mathit{Action} \\
\hline
\mathit{capableof} \neq \{\,\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\mathit{Agent} \\
\mathit{Object} \\
\mathit{goals} : \mathbb{P}\ \mathit{Goal} \\
\hline
\mathit{goals} \neq \{\,\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\mathit{AutonomousAgent} \\
\mathit{Agent} \\
\mathit{motivations} : \mathbb{P}\ \mathit{Motivation} \\
\hline
\mathit{motivations} \neq \{\,\} \\
\hline
\end{array}
$$

In summary, if there are grouped attributes but no capabilities, then we identify an *entity*. If there are no goals, then the entity is an *object*, and if there are goals but no motivations, then the entity is an *agent*. Finally, if neither the motivation nor goal sets are empty, then the entity is an *autonomous agent*.

In the next section, we describe the computational environment that is the subject of this paper and show how the basic components of the framework can be refined to provide a formal specification of the system. Due to space constraints we will only indicate, as opposed to giving a detailed specification, the form both of the abstract framework and the computational system.

3 An Agent Simulation Environment

This formal theory, while providing a principled foundation on which to base development of agent systems, does not specify, or even indicate, *how* such systems can be constructed. The aim of the work described in this paper is to provide an environment which allows the development and investigation of a variety of agent systems, within the confines of the framework. In particular, the framework specifies certain constraints on the design of agents and describes inheritance of properties between different classes of entity, from object to agent and from agent to autonomous agent.

There are several requirements that should be made of a system intended for such a purpose. First, the environment in which the agents are to be situated should be sufficiently flexible to allow any number of entities to exist. These should include both objects, which have no internal control but respond directly to the environment, and agents, whose control is provided both in relation to *mental components* and the external environment. Second, output from the system should be in two forms. In addition to a standard textual trace giving precise in-depth data regarding the step-by-step actions of agents, it should also have a minimal graphical interface to facilitate an easy but high-level analysis of agent behaviour. The following subsections describe *typical* entities in each class, some of which are provided within the environment as predefined systems components.

3.1 Entity and Object Representation

All entities are situated in the agent simulation environment (ASE) which is taken to be a very simple unit torus in order to avoid imposing artificial boundaries. Explicit boundaries may be designed and incorporated into the environment by constructing *entities*, the most basic component in the world. Locations and sizes are determined in relation to the unit torus, specified in terms of x and y coordinates, to any desired level of accuracy allowed by the underlying architecture. All entities (and therefore objects and so on) in ASE are square, and are either stationary or move in some way.

We can introduce into the formal specification subtypes which relate to such classes of attributes. For example, the type, $Size$, defines the attributes which specify size, and so on.

$$Size == Attribute$$
$$Location == Attribute$$
$$Shape == Attribute$$
$$Velocity == Attribute$$

This allows us to formally *model* ASE entities by adding more detail to the earlier entity schema.

$$
\begin{array}{|l}
\hline
ASEEntity \\
\hline
Entity \\
size : Size \\
shape : Shape \\
location : Location \\
velocity : Velocity \\
\hline
\{size, shape, location, velocity\} \subseteq attributes \\
\hline
\end{array}
$$

ASE *Entities* provide the facility for including *static* fixed barriers as mentioned above. Such static entities have a fixed location and size, which are represented by coordinate pairs in the attribute list, and have no further necessary features though optional ones are possible. Their set of actions are typically empty (or limited to occupying space determined by their size in the ASE world). A static entity is, therefore, a special case of an entity with zero velocity. In the following schema, we take zero to be the value representing zero velocity.

$$
\begin{array}{|l}
\hline
StaticASEEntity \\
\hline
ASEEntity \\
\hline
velocity = \mathsf{zero} \\
\hline
\end{array}
$$

As described above, all objects are derived from entities, and are just defined in terms of their attributes and actions. In our simulation environment, objects are computational components whose attributes are represented in a declarative fashion as a simple list describing it, and which are made externally globally accessible. The actions of the object are also limited, and are determined by the nature of the object.

