A Conceptual Framework for Agent Definition and Development

Michael Luck
Department of Computer Science
University of Warwick
Coventry, CV4 7AL
United Kingdom
EMAIL Michael.Luck@dcs.warwick.ac.uk
TEL +44 24 7652 3364
FAX +44 24 7657 3024

Mark d'Inverno
Cavendish School of Computer Science
University of Westminster
London, W1M 8JS
United Kingdom
EMAIL Mark.dInverno@westminster.ac.uk
TEL +44 20 7911 5131
FAX +44 20 7911 5089

Abstract

The use of *agents* of many different kinds in a variety of fields of computer science and artificial intelligence is increasing rapidly and is due, in part, to their wide applicability. The richness of the agent metaphor that leads to many different uses of the term is, however, both a strength and a weakness: its strength lies in the fact that it can be applied in very many different ways in many situations for different purposes; the weakness is that the term *agent* is now used so frequently that there is no commonly accepted notion of what it is that constitutes an agent. This paper addresses this issue by applying formal methods to provide a defining framework for agent systems. The Z specification language is used to provide an accessible and unified formal account of agent systems, allowing us to escape from the terminological chaos that surrounds agents. In particular, the framework precisely and unambiguously provides meanings for common concepts and terms, enables alternative models of particular classes of system to be described within it, and provides a foundation for subsequent development of increasingly more refined concepts.

1 Introduction

Over the last five to ten years, the notions underlying agent-based systems have become almost commonplace, yet were virtually unknown in earlier years. Not only have agent-based systems moved into the mainstream, they have spread beyond a niche area of interest in artificial intelligence, and have come to be a significant and generic computing technology. The dramatic and sustained growth of interest is demonstrated by the increasing number of major conferences and workshops in this very dynamic field, covering a depth and breadth of research that testifies to an impressive level of maturity for such a relatively young area.

Some of the reasons for the growth in popularity of the field (apart from the obvious intuitive appeal of the agent metaphor) can be seen in the progress made in complementary technologies [1], of which perhaps the most dramatic has been the emergence of the World Wide Web. The distribution of information and associated technologies lend themselves almost ideally to use by, in and for multi-agent systems, while the problems that arise as a consequence suggest no solution quite as much as agents. The dual aspect of this interaction with the World Wide Web has thus been a major driving force. Other contributing factors include advances in distributed object technology that have provided an infrastructure without which the development of large-scale agent systems would become much more difficult and less effective. For example, the CORBA distributed computing platform [2], to handle low-level interoperation of heterogeneous distributed components, is a valuable piece of technology that can underpin the development of agent systems without the need for re-invention of fundamental techniques.

The contradiction of agent-based systems is that there is still an effort to provide a sound conceptual foundation despite the onward march of applications development. Indeed, while there are still disagreements over the nature of agents themselves, significant commercial and industrial research and development efforts have been underway for some time [3, 4, 5, 6], and are set to grow further.

A recurrent theme that is raised in one form or another at many agent conferences and workshops is the lack of agreement over what it is that actually constitutes an agent. It is difficult to know if this is a help or hindrance, but the truth is that it is probably both. On the one hand, the immediately engaging concepts and images that spring to mind when the term is mentioned are a prime reason for the popularisation of agent systems in the broader (and even public) community, and for the extremely rapid growth and development of the field. Indeed the elasticity in terminology and definition of agent concepts has led to the adoption of common terms for a broad range of research activity, providing an inclusive and encompassing set of interacting and cross-fertilising sub-fields. This is partly responsible for the richness of the area, and for the variety of approaches and applications. On the other hand, however, the lack of a common understanding leads to difficulties in communication, a lack of precision (and sometimes even confusion) in nomenclature, vast overuse and abuse of the terminology, and a proliferation of systems adopting the agent label without obvious justification for doing so. The discussion is valuable and important, for without a common language, there can be significant barriers to solid progress, but it is problematic to find a way to converge on such a language without constraining or excluding areas in the current spectrum of activity.

In this paper we seek to address the aforementioned problems by providing a sound conceptual framework with which to understand and organise the landscape of agent-based systems. Our approach is not to constrain the use of terminology through rigid definition, but to provide an encompassing infrastructure that may be used to understand the nature of different systems. The benefit of this is that the richness of the agent metaphor is preserved throughout its diverse uses, while the distinct identities of different perspectives are highlighted and used to direct and focus research and development according to the particular objectives of a sub-area.

The paper begins with a brief review and overview of agents, and then argues for the use of formal methods in providing a foundational framework for them, and describes the notation used here. Section 4 introduces the agent framework, and Section 5 describes the key agent components in detail. The paper ends with a discussion of the implications of this view of agents and how it enables the field to be organised and understood, and with a consideration of the ways in which the framework can be extended to describe more specific agent architectures.

2 The Agent Landscape

2.1 Terminology

In artificial intelligence (AI), the introduction of the notion of agents is partly due to the difficulties that have arisen when attempting to solve problems without regard to a real external environment or to the entity involved in that problem-solving process. Thus, though the solutions constructed to address these problems are in themselves important, they can be limited and inflexible in not coping well in real-world situations. In response, agents have been proposed as *situated* and *embodied* problem-solvers that are capable of functioning effectively and efficiently in complex environments. This means that the agent receives input from its environment through some sensory device, and acts so as to affect that environment in some way through effectors. Such a simple but powerful concept has been adopted with remarkable speed and vigour by many branches of computer science because of its usefulness and broad applicability.

Indeed, there is now a plethora of different labels for agents ranging from the generic *autonomous agents* [7], *software agents* [8], and *intelligent agents* [9] to the more specific *interface agents* [10], *virtual agents* [11], *information agents* [12], *mobile agents* [13, 14], and so on. The diverse range of applications for which agents are being touted include operating systems interfaces [15], processing satellite imaging data [16], electricity distribution management [17], air-traffic control [18] business process management [19], electronic commerce [20] and computer games [21], to name a few.

The richness of the agent metaphor that leads to such different uses of the term is both a strength and a weakness. Its strength lies in the fact that it can be applied in very many different ways in many situations for different purposes. The weakness, however, is that the term *agent* is now used so frequently that there is no commonly accepted notion of what it is that constitutes an agent. Given the range of areas in which the notions and terms are applied, this lack of consensus over meaning is not surprising. As Shoham [22] points out, the number of diverse uses of the term *agent* are so many that it is almost meaningless without reference to a particular concept of agent.

That there is no agreement on what it is that makes something an agent is now generally recognised, and it is standard, therefore, for many researchers to provide their own definition. In a relatively early collection of papers, for example, several different views emerge. Smith [23] takes an agent to be a "persistent software entity dedicated to a specific purpose." Selker [24] views agents as "computer programs that simulate a human relationship by doing something that another person could do for you." More loosely, Riecken [25] refers to "integrated reasoning processes" as agents. Others take agents to be computer programs that behave in a manner analogous to human agents, such as travel agents or insurance agents [26] or software entities capable of autonomous goal-oriented behaviour in a heterogeneous computing environment [27], while some avoid the issue completely and leave the interpretation of their agents to the reader. Many such other agent definitions can be found in the excellent review by Franklin and Graesser [28], in advance of proposing their own definition.

Typically, however, agents are *characterised* along certain dimensions, rather than defined precisely. For example, in the now foundational survey of the field by Wooldridge and Jennings [9], a *weak notion* of agency is identified that involves *autonomy* or the ability to function without intervention, *social ability* by which agents interact with other agents, *reactivity* allowing agents to perceive and respond to a changing environment, and *pro-activeness* through which agents behave in a goal-directed fashion. To some extent, these characteristics are broadly accepted by many as representative of the key qualities that can be used to assess '*agentness*'.

Wooldridge and Jennings also describe a *strong notion* of agency, prevalent in AI which, in addition to the *weak* notion, also uses mental components such as belief, desire, intention, knowledge and so on. Similarly, Etzioni and Weld [26] summarise desirable agent characteristics as including *autonomy*, *temporal continuity* by which agents are not simply 'one-shot' computations, believable *personality* in order to facilitate effective interaction, *communication ability* with other agents or people, *adaptability* to user-preferences and *mobility* which allows agents to be transported across different machines and architectures. They further characterise the first of these, autonomy, as requiring that agents are *goal-oriented* and accept high-level requests, *collaborative* in that they can modify these requests and clarify them, *flexible* in not having hard, scripted actions, and *self-starting* in that they can sense changes and decide when to take action. Other characteristics are often considered, both implicitly and explicitly, with regard to notions of agency including, for example, *veracity*, *benevolence* and *rationality*.

Wooldridge and Jennings recognise that many such qualities have been proposed by others as being necessary for agenthood but, in a joint paper with Sycara [29], suggest that the four characteristics enumerated in their *weak notion* above are the "essence" of agenthood. Despite some broad acceptance of this view, there are still many problems. For example, in a more recent paper, Müller [30] seeks to survey *autonomous* agent architectures by considering the three strands of *reactive* agents, *deliberative* (or pro-active) agents and *interacting* (or social) agents. The properties here correspond perfectly to three of these four key characteristics, but instead of being used to represent all agents, they are used to break down the classes of agents into three distinct streams of research.

The difficulty with this approach of *characterising* agents through identifying their properties is exemplified by considering *mobile agents* [13, 14], which are quite distinct and identifiable in the focus on movement of code between host machines. Here, the key characteristic is precisely this mobility, and indeed mobility has been regarded by some as an intrinsic agent property as noted earlier. A critical analysis of the area of mobile agents would, however, unearth a recognition that this mobility augments other, more central agent characteristics in mobile agents, so that mobility is valuable in identifying the kind of agent, rather than understanding all agents. Similarly, some of the more specific labels for agents describe other characteristics that do not impact on agents as a whole, but relate to a particular domain or capability.

This area is fraught with difficulty, yet there have been several efforts to address these issues. For example, in attempting to distinguish agents from programs, Franklin and Graesser constructed an agent taxonomy [28] aimed at identifying the key features of agent systems in relation to different branches of the field. Their aim, amply described by the title of the paper, "Is it an agent or just a program?", highlights what might be regarded as the problem of the *Emperor's clothes*, as to whether there is any value to the notion of agents. The definition provided, that an "autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to affect what it sense in the future," serves to distinguish some non-agent programs from agents through the introduction of *temporal continuity*, for example, but still suffers from simply providing a *characterisation.* Using this, Franklin and Graesser then move to classify existing notions of agents within a taxonomic hierarchy. While interesting and valuable, it still does not provide a solution to the problem of identifying agentness. As Petrie points out, for example, *autonomy* remains *unelaborated*, yet it is a key part of the definition [31].

In somewhat similar fashion, Müller [30] also provides a taxonomy of intelligent agents that reflects different application areas, and which can be used to identify classes of agent architectures that are suited to particular problems. While this is also a valuable aid to understanding the range of work done in this area, it does not help in clarifying the issues discussed above. Nwana [32], too, offers an interesting typology of agent systems in his review of the field, but importantly warns against the dangers associated with the "rampant' use of the agent buzzword, its overselling and the possibility of it becoming a "passing fad" as a result.

Thus, while agent properties illustrate the range and diversity both of the design and potential application of agents, such a discussion is inadequate for a more detailed and precise analysis of the basic underlying concepts. If we are to be able to make sense of this rapidly growing area, then we need to progress beyond a vague appreciation of the nature of agents. Indeed, as Wooldridge argues [33], to avoid the term 'agent' becoming meaningless and empty and attached to everything, only those systems that merit the agent label should have it. In this paper, we address this problem by describing a formal framework that we have constructed for *autonomy* and *agency*, which attempts to bring together these disparate notions [34].

3 Formalising Agents

3.1 The Need for Formality

There are many threads of research in artificial intelligence but, to a greater or lesser extent, they can be grouped under the banner either of experimental work, or of formal, theoretical work. This distinction, sometimes characterised as *scruffy* and *neat* styles, has led to a gap that has arisen between the formal logics used to describe theories and systems on the one hand, and the less formal language of those involved in the construction of software.

Recently, however, some efforts have been made to provide a greater harmony between these two camps, and to integrate the complementary aspects. For example, Wooldridge and Jennings have developed a model of cooperative problem solving (CPS) [35] that attempts to capture relevant properties of CPS in a mathematical framework while serving as a top-level specification of a CPS system. In another strand of work, 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 [36]. 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 [37]. Goodwin has also attempted to bridge the gap by providing a formal description of agents, tasks and environments, and then defining agent properties in these terms [38].

Similar concerns have arisen in software engineering. As computer systems become more sophisticated and complex, the associated difficulties of effectively managing the development process increase dramatically [39]. The combination of large systems, in many cases with hundreds of thousands of lines of code, with complex concurrent, distributed and real-time applications, leads to serious concerns over the quality and correctness of the software product. When there are critical safety and security issues involved, informal analyses can prove inadequate in assuring the quality of the software. Testing can lead to improved quality, but is limited in its effectiveness. In order to address this *software crisis*, a vast array of tools and techniques has arisen, including those described under the banner of *formal methods*.

Formal specification, in particular, has been concerned with the description of a software design and its properties in a mathematical logic or some other formal notation. Instead of using natural language with all its inherent vagueness and ambiguity, such formal notations provide a means for precise specification.

The effort by Goodwin at integration in AI described above highlights a new awareness of the possibility, and indeed desirability in many cases, of adopting such software engineering techniques in addressing the problems faced in artificial intelligence by the demands of both theoretical analysis and system development. Krause et al., for example, have investigated the formal specification of a medical decision support system [40], Craig has specified both blackboard architectures [41] and a production-rule interpreter [42], Wooldridge has formally described MYWORLD, a testbed for experimentation in distributed AI [43], and Milnes has described the SOAR cognitive architecture [44]. More complete surveys of related work with formal methods and formal languages in software engineering and artificial intelligence are provided by van Harmelen and Fensel in a series of papers [45, 46, 47].

3.2 Requirements of Formal Frameworks

As argued by Garlan and Notkin [48], software systems are not typically conceived in isolation, but are related to a variety of other systems with which they share common design features. These commonalities, however, are rarely exploited, and only in typically unstructured ways. Thus there is, in software engineering in general as in agent research and development, a proliferation of designs and implementations which are seemingly unrelated, or related in vague and uncertain terms, despite many such common features. Two reasons are given for this: first, different designs reflect different requirements; second, the common properties of the designs are poorly understood.