The basic *dynamic* objects can have motions which are *linear*, *circular* or what we call *random*. Linear objects are defined and implemented in terms of entities with the additional feature of velocity, given in x and y directions. They simply move by a fixed amount, determined by the velocity vector, on each iteration. Circular objects move in circular motions determined by a given radius, angle and granularity of movement, and are also defined and implemented in terms of entities. Finally, random objects are circular objects whose radius and angle are varied randomly within a fixed range, leading to an unpredictable *wobbly* motion.

Actions include anything which can change the environment such as, for example, movebyvelocity, which is an action to move by a fixed amount in a linear direction determined by the current velocity. We can thus continue to refine the framework to provide a detailed specification of the ASE environment by defining linear objects as follows.

$$
\begin{array}{|l}
\hline
LinearASEObject \\
\hline
Object \\
ASEEntity \\
\hline
\mathsf{movebyvelocity} \in capableof \\
\hline
\end{array}
$$

These predefined entities allow basic components such as food, water, barriers and so on, as well as other, more sophisticated components, to be added to the agent simulation environment in a uniform and elegant manner that is consistent with the formal framework given earlier. That they are implemented in an object-oriented fashion allows the qualities of entity classes and inheritance between different classes to be preserved and extended across different types of entity within a class. This is a general point that holds true for the agent designs that follow, and which will be discussed in the context of implementation and methodology later in the paper.

3.2 Agent Design

Just as the objects above inherit properties from entities, so do agents, maintaining the relationships described in the original theory. Agents are objects with goals which are the mental components that drive intelligent behaviour. This behaviour does not need to be sophisticated for agency to exist and can instead be simple reactive behaviour such as wandering or avoiding objects. To be more precise, the *goal* of wandering leads to the *actions* that are concerned with the particular movements of the agent. Similar relations hold between other goals and actions.

In a reactive agent architecture, the predefined action sequences are executed to satisfy a goal in particular conditions. It is a prerequisite of an agent, therefore, that it is able to perceive the conditions that initiate behaviour.

In the framework, perception is defined to be a function of goals, the current environment and the potential view of the environment (see [16]). Here, a simplified version of the perception function for an agent is simply applied to the current environment and gives the current agent *view* as specified below. In this way, this process of refinement forces the clarification of assumptions in the design of agents.

$$
\begin{array}{|l}
\hline
ASEAgentPerception \\
\hline
environment : \mathbb{P}\ AttributeAgent \\
ASEEntity \\
view : \mathbb{P}\ Attribute \\
asewillperceive : \mathbb{P}\ Attribute \longrightarrow \mathbb{P}\ Attribute \\
\hline
view = asewillperceive\ environment \\
\hline
\end{array}
$$

In order to *implement* such an agent from a base object, all that is necessary is to specify the particular perception function so that the appropriate actions can be taken. Many such functions are possible, the simplest being to allow unlimited and omniscient perception, or to have null percepts. In ASE agents, the typical method of limiting perception is to define a *field of perception* by specifying a radius of completely accurate perception beyond which perception is impossible.

3.3 Simple Reactive Agents

Several different agents have been implemented. These require the inheritance of *behaviours* from previously defined instances of agents, or the implementation of new

behaviours. Among the behaviours implemented for the experimental agents designed in ASE are *wander*, *avoid-obstacle*, *flee*, and *pursue*. Wandering in agents is implemented as random movement as in the random objects described above. Avoiding obstacles requires the agent to alter direction away from any obstacle that comes within a certain specified distance within its field of perception. The *flee* behaviour relies on the ability of agents to inspect the characterising attributes of any entity within their field of perception so that they may flee from particular attributes or combinations of attributes. Finally, the *pursue* behaviour is the inverse of *flee* — agents will alter direction to move towards an entity with certain attributes, again within the field of perception.