The claim is then made that formal specification can address these issues by defining a framework through which common properties of a family of systems can be identified. As a result of such a specification, it becomes possible to see how different systems can be considered as instances of one design, and how new designs can be constructed out of an existing design framework [49, 50, 51].

Similarly, our intention in this paper is to provide a formal framework in which to situate agent research and so facilitate fruitful discussion and effective communication. We argue that such a *formal framework* must satisfy three distinct requirements, as follows.

1. A formal framework must precisely and unambiguously provide meanings for common concepts and terms and do so in a readable and understandable manner. The availability of readable explicit notations allows a movement from vague and conflicting informal understandings of a class of models towards a common conceptual framework. A common conceptual framework exists if there is a generally held understanding of the salient features and issues involved in the class of models relevant to the problem.

2. The framework should be sufficiently well-structured to provide a foundation for subsequent development of new and increasingly more refined concepts. In particular, it is important for a practitioner to be in a position in which to choose the level of abstraction suitable for their current purpose or task.

3. Alternative designs of particular models and systems should be able to be explicitly presented, compared and evaluated with relative ease within the framework. The framework must therefore provide a description of the common abstractions found within that class of models as well as a means of refining these descriptions further to detail particular models and systems.

By constructing such a framework, we also aim to achieve the benefits suggested by Garlan [52] in discussing formal reusable frameworks. Framework development costs can be offset by a family of products rather than just one, different products can display an element of *desirable uniformity* and reusable software components can be employed. Of particular interest, given our earlier discussion regarding agent definitions, is that the act of constructing a generic framework for many products can lead to especially *elegant abstractions*, and to *cleaner definitions of fundamental concepts behind applications.*

3.3 The Z Notation

We have adopted the specification language, Z [53], in the current work for two major reasons. First, it is sufficiently expressive to allow a consistent, unified and structured account of a computer system and its associated operations. Structured specifications, which are made possible by Z allowing schemas and schema inclusion, enable the description of systems at different levels of abstraction, with system complexity being added at successively lower levels. Second, we view our enterprise as that of building programs. Z schemas are particularly suitable in squaring the demands of formal modelling with the need for implementation by allowing transition between specification and program. Thus our approach to formal specification is pragmatic — we need to be formal to be precise about the concepts we discuss, yet we want to remain directly connected to issues of implementation and program development.

The Z language is increasingly being used both in industry and academia, as a strong and elegant means of formal specification, and is supported by a large array of books (e.g. [54, 55, 56]), articles (e.g. [57, 58, 59, 60]) and development tools. Furthermore, Z is gaining increasing acceptance as a tool within the artificial intelligence community (e.g. [34, 38, 41, 44]) and is therefore appropriate in terms of standards and dissemination capabilities.

3.4 Syntax

The formal specification language, Z, is based on set theory and first order predicate calculus. It extends the use of these languages by allowing an additional mathematical type known as the *schema type*. Z schemas have two parts: the upper declarative part, which declares variables and their types, and the lower predicate part, which relates and constrains those variables. The type of any schema can be considered as the Cartesian product of the types of each of its variables, without any notion of order, but constrained by the schema's predicates. Modularity is facilitated in Z by allowing schemas to be included within other schemas. We can select a state variable, *var*, of a schema, *schema*, by writing *schema.var*.

To introduce a type in Z, where we wish to abstract away from the actual content of elements of the type, we use the notion of a *given set*. We write $[NODE]$ to represent the set of all nodes. If we wish to state that a variable takes on some set of values or an ordered pair of values we write $x : \mathbb{P}\, NODE$ and $x : NODE \times NODE$, respectively.

A *relation* type expresses some relationship between two existing types, known as the *source* and *target* types. The type of a relation with source X and target Y is $\mathbb{P}(X \times Y)$. A relation is therefore a set of ordered pairs. When no element from the source type can be related to two or more elements from the target type, the relation is a *function.* A *total* function ($\rightarrow$) is one for which every element in the source set is related, while a *partial* function ($\nrightarrow$) is one for which not every element in the source is related. A sequence (*seq*) is a special type of function where the domain is the contiguous set of numbers from 1 up to the number of elements in the sequence. For example, the first relation below defines a *function* between nodes, while the second defines a *sequence* of nodes.

$$Rel1 = \{(n1, n2), (n2, n3), (n3, n2)\}$$

$$Rel2 = \{(2, n3), (3, n2), (1, n4)\}$$

The *domain* (dom) of a relation or function comprises those elements in the source set that are related, and the *range* (ran) comprises those elements in the target set that are related. In the examples above, $\operatorname{dom} Rel1 = \{n1, n2, n3\}$, $\operatorname{ran} Rel1 = \{n2, n3\}$, $\operatorname{dom} Rel2 = \{1, 2, 3\}$ and $\operatorname{ran} Rel2 = \{n2, n3, n4\}$.

Sets of elements can be defined using set comprehension. For example, the following expression denotes the set of squares of natural numbers greater than 10 : $\{x : \mathbb{N} \mid x > 10 \bullet x * x\}$.

Further details of the Z notation will be introduced as required through the course of the paper, but the framework is also presented in an informal and intuitive fashion. A summary of the notation

Definitions and declarations

a, b	Identifiers
p, q	Predicates
s, t	Sequences
x, y	Expressions
A, B	Sets
R, S	Relations
$d;\ e$	Declarations
$a == x$	Abbreviated definition
$[a]$	Given set
$A ::= b\langle\langle B\rangle\rangle \mid c\langle\langle C\rangle\rangle$	Free type declaration

Logic

$p \land q$	Logical conjunction
$p \Rightarrow q$	Logical implication
$\forall X \bullet q$	Universal quantification

Sets

$x \in A$	Set membership
$\varnothing$	Empty set
$A \subseteq B$	Set inclusion
$\{x, y, \ldots\}$	Set of elements
(x, y)	Ordered pair
$A \times B$	Cartesian product
$\mathbb{P}\, A$	Power set
$\mathbb{P}_1\, A$	Non-empty power set
$A \cap B$	Set intersection
$A \cup B$	Set union
$\bigcup A$	Generalized union
$\# A$	Size of a finite set
$\{d;\ e \ldots \mid p \bullet x\}$	Set comprehension

Relations

$A \leftrightarrow B$	Relation
$\mathrm{dom}\, R$	Domain of a relation
$\mathrm{ran}\, R$	Range of a relation
R^+	Transitive Closure

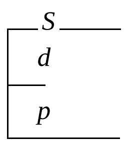
Functions

$A \nrightarrow B$	Partial function
$A \rightarrow B$	Total function

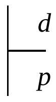
Sequences

$\mathrm{seq}\, A$	Set of finite sequences

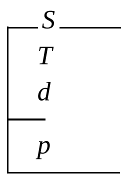
Schema notation



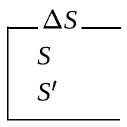
Vertical schema



Axiomatic definition



Schema inclusion



Operation schema

Conventions

a	State component before operation
a'	State component after operation
S	State schema before operation
S'	State schema after operation
ΔS	Change of state ($S \land S'$)
ΞS	No change of state

Figure 1: Summary of Z notation in the Agent Framework.

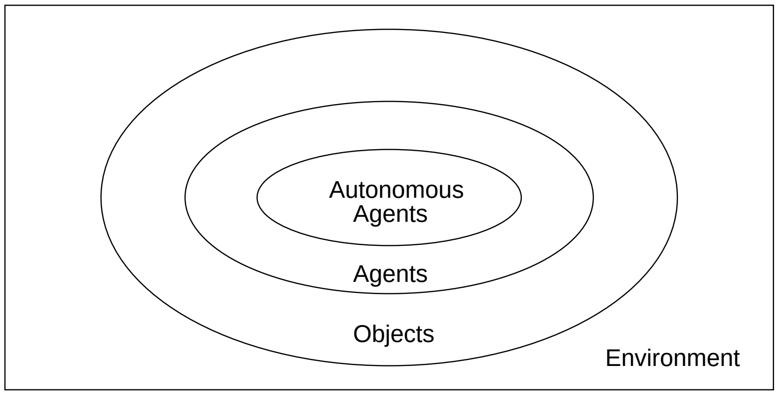


Figure 2: The Agent Hierarchy

to be used is given in Figure 1. For a more complete treatment of the Z language, the interested reader is referred to one of the numerous texts, such as [54, 61, 53]. Details of the formal semantics of Z are given in [56]. We will not consider such issues further in this paper.

4 An Agent Framework

As discussed in some detail above, there exist many diverse notions of agency, and there is a distinct lack of consensus over the meaning of the term. This is just the sort of problem that the construction of a formal framework of the kind considered in the previous section can address. In this section, we introduce the terms and concepts that are used to explicate our understanding of *agents* and *autonomous agents* and then developed into formal definitions.

According to Shoham [22], an *agent* is any entity to which mental state can be ascribed. Such mental state consists of components such as beliefs, capabilities and commitments, but there is no unique correct selection of them. This is sensible, and we too do not demand that all agents necessarily have the same set of mental components. Indeed, we recognise the limitations associated with assuming an environment comprising homogeneous agents and consequently deliberately direct this discussion at heterogeneous agents with varying capabilities. However, in our framework we do specify what is minimally *required* of an entity for it to be considered an agent. This approach is intended to be encompassing and inclusive in providing a way of relating different classes of agent, rather than an attempt to exclude through rigid definition.

Initially, we must describe the environment and then, through increasingly detailed description, define objects, agents and autonomous agents to provide an account of a general agent-oriented system. The definition of agency that follows is intended to subsume existing concepts as far as possible. In short, we propose a four-tiered hierarchy comprising *entities*, *objects*, *agents* and *autonomous agents*. The basic idea underlying this hierarchy is that an environment consists of entities, some of which are objects. Of this set of objects, some are agents, and of these agents, some are autonomous agents.

These four classes are the fundamental components that comprise our view of the world. Though the choice of autonomy as a fundamental distinguishing quality in this framework may not be immediately obvious in the light of the preceding discussion, its importance arises through the functionality of goals. As we will see below, goals define agency, and the generation of goals defines autonomy. In this sense, autonomy is not simply one of many possible characterising properties or qualities, but is foundational in its significance, and serves as a platform on which other properties can be imposed, just as agency.

The specification is structured so that it reflects our view of the world as shown in the Venn diagram of Figure 2. It must be built up in such a way that, starting from a basic description of an entity, each succeeding definition can be a refinement of that previously described. In this way, an object is a refinement of an entity, an agent is a refinement of an object, and an autonomous agent is a refinement of an agent. Accordingly, the specification is thus structured into four parts.

Entity and Environment The most abstract description we provide is of an entity, which is simply a collection of attributes. An environment can then be defined as a collection of entities.

Object We can also consider objects in the environment as collections of attributes, but we may also give a more detailed description of these entities by describing their capabilities. The capabilities of an object are defined by a set of action primitives which can theoretically be performed by the object in some environment and, consequently, change the state of that environment.

Agent If we consider objects more closely, we can distinguish some objects which are serving some purpose or, equally, can be attributed some set of goals. This then becomes our definition of an agent, namely, an object with goals. With this increased level of detail in our description, we can define the greater functionality of agents over objects.

Autonomous Agent Refining this description further enables us to distinguish a subclass of the previously defined class of agents as those agents that are autonomous. These autonomous agents are self-motivated agents in the sense that they pursue their own *agendas* as opposed to functioning under the control of another agent. We thus define an autonomous agent as an agent with motivations and, in turn, show how these agents behave in a more sophisticated manner than non-autonomous agents.

Just such a structured specification is facilitated in the Z specification language by describing a system at its highest level of abstraction with further complexity being added at each successive lower level of abstraction. An overview of the structure of our framework is given in Figure 3, where the boxes represent schemas, and the arrows represent schema inclusion. Definitions of *entity*, *object*, *agent* and *autonomous agent* are given by *Entity*, *Object*, *Agent* and *AutonomousAgent* respectively. This shows how we construct the most detailed entity description in the framework (of an autonomous agent situated in an environment) from the least detailed description (of an entity). Moving from the top of the figure to the bottom shows how decreasing levels of abstraction are achieved in the specification.

We define how objects, agents and autonomous agents act in an environment in the schemas *ObjectAction*, *AgentAction* and *AutonomousAgentAction* respectively, and we detail how agents and autonomous agents perceive in an environment in *AgentPerception* and *AutonomousAgentPerception*. In a similar fashion, *ObjectState*, *AgentState* and *AutonomousAgentState* define the state of objects, agents and autonomous agents when situated in an environment as defined by *EnvironmentState*.

5 The Framework Specification

5.1 Entities and Environment

Before it is possible to construct agent models it is necessary to define the building blocks or *primitives* from which these models are created. Formally, these primitives are specified as given sets.