Formally, behaviours are precompiled sequences of sets of actions.

$$Behaviour == \mathbb{P}(\mathsf{seq}\,(\mathbb{P}\,Action))$$

This allows us to specify an action-selection function for an agent. The following schema does not include details of a particular function, but specifies the form of all such functions so that it can be instantiated for any agent.

$$\begin{array}{|l} \hline ASEAgentAct \\ Agent \\ ASEAgentPerception \\ ASEEntity \\ aseagentactions : \mathbb{P}\,Goal \longrightarrow \mathbb{P}\,Attribute \longrightarrow Behaviour \\ \hline \end{array}$$

The action-selection functions allow behaviours to be ordered on priority, providing a subsumption-like architecture whereby less critical behaviours cede control to more important ones. An example agent might have its behaviours in the following order of priority: *avoid-obstacle*, *flee*, *pursue*, *wander*, where avoiding obstacles is most critical and wandering is least critical. Thus when the environment does not demand anything else, wandering is normal.

3.4 Autonomous Agents

Autonomous agents are defined to be agents with a higher-level control provided internally by motivations. Thus we can specify motivations of *curiosity*, *safety*, *fear*, *hunger*, and so on. In a simple agent design, we might then associate the motivation of *curiosity* with the goal of *avoiding obstacles* which, in turn, is associated with the actions required to achieve such results. In ASE, only a simple motivational mechanism is used, whereby each motivation has an associated strength value which can vary in response to the environment. Note that we are not concerned here with the large variety of possible methods of representing and manipulating motivations, but with the process of moving from theory to implementation. More sophisticated mechanisms are possible such as those described by Norman and Long [23, 24], Sloman [1, 29] and Moffat and Frijda [21, 20], for example, but we will not discuss them further.

The effects and causes of changes to the motivational state are similar to those described by Maes [18, 19], but in a much simplified version. The strength levels of motivations increase in certain situations defined to impact on them. For example,

in close proximity to an obstacle, the *safety* motivation is incremented by a fixed predetermined amount. If a behaviour which mitigates a motivation is performed, the strength of that motivation is decreased. When the strength of a motivation exceeds a given threshold, an appropriate behaviour is performed, often by direct association through a production rule. Alternatively, behaviour can be selected through a more sophisticated combination of conditions on motivation levels. *Meta-behaviours* which directly modify motivational levels in response to the environment and sometimes other motivational levels, are also possible.

It is worth noting that the agents and autonomous agents described above, and the manner in which they are related, are similar to the agents discussed by Davis [5] in using goals and motivations and a hierarchical structure.

3.5 Experiments

The simulation environment provides a general, computational framework within which agent development can take place in a relatively simple fashion. The specific aim of the environment is to facilitate the investigation of different agent architectures in a limited but modifiable setting. Thus, many different scenarios can be constructed. As part of an initial phase of experimentation, several scenarios were developed to provide a demonstration of the simulation environment capabilities.

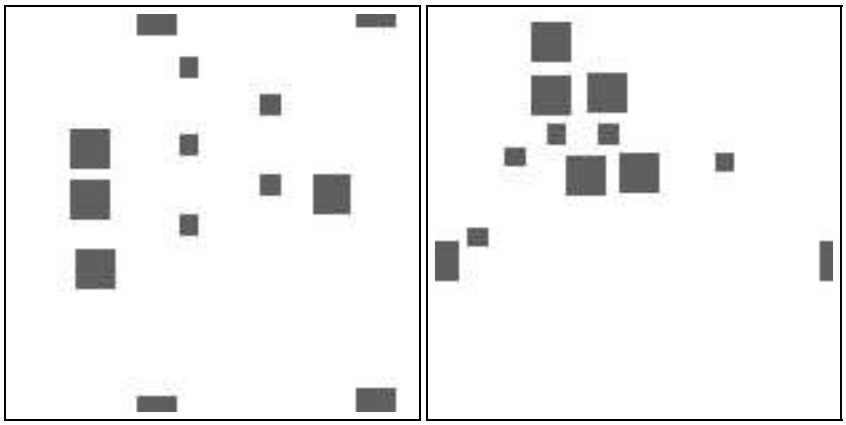

Fig. 2. The Pursuit Problem on initialisation and after several iterations.

For example, one experiment is a version of the well-known *pursuit problem* in which several *pursuer* agents attempt to capture *prey* agents. The pursuers are able to *wander* and *avoid-obstacles*, and also *pursue* the prey. Prey, too, can *wander* and *avoid-obstacles*, but they also *flee* from pursuers. With small numbers of agents, the pursuers are not able to capture the prey and continually wander around the environment. More pursuers enables effective capture, and more prey enables easier escape through distraction. The

left-hand side of Figure 2 gives a screen shot of an initial configuration of agents, where the larger squares represent pursuers, and the smaller squares represent prey. After a period of time, the agents converge to the sort of situation shown in the right-hand side of Figure 2. Here, the pursuers have moved from their initial random locations in the torus to attempt to capture the smaller prey in the upper center of the picture.

4 Implementation

The system is implemented using object-oriented methods in C++, based on the formal framework outlined earlier. It both relies upon the structure of the framework, and reflects it, so that they are very strongly related.

Underlying the simulation environment is a *container* data structure that contains all of the entities within the environment. This reflects the definition of an environment as collection of all the attributes present in the world, which are recoverable from the entities within it. Objects and agents, both predefined and user-created, are situated within this environment in a uniform way, and are linked together in the container class.

The simulation runs for a user-specified number of iterations, the granularity of the iterations determining the simulation time. On each iteration, each object or agent is inspected in turn and is updated in response to the environment. Specifically, perception functions are invoked to obtain an individual world view for each agent, any internal levels of motivation are updated if appropriate, and the initial conditions for each behaviour are checked to see if they become active in the current environment. Active behaviours are then performed, resulting in a change to the environment. An important aspect of this, however, is that the current state of an agent and its configuration are maintained by the agent itself rather than the environment so that all information is distributed and can only be *collected* centrally.

The formal definitions of agents and autonomous agents rely on inheriting the properties of lower-level components. In the Z notation, this is achieved through schema inclusion by which one schema is included in another which inherits all its properties. This is easily modelled in C++ by deriving one class from another. Thus, just as the agents are defined in terms of objects, and autonomous agents in terms of agents in the framework, the implemented agent classes are derived from object classes, and autonomous agent classes are derived from agent classes.

Figure 3 shows the structure of the system with a number of different components, including predefined and user-specified entities. The ASEEntity class is the base class for all subsequent objects and agents, and they all inherit its properties of size and location. the relationship of linear, circular and random objects has already been described above, and is well illustrated here. Agents with particular behaviours are all derived from the ASEEntity class, and have behaviours incrementally added. Thus the agent with *avoid-obstacle* and *wander* behaviours is derived from the ASEEntity and derives the agent with the additional *pursue* behaviour which, in turn, derives the agent with the *flee* behaviour. Similarly for autonomous agents, which are derived from those with a subset of behaviours. In agents and autonomous agents, these vertical lines indicate inheritance of methods of action-selection which can be essentially similar within a class. The thick horizontal lines, by contrast, denote the inheritance of behaviours across classes.

Thus an autonomous agent can be derived from a non-autonomous agent with similar behaviour together with an autonomous agent with similar action-selection.