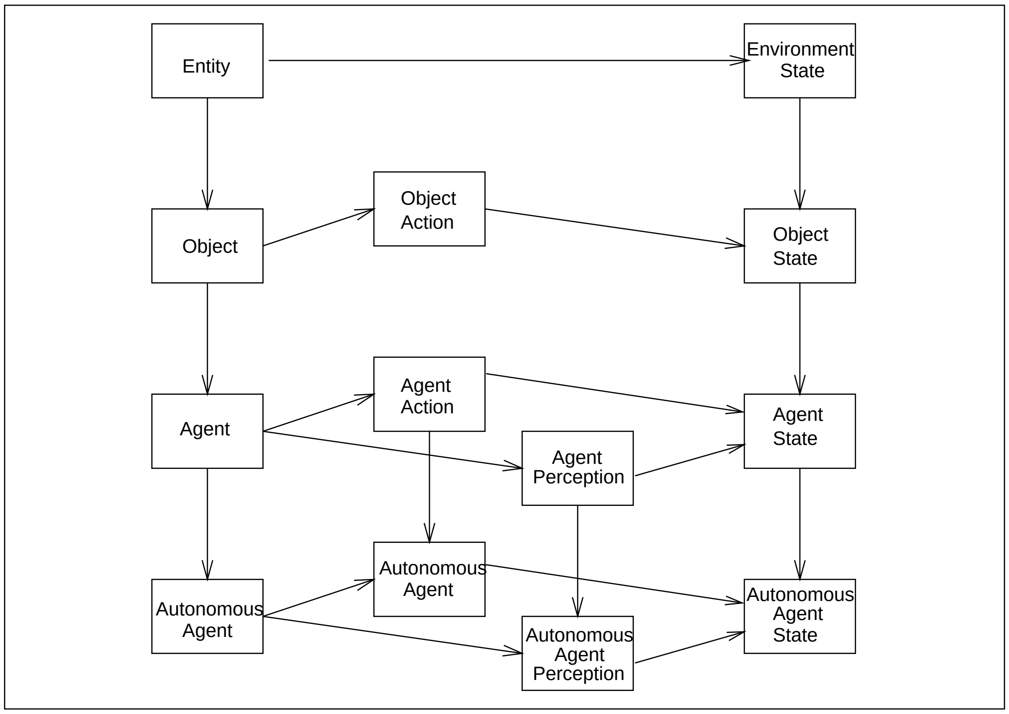


Figure 3: Schema Inclusion in the Agent Framework

First, we need to define the *attribute* primitive. Attributes are simply features of the world, and are the only characteristics that are manifest. They need not be perceived by any particular entity, but must be potentially perceivable in an omniscient sense. This notion of a feature allows anything to be included such as, for example, the fact that a tree is green, or is in a park, or is twenty feet tall. In this sense, attributes correspond to *propositional* fluents rather than *function* fluents such as the *colour* of the tree.

Definition: An *attribute* is a perceivable feature.

In Z, the set of all attributes, or the attribute type, is defined as follows.

$[Attribute]$

An *entity* is any component described at the very highest level of abstraction; just as a collections of attributes. We do not care which attributes are grouped together to describe a given entity, or how that is achieved. We are only concerned with being able to describe a collection of attributes as a single component.

Now, a state schema can be constructed that defines an entity. Very simply, an entity is a set of attributes with the constraint that the set is non-empty. (Making the constraint explicit in this way enables the difference between levels of the hierarchy to be highlighted as clearly as possible.)

Entity

$attributes : \mathbb{P}\, Attribute$

$attributes \neq \varnothing$

Considering the example of a tea-cup, its attributes of the cup may state that it is stable, blue, hard, and so on. Similarly, a robot's attributes may specify that it is red, large, heavy, and has three arms.

An environment is simply a set of attributes that describe *all* the features within that environment that are currently true. For the purposes of readability, we define a new type, *Env*, to be a (non-empty) set of attributes.

$$Env == \mathbb{P}_1\, Attribute$$

Now, an entity must be situated in an environment. Conversely, an environment will include all the entities within it, and this is formalised in the *EnvironmentState* schema below. The environment state is represented by the *env* variable of type *Env*, which must consist of a *non-empty* set of attributes, and the *entities* variable, which refers to the set of entities in the environment. The last predicate formalises the requirement that the sum of the attributes of the entities in an environment is a subset of all the attributes of that environment.

EnvironmentState

$$env : Env$$
$$entities : \mathbb{P}\, Entity$$

$$\bigcup\{e : entities \bullet e.attributes\} \subseteq env$$

While it may be possible to envisage scenarios where entities share attributes, for simplicity we take it to be the norm that entities are distinct elements with distinct attributes.

5.2 Objects

An object can be defined at a basic level in terms of its abilities as well as its attributes. In this sense, an object is just an entity with the capacity to interact with its environment. This notion provides us with the basic building block with which to develop our notion of agency.

5.2.1 Object Specification

In order to define objects, we need to introduce another primitive, for an *action*. Actions can change environments by adding or removing attributes. For example the action of a robot, responsible for attaching tyres to cars in a factory, moving from one wheel to the next, will delete the attribute that the robot is at the first wheel and add the attribute that it is at the second. This is not dissimilar to the notions of add and delete list used in STRIPS; this work is not intended to address the semantic difficulties of modelling actions in this way, but to provide a simple and inclusive representation.

Definition: An *action* is a discrete event that can change the state of the environment when performed.

$$[Action]$$

Again for the purposes of readability we define a new type, *Actions*, to be a set of Actions.

$$Actions == \mathbb{P}\, Action$$

Definition: An *object* comprises a set of actions and a set of attributes.

An object in our model then is an entity to which we ascribe notion of a set of basic capabilities. In the schema below, we introduce *capabilities*, which is the set of actions of the object, sometimes referred to as the *competence* of the object, and include the definition of an entity. An object is thus a refinement of an entity and is defined by its ability in terms of its actions, and its configuration in terms of its attributes, which includes references to the body of the object, and is similar to the notion used by Goodwin [38]. The attributes of an object are accessible from the environment (in the sense that they can be *perceived*), while the capabilities of an object are not.

The schema below formalises the definition of an object. It has a declarative part containing the *Entity* schema and the *capabilities* variable, as well as a predicate part that specifies that *capabilities* is non-empty.

$$\begin{array}{|l}
\hline \textit{Object} \\
\textit{Entity} \\
\textit{capabilities} : \textit{Actions} \\
\hline
\textit{capabilities} \neq \varnothing \\
\hline
\end{array}$$

As an example of an object, consider again the tyre-attaching robot, and assume that it does not have a power supply. Since the robot has no power, its capabilities are severely limited and include just those which rely on its physical presence, such as supporting things, weighing things down, and so on. Similarly, the capabilities of the tea-cup include that it can support things and that it can contain liquid. Once the robot is connected to a power supply it becomes a new object with increased capabilities, including being able to fix tyres to a car and reporting defective ones.

Since an object has actions, these may be performed in certain environments that will be determined by the state of that environment. The behaviour of an object can therefore be modelled as a mapping from the environment to a set of actions that are a subset of its capabilities. This mapping is known as the *action-selection* function, which is *defined* in the *ObjectAction* schema but applied in the *ObjectState* schema. In this respect, the information included in the *Object* and *ObjectAction* definitions relate to the intrinsic object properties and not to its state which is only defined once it is placed in an environment, as below.

Thus, *ObjectAction* schema below formalises this view and refines the *Object* schema, which is included in it. The action-selection function, *objectact*, determines which set of actions are performed next in a given environment and is defined as a total function. That is, given an environment, it returns a (possibly empty) set of actions. The assertion in the predicate part of the schema constrains the next actions to be taken by the object (determined by applying *objectact*) to be within the object's capabilities.

$$\begin{array}{|l}
\hline \textit{ObjectAction} \\
\textit{Object} \\
\textit{objectact} : \textit{Env} \rightarrow \textit{Actions} \\
\hline
\forall \textit{env} : \textit{Env} \bullet (\textit{objectact env}) \subseteq \textit{capabilities} \\
\hline
\end{array}$$

5.2.2 Object State

An object must be situated in an environment, which can be used to determine those actions that it is to perform next. Different states will give different actions as determined by application of the action-selection function.