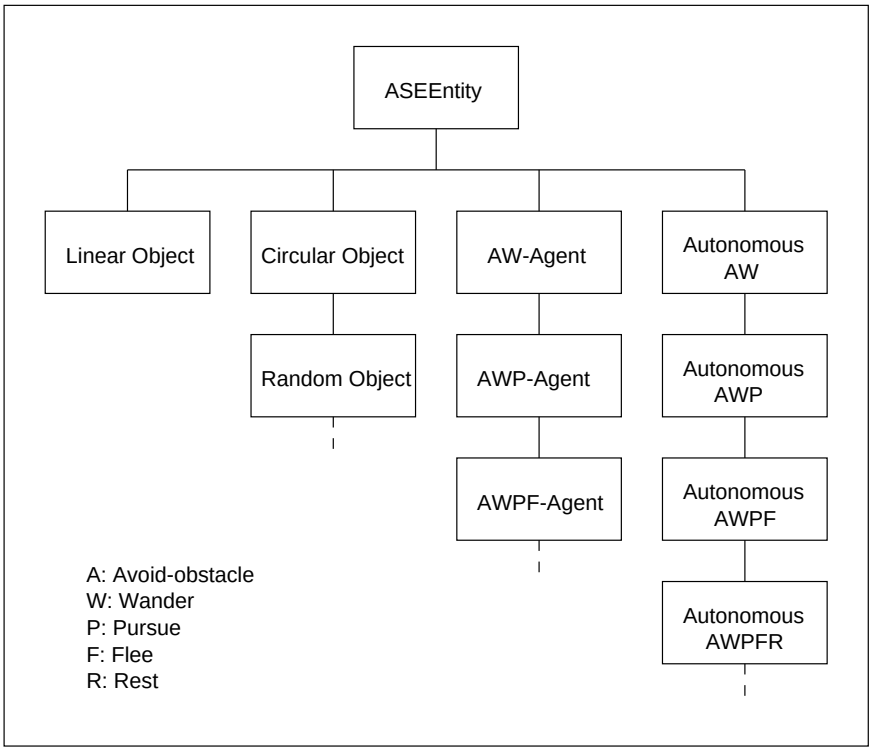

Fig. 3. The agent class hierarchy in ASE

This is not just an elegant means of relating agent architecture and design. It provides increasingly more sophisticated building blocks with which to construct more sophisticated agents incrementally in a rigorous and structured fashion. One question that arises from such a transition between theory and practice is to what extent this can be used as a basis for providing a methodology of agent-based systems. This question is addressed in the next section.

5 Towards a Methodology for System Development

As described in the introduction, theoretical and practical work in agent-oriented systems have tended to be separate and distinct, and in many cases completely unrelated. Our aim is not only to attempt to bridge this divide, but in so doing to move towards a principled methodology for the development of agent systems, and consequently provide a firmer footing for the more practical aspects of the field.

Kinny et al. [15] argue that "a clear conceptual framework that enables the complexity of the system to be managed by decomposition and abstraction," is vital in such a methodology. This is our starting point, and indeed our formal framework plays exactly this role, using the standard properties of the Z specification language to satisfy these requirements. In particular, we can construct a model of the computational system by refining the abstract definitions provided in the framework to include the relevant systems constraints. The ASE description given earlier, together with the sampling of schemas, sketches how this might be achieved. In addition, once a methodology has been established, it would become possible to use Z to formalise the methodology itself as has been done with high-performance systems [7].

It is important at this point to distinguish between three key and distinct aspects of our work. The first of these is the theoretical framework that enumerates the hierarchy of entities that includes objects, agents and autonomous agents. This, together with the various mechanisms and structures that have been elaborated from it (discussed elsewhere [8, 9, 10, 17]), provides the conceptual base from which to start. The second aspect is the natural formal expression of this framework in the Z specification language, which lends itself well to describing the sorts of structures and systems in which we are interested, and reflecting the structure of the conceptual base. Finally, the process of moving from concepts through specifications to systems suggests a methodology which can be used for building such agent-oriented systems more generally.

The first task is to identify each of the distinct entities in the application domain through an analysis of their functionality in terms of behaviour. That is to say that the result of this first step is an enumeration of all entities together with their purpose. Each of these entities can then be considered in terms of control, both with regard to themselves, and of others. This involves the examination of the dependencies that exist between entities, which rely on others to determine current behaviour and which are independent of others. At this point we should be able to classify each entity as an object, agent or autonomous agent, and then to use the analysis of functionality to design the necessary behaviours and methods for their control (essentially, the action-selection functions) for each. Finally, the hierarchical relationships between entities must be considered in more detail so that any structural similarities which can be exploited are revealed.

So far, we have restricted our concern to simple, primarily reactive, agents, for ease both of implementation and discussion. However, many other aspects of intelligent agents have been addressed by the framework with the construction of specifications for mechanisms for goal generation and adoption, for example. We envisage a library of such specifications for agent components and mechanisms, which can be combined in appropriate ways to construct agent systems in accordance with the above outline. Just as Kinny et al. consider reusable components for building Belief-Desire-Intention architectures based on object-oriented techniques and methodologies, we too aim to provide an effective means for software developers to implement such systems.

Indeed, object-oriented approaches provide an ideal paradigm for the implementation of the agents designed as a result of such a process. The structural relationships inherent in the multi-agent system can be readily captured by the abstraction provided by object classes, and the inheritance that is available within class hierarchies. Perhaps more importantly, object-oriented methods provide a means by which the model given in

formal specification can be easily transformed into an executable program with minimal effort, and making use of existing object or agent class libraries.

Thus we move from principled but abstract theoretical framework through a more detailed, yet still formal, model of the system, down to an object-oriented implementation, preserving the hierarchical structure at each stage.

6 Concluding Remarks

There have been very many efforts directed at the construction of simulation environments for agents with a variety of emphases. For example, TileWorld is a testbed intended to support experiments on a simulated robot agent in a dynamic and unpredictable grid-based environment [25]. The system is highly parameterised so that characteristics of both the agent and the environment can be controlled by the experimenter. Similarly, the Phoenix system provides a simulation environment in the domain of fighting forest fires. The initial state of the environment and the way it changes over time can be controlled by the experimenter, and detailed information of the environment and the agents within it can be collected [3]. Several other testbeds with have also been constructed (e.g. [2, 12, 30]), and provide a similar functionality to a greater or lesser extent. The environment described in this paper, however, is intended for a purpose very different to a generic agent testbed. Though it allows the construction of experiments regarding agent design, it is itself part of these experiments in providing a practical complement to, and a computational demonstration of, the formal framework that has been the basis for much work in the field. Moreover, the framework provides a structure both for the environment itself, and the agents within it, that facilitates the transition to executable program. This is the key characterising feature of the work described in this paper.

Though the choice of Z as a formal language has been valuable in allowing the expression of the framework in a natural and elegant way, it has also imposed some restrictions. For example, because there is no notion of time in Z, it is difficult to specify the sort of liveness and safety properties that can easily be specified in temporal logic. As we progress further, we anticipate adding to Z with another formalism more suited to the particular task at hand or, possibly, combining formalisms to obtain something with the desired properties such as CSP and Z (e.g. [14]).

The process of making the transition from formal framework to executable object-oriented program of the simulation environment also provides an indication of the way in which a methodology for the development of agent systems may be formulated, as described in the previous section. Our initial assessment of this shows many similarities to the work of Kinny et al. [15] who propose an agent-oriented methodology and modelling technique in much greater detail. By contrast, any suggestions made here regarding how to proceed on the methodological front are tentative. We have only one case-study and, though the relative ease with which the system was developed bodes well, more effort is necessary before definitive claims can be made.

References

1. L. P. Beaudoin and A. Sloman. A study of motive processing and attention. In *Prospects for Artificial Intelligence: Proceedings of AISB93*, pages 229–238, Birmingham, 1993.