We define the state of an object in its environment in the *ObjectState* schema, which formalises the state of an object in an environment and includes the schema representing the state of the environment, *EnvironmentState*, and the *ObjectAction* schema. This is equivalent to incorporating all of the declarations and predicates from both schemas. The variable, *willdo*, specifies the next actions that the object will perform. It is redundant since it is recoverable in exactly the way in which it is specified by applying the *objectact* function from the *ObjectAction* schema to the current environment, *env*, and is a subset of the capabilities of the object.

$$
\begin{array}{|l}
\hline
\textit{ObjectState} \\
\textit{EnvironmentState} \\
\textit{ObjectAction} \\
\textit{willdo} : \textit{Actions} \\
\hline
\textit{willdo} = \textit{objectact env} \\
\textit{willdo} \subseteq \textit{capabilities} \\
\hline
\end{array}
$$

For example, the tyre-attaching robot, in a situation that includes holding a tyre, may now have *willdo* as a set of actions to attach the tyre to the car.

5.2.3 Object Operation

So far, we have described an object and the way in which its actions are selected. Next, we describe how the performance of these actions affects the environment in which the object is situated. Those variables that relate to the state of the object (in particular, its next actions) can change, while the other variables that are not concerned with state but with the *nature* of the object itself (namely, its attributes, capabilities and action-selection function) remain unchanged. If these latter variables ever did change, then a *new* object would be instantiated. In this view a robot without a power supply is a different object from a robot with a power supply.

The Δ*ObjectState* schema shows how these constraints are formalised. It specifies that a change to the *ObjectState* schema will leave the *ObjectAction* schema unchanged by 'Ξ*ObjectAction*', which states that none of the variables in it (and in schemas included in it such as *Object*) are affected by a change of state, so that attributes, capabilities and the action-selection function do not change.

$$
\begin{array}{|l}
\hline
\Delta \textit{ObjectState} \\
\textit{ObjectState} \\
\textit{ObjectState}' \\
\Xi \textit{ObjectAction} \\
\hline
\end{array}
$$

Now, when actions are performed in an environment, we say that an *interaction* takes place. An interaction changes the state of the environment by adding and removing attributes. In our model, all actions result in the same change to an environment whether taken by an object, agent or autonomous agent. The function that formalises how the environment is affected by actions performed within it can therefore be defined *axiomatically*. It maps the current environment and the performed actions to the resulting environment.

$$
\begin{array}{|l}
\textit{inteffect} : \textit{Env} \rightarrow \textit{Actions} \nrightarrow \textit{Env}
\end{array}
$$

This allows us to model an object interacting with its environment. (In this paper we restrict out analysis to individual objects and do not consider multiple interacting objects. However, one possibility is to model them through interleaving this kind of interaction.) Both the state of the object and the environment change as specified by the schema *ObjectInteracts*. The resulting environment is determined by applying *inteffect* to the current state of the environment and the current set of actions. In turn, this environment then determines the next set of actions to be performed by applying *objectact* again.

$$\begin{array}{|l}
\textit{ObjectInteracts} \\
\Delta \textit{ObjectState} \\
\hline
\textit{env}' = \textit{inteffect env willdo} \\
\textit{willdo}' = \textit{objectact env}' \\
\hline
\end{array}$$

5.3 Agents

5.3.1 Introduction

There are many dictionary definitions for an agent. Wooldridge and Jennings [9] quote the definition of an agent as "one who, or that which, exerts power or produces an effect."[1] However, they omit the second sense of agent, which is given as "one who acts for another . . . ". This is important, for it is not the acting alone that defines agency, but the acting for *someone or something* that is defining. Indeed, Wooldridge and Jennings acknowledge the difficulties in a purely action-based analysis of agency.

In our view agents are just objects with certain dispositions. Specifically, we regard an object as an agent if it is serving some purpose[2]. They may always be agents, or they may revert to being objects in certain circumstances. For the moment, we concentrate on the nature of the disposition that characterises an agent. An object is an agent if it serves a useful purpose either to a different agent, or to itself, in which latter case the agent is *autonomous.* Specifically, an agent is something that satisfies a goal or set of goals (often of another). Thus if I want to use some object for my purpose, then that object becomes my agent. In some cases, goals are *actually adopted* by computational agents that have explicit goal representation. However, when dealing with non-computational entities, and entities that cannot represent goals explicitly, we can ascribe a goal to an entity, or we can anthropomorphise and state that the entity has *adopted* the goal.

5.3.2 Agent Specification

Before defining agents, we must first define goals. A goal describes a state of affairs that is desirable in some way. For example, a robot may have the goal of attaching a tyre to a car.

Definition: A *goal* is a state of affairs to be achieved in the environment.

We can define goals to be just (non-empty) sets of attributes that describe a state of affairs in the world.

$$\textit{Goal} == \mathbb{P}_1 \textit{Attribute}$$

An agent can then be defined in terms of an object as follows.

Definition: An *agent* is an object with a non-empty set of goals.

The formal description of an agent is specified by the *Agent* schema. This refines the object schema and constrains the set of goals to be non-empty.

[1] *The Concise Oxford Dictionary of Current English (7th edition)*, Oxford University Press, 1988.

[2] It may seem strange to consider an object in different ways depending on the view one adopts, but this provides an explicit representation of the relationship between one entity and another that enables a situation to be correctly analysed. For example, a cup may be viewed from one perspective as storing tea and from another as acting as a paperweight. These are distinct and separate agent roles, and should be treated as such.

```
┌─ Agent ──────────────────────────────
│ Object
│ goals : ℙ Goal
├──────────────
│ goals ≠ { }
└──────────────────────────────────────
```

Thus an agent has, or is *ascribed*, a set of goals that it retains over any instantiation (or lifetime). One object may give rise to different instantiations of agents. An agent is instantiated from an object in response to another agent. Thus agency is *transient*, and an object that becomes an agent at some time may subsequently revert to being an object.

Note, that this definition means that in the limiting case, very simple non-computational entities without perception can be agents. For example, a cup is an object. We can regard it as an agent and ascribe to it mental state, but it serves no useful purpose to do so without considering the circumstances. A cup is an agent *if* it is containing liquid and it is doing so to some end. In other words, if I fill a cup with tea, then the cup is my agent; it serves my purpose. Alternatively, the cup would also be an agent if it were placed upside down on a stack of papers and used as a paperweight. It would *not* be an agent if it were just sitting on a table without serving any purpose to any one. In this case it would be an object. As this example shows, we do not require an entity to be intelligent for it to be an agent. Clearly, the example of the cup is counter-intuitive and it is much more intuitive to talk about robots, but it is important to realise that any object, computational or otherwise, can be an agent once it is serving a purpose.

Considering the robot example, suppose now that the robot has a power supply. If the robot has no goal, then it cannot use its actuators in any sensible way but only, perhaps, in a random way, and must be considered an object. Alternatively, if the robot has some goal or set of goals that allow it to employ its actuators in some directed way, such as picking up a cup, or fixing a tyre onto a car, then it is an agent. The goal need not be explicitly represented, but can instead be implicit in the hardware or software design of the robot. It is merely necessary for there to be a goal of some kind. Note that it is nevertheless possible that random actions of a robot may satisfy some goal of entertainment or distraction, in which case the robot is considered to be an agent.