2. T. Bouron, J. Ferber, and F. Samuel. MAGES: A multi-agent testbed for heterogeneous agents. In Y. Demazeau and J.-P. Müller, editors, *Decentralized AI 2: Proceedings of the Second European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 195–216. Elsevier, 1991.
3. P. R. Cohen, M. Greenberg, D. M. Hart, and A. E. Howe. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine*, 10(3), 1989.
4. I. D. Craig. *The Formal Specification of Advanced AI Architectures*. Ellis Horwood, 1991.
5. D. N. Davis. Reactive and motivational agents: towards a collective minder. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1996. In this volume.
6. F. Dignum and B. van Linder. Modeling social agents: Communication as action. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1996. In this volume.
7. M. d'Inverno, G. R. Justo, and P. Howells. A formal framework for specifying design methodologies. In *29th Annual Hawaii International Conference on System Sciences*, pages 741–750. IEEE Computer Society Press, 1996.
8. M. d'Inverno and M. Luck. A formal view of social dependence networks. In *Proceedings of the First Australian DAI Workshop, Lecture Notes in Artificial Intelligence, 1087*, pages 115–129. Springer Verlag, 1996.
9. M. d'Inverno and M. Luck. Formalising the contract net as a goal directed system. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, pages 72–85. Springer-Verlag, 1996.
10. M. d'Inverno and M. Luck. Understanding autonomous interaction. In W. Wahlster, editor, *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 529–533. John Wiley & Sons, 1996.
11. S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents . In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1996. In this volume.
12. L. Gasser, C. Braganza, and N. Hermann. MACE: A flexible testbed for distributed AI research. In M. Huhns, editor, *Distributed Artificial Intelligence*, pages 119–152. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1987.
13. R. Goodwin. Formalizing properties of agents. Technical Report CMU-CS-93-159, Carnegie-Mellon University, 1993.
14. M. B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3:9–18, 1988.
15. D. Kinny, M. Georgeff, and A. Rao. A methodology and modelling technique for systems of BDI agents. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, pages 56–71. Springer-Verlag: Heidelberg, Germany, 1996.
16. M. Luck and M. d'Inverno. A formal framework for agency and autonomy. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press / MIT Press, 1995.
17. M. Luck and M. d'Inverno. Engagement and cooperation in motivated agent modelling. In *Proceedings of the First Australian DAI Workshop, Lecture Notes in Artificial Intelligence,*

1087, pages 70–84. Springer Verlag, 1996.
18. P. Maes. The dynamics of action selection. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 991–997, Detroit, 1989.
19. P. Maes. How to do the right thing. *Connection Science*, 1(3):291–323, 1989.
20. D. Moffat and N. H. Frijda. Where there's a will there's an agent. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages, LNAI 890*, pages 245–260. Springer-Verlag: Heidelberg, Germany, 1995.
21. D. Moffat, N. H. Frijda, and R. H. Phaf. Analysis of a model of emotions. In *Prospects for Artificial Intelligence: Proceedings of AISB93*, pages 219–228, Birmingham, 1993.
22. A. Moreno and T. Sales. Dynamic belief analysis. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1996. In this volume.
23. T. J. Norman and D. Long. Goal creation in motivated agents. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages, LNAI 890*, pages 277–290. Springer-Verlag: Heidelberg, Germany, 1995.
24. T. J. Norman and D. Long. Alarms: An implementation of motivated agency. In M. Wooldridge, J.P. Muller, and M. Tambe, editors, *Intelligent Agents: Theories, Architectures, and Languages, LNAI 1037*, pages 219–234. Springer-Verlag: Heidelberg, Germany, 1996.
25. M. E. Pollack and M. Ringuette. Introducing the tileworld: Experimentally evaluting agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189, 1990.
26. A. S. Rao. Agentspeak(l): BDI agents speak out in a logical computable language. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, pages 42–55. Springer-Verlag: Heidelberg, Germany, 1996.
27. A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning*, pages 439–449, 1992.
28. G. Sandu. Reasoning about collective goals. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1996. In this volume.
29. A. Sloman. Motives, mechanisms, and emotions. *Cognition and Emotion*, 1(3):217–233, 1987.
30. A. Sloman and R. Poli. SIM_AGENT: A toolkit for exploring agent designs. In M. Wooldridge, J.P. Muller, and M. Tambe, editors, *Intelligent Agents: Theories, Architectures, and Languages, LNAI 1037*, 1996.
31. J. M. Spivey. *The Z Notation*. Prentice Hall, Hemel Hempstead, 2nd edition, 1992.
32. M. Wooldridge and D. Vandekerckhove. MYWORLD: An agent-oriented testbed for distributed artificial intelligence. In S. M. Deen, editor, *Proceedings of the 1993 Workshop on Cooperating Knowledge Based Systems (CKBS-93)*, pages 263–274. DAKE Centre, University of Keele, UK, 1994.
33. M. J. Wooldridge and N. R. Jennings. Formalizing the cooperative problem solving process. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, 1994.

This article was processed using the LaTeX macro package with LLNCS style