Returning to the example of the cup as my agent, it is clear that not everyone will know about this agency. If, for example, I am in a café and there is a half-full cup of tea on my table, there are several views that can be taken. It can be regarded by the waiter as an agent for me, storing my tea, or it can be regarded as an object serving no purpose if the waiter thinks it is not mine or that I have finished. The waiter's view of the cup either as an object or agent is relevant to whether he will remove the cup or leave it at the table. Note that we are not suggesting that the cup actually possesses a goal, just that there is a goal that it is satisfying.

These examples highlight the range of behaviour that is available from agents. The tea-cup is passive and has goals *imposed* upon and *ascribed* to it, while the robot is capable of actively manipulating the environment by performing actions designed to satisfy its goals.

5.3.3 Agent Perception

We now introduce perception. An agent in an environment may have a set of percepts available, which are the possible attributes that an agent could perceive, subject to its capabilities and current state. We refer to these as the *possible percepts* of an agent. However, due to limited resources, an agent will not normally be able to perceive all those attributes possible, and will base its actions on a subset, which we call the *actual percepts* of an agent. Indeed, some agents will not be able to perceive at all.

In the case of a cup, for example, the set of possible percepts will be empty and consequently the set of actual percepts will also be empty. The robot, however, may have several sensors that allow it to perceive. Thus it is not a requirement of an agent that it is able to perceive.

To distinguish between representations of mental models and representations of the *actual* environment, we define a type, *View*, to be the perception of an environment by an agent. This has an equivalent type to that of *Environment*, but now we can distinguish between physical and mental components of the same type.

$$View == \mathbb{P}_1\, Attribute$$

It is also important to note that it is only meaningful for us to consider perceptual abilities in the context of goals. Thus when considering objects without goals, perceptual abilities are not relevant. Objects respond directly to their environments and make no use of percepts even if they are available. We say that perceptual capabilities are *inert* in the context of objects.

An agent has a (possibly empty) set of actions that enable it to perceive its world, and which we call its *perceiving actions*. The set of percepts that an agent is potentially capable of perceiving is a function of the current environment and the agent's perceiving actions. Since agents are typically resource-bounded, they may not be able to perceive the entire set of attributes and select a subset based on their current goals. For example, the distributed Multi-Agent Reasoning System (dMARS) [62], may have a set of events that it has to process, where events correspond to environmental change. Each of these percepts is available to the agent but because of its limited resources it may only be able to process one event, and must make a selection based on its goals.

The perception capabilities of an agent are defined in the *AgentPerception* schema, which includes the *Agent* schema and refines it by introducing three variables. The set of perceiving actions is denoted by *peractions*, a subset of the capabilities of an agent, while the *canperceive* function determines the attributes that are potentially available to an agent through its perception capabilities. Notice that this function is applied to a physical environment (in which it is situated) and returns a mental environment. The second argument of this schema is constrained to be equal to *peractions*. Finally, the function, *agperceives*, describes those attributes actually perceived by an agent to deal with the problem of resource-bounds by which an agent cannot necessarily perceive everything and must focus on some subset according to its current goals, as mentioned above. This function is always applied to the goals of the agent and, in contrast to the previous function, takes a mental environment and returns another mental environment.

AgentPerception

$$\begin{array}{l}
Agent \\
peractions : Actions \\
canperceive : Env \rightarrow Actions \nrightarrow View \\
agperceives : \mathbb{P}\, Goal \rightarrow View \rightarrow View \\
\hline
peractions \subseteq capabilities \\
\forall\, env : Env;\ as : Actions \bullet \\
\quad as \in \operatorname{dom}(canperceive\ env) \Rightarrow \\
\quad\quad as = peractions \\
\operatorname{dom} agperceives = \{goals\}
\end{array}$$

Simple reflexive agents can also be modelled in this way. For such agents, however, the goals may not be represented explicitly but are implicit in the stimulus-response rules (or other such mechanisms) that drive the agents.

5.3.4 Agent Action

Now, any agent can be viewed as an agent, object or entity. For example, I may view a robot as an entity if I am solely interested in its colour; this provides all the information that I require and at the same time takes less modelling effort. If, on the other hand, I want the robot to perform a task for me, then I must model it at the agent level with goals included. At the agent level of abstraction, goals and perceptions as well as the environment can be viewed as directing behaviour. This is specified by the *agentact* function in the *AgentAction* schema below, which is dependent on the goals, the actual perceptions of the agent and the current environment itself. Since the *objectact* function is still applicable for modelling the agent solely at the object level, the *ObjectAction* schema is included. The first predicate requires that *agentact* returns a set of actions within the agent's capabilities, while the last predicate constrains its application to the agent's goals. If there are no perceptions, then the action-selection function is dependent only on the environment, as it is with *objectact*.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{AgentAction} \quad}}\\
\mathit{Agent}\\
\mathit{ObjectAction}\\
\mathit{agentact} : \mathbb{P}\,\mathit{Goal} \rightarrow \mathit{View} \rightarrow \mathit{Env} \rightarrow \mathbb{P}\,\mathit{Action}\\
\hline
\forall\, g : \mathbb{P}\,\mathit{Goal};\ v : \mathit{View};\ \mathit{env} : \mathit{Env} \bullet\\
\qquad (\mathit{agentact}\ g\ v\ \mathit{env}) \subseteq \mathit{capabilities}\\
\operatorname{dom} \mathit{agentact} = \{\mathit{goals}\}\\
\hline
\end{array}
$$

5.3.5 Agent State

Now, to describe an agent with capabilities and behaviours for perception and action situated in an environment, we include the two schemas previously defined for action and perception as well as the schema defining the agent as a situated object. The *AgentState* schema, which formalises an agent situated in an environment, therefore includes the schemas *AgentAction*, *AgentPerception* and *ObjectState*. In addition, since the attributes of the environment are now accessible, it is possible to specify the *possible percepts* and *actual percepts* of the agent. These are denoted by the variables, *posspercepts* and *percepts*, which are calculated using the *canperceive* and *agperceives* functions respectively.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\quad \mathit{AgentState} \quad}}\\
\mathit{AgentPerception}\\
\mathit{AgentAction}\\
\mathit{ObjectState}\\
\mathit{posspercepts}, \mathit{percepts} : \mathit{View}\\
\hline
\mathit{percepts} \subseteq \mathit{posspercepts}\\
\mathit{posspercepts} = \mathit{canperceive}\ \mathit{env}\ \mathit{peractions}\\
\mathit{percepts} = \mathit{agperceives}\ \mathit{goals}\ \mathit{posspercepts}\\
\mathit{peractions} = \{\,\} \Rightarrow \mathit{posspercepts} = \{\,\}\\
\mathit{willdo} = \mathit{agentact}\ \mathit{goals}\ \mathit{percepts}\ \mathit{env}\\
\hline
\end{array}
$$

Consider again the robot agent and the cup agent which are attaching tyres and storing tea respectively. Now, suppose that the robot also has perceptual capabilities that allow it to perceive attributes in its environment. Potentially, as a consequence of its current environment the robot may be able to perceive a multitude of attributes including that the car is red, a tyre is flat, the car door is open, and so on. Again, however, due to limited perceptual and processing abilities, and to the goal of attaching tyres, the actual percepts of the robot may only include that the tyre is flat and not the relatively insignificant attribute of the car being red. The cup agent, on the other hand, has no perceiving capabilities and consequently no possible or actual percepts.

Since goals are fixed for any agent, changes to the actual percepts of an agent affect its selection of actions. An agent without perceptions does not therefore have any increased functionality as a result of having goals, but the behaviour of an agent without perceptions can still be viewed and modelled in terms of goals affecting its action selection.

5.3.6 Agent Operation

Operations characterising agent behaviour are constrained to affect only certain aspects. The attributes, capabilities, goals, perceptual capabilities, and action and perception selection functions are unchanged by any operation. If any of these variables change, a new agent is instantiated. The only variables that may change are necessarily associated with the state of the agent such as its situation and possible and actual percepts. These constraints are formalised in the Δ*Agent* schema, which defines a change in agent state. It includes Δ*ObjectState* to ensure that only the state properties of objects change and, in addition, that variables included in the *AgentAction* and *AgentPerception* schemas are unaltered.

$$\begin{array}{|l}
\hline \Delta \mathit{AgentState} \\
\mathit{AgentState} \\
\mathit{AgentState}' \\
\Delta \mathit{ObjectState} \\
\Xi \mathit{AgentAction} \\
\Xi \mathit{AgentPerception} \\
\hline
\end{array}$$

When an agent acts in an environment, the environment changes according to the specific actions performed. This does not depend on whether the entity is an object or an agent. Thus the schema describing object interaction is still directly applicable. Formally, the *AgentInteracts* schema includes *ObjectInteracts* and affects the state of an agent as specified by Δ*AgentState.* The three predicates of this schema show explicitly how the schema variables are updated.

$$\begin{array}{|l}
\hline \mathit{AgentInteracts} \\
\Delta \mathit{AgentState} \\
\mathit{ObjectInteracts} \\
\hline
\mathit{posspercepts}' = \mathit{canperceive}\ \mathit{env}'\ \ \mathit{peractions} \\
\mathit{percepts}' = \mathit{agperceives}\ \mathit{goals}\ \mathit{posspercepts}' \\
\mathit{willdo}' = \mathit{agentact}\ \mathit{goals}\ \mathit{percepts}'\ \mathit{env}' \\
\hline
\end{array}$$

5.4 Autonomy

5.4.1 Introduction

The definition of agency developed so far relies upon the existence of other agents to provide the goals that are adopted when an agent is instantiated. In order to ground this chain of goal adoption, to escape what could be an infinite regress, and also to bring out the notion of *autonomy*, we introduce *motivation.*

Grounding the hierarchies of goal adoption demands that we have some agents that can generate their own goals. These agents are *autonomous* since they are not dependent on the goals of others, and possess goals that are *generated within* rather than *adopted from* other agents. Such goals are generated from *motivations*, higher-level non-derivative components characterising the nature of the agent, but which are related to goals. Motivations are, however, qualitatively different from goals in that they are not describable states of affairs in the environment. For example, consider the motivation *greed*. This does not specify a state of affairs to be achieved, nor is it describable in terms of the environment, but it may (if other motivations permit) give rise to the generation of a goal to rob a bank. The distinction between the motivation of greed and the goal of robbing a bank is clear, with the former providing a reason to do the latter, and the latter specifying what must be done[3].

A *motivated agent* is thus an agent that pursues its own agenda for reasoning and behaviour in accordance with its internal motivation. Since motivations ground the goal-generation regress, we claim that motivation is critical in achieving autonomy. An *autonomous agent* must be a *motivated* agent.

Although it draws on Kunda's work on motivation in psychology [63], the definition used for motivation above expresses its role but does not tie us to any particular implementation. Indeed, there are several views as to exactly how the role of motivation as defined here can be fulfilled. Simon, for example, takes motivation to be "that which controls attention at any given time," and explores the relation of motivation to information-processing behaviour, but from a cognitive perspective [64]. More recently, Sloman has elaborated on Simon's work, showing how motivations are relevant to emotions and the development of a computational theory of mind [65, 66]. Some have used motivation and related notions such as *motives* [67], and *concerns* [68], in developing computational architectures for autonomous agents while others have argued for an approach based on rationality that relies on utility theory [69].

In this framework, we take a neutral stance on such detail by specifying motivation as a *given set*, omitting any further information. This allows us to use the concept of distinct and possibly conflicting motivations influencing the behaviour of the agent, but also defers the choice of the actual mechanism to a subsequent point of refinement or implementation. Moreover, whilst others have been concerned with modelling motivation [68, 67], our work is concerned with its use in defining autonomy.

5.4.2 Autonomous Agent Specification

We begin our specification of autonomous agents by introducing *motivations.*

Definition: A *motivation* is any desire or preference that can lead to the generation and adoption of goals and that affects the outcome of the reasoning or behavioural task intended to satisfy those goals.

As with actions and attributes, the type of all motivations is defined as a given set.

$[Motivation]$

[3]In fact, the distinction between goals and motivations depends on the level at which the environment is modelled and specifically how attributes are represented. For example, if "having more money than anyone else" can be represented as an attribute, then it can be a goal, otherwise it is a motivation. However, incorporating too much detail at the level of attributes does not provide a natural way of modelling, and omitting the overarching concept of motivation obscures an important conceptual distinction. Of more practical concern is that it also does not enable goal adoption to be *grounded* as described above.

Table 1: Example: Descriptions of a Robot and Cup in the Agent Framework

Schema	**Variable**	**Tea − Cup**	**Robot**
Entity	*attributes*	$\{stable, hard, \ldots\}$	$\{red, large, heavy, \ldots\}$
Object	*capabilities*	$\{support, store, \ldots\}$	$\{lift, carry, hold, \ldots\}$
Agent	*goals*	$\{store\_tea\}$	$\{fix\_tyres, charge, \ldots\}$
AutonomousAgent	*motivations*	$\{\,\}$	$\{achievement, hunger, \ldots\}$

An autonomous agent may now be defined.
Definition: An *autonomous agent* is an agent with a non-empty set of motivations.
It is specified simply as an agent with motivations.

```
── AutonomousAgent ──────────────────────────
│ Agent
│ mots : ℙ Motivation
├──────────────
│ mots ≠ { }
└─────────────────────────────────────────────
```

In illustration of these ideas, note that the cup cannot be considered autonomous because, while it can have goals ascribed to it, it cannot *generate* its own goals. In this respect it relies on other entities for purposeful existence. The robot, however, is potentially autonomous in the sense that it may have a mechanism for internal goal generation. Suppose the robot has motivations of achievement, hunger and self-preservation, where achievement is related to attaching tyres onto a car on a production line, hunger is related to maintaining power levels, and self-preservation is related to avoiding system breakdowns. In normal operation, the robot will generate goals to attach tyres to cars through a series of subgoals. If its power levels are low, however, it may replace the goal of attaching tyres with a newly-generated goal of recharging its batteries. A third possibility is that in satisfying its achievement motivation, it works for too long and is in danger of overheating. In this case, the robot can generate a goal of pausing for an appropriate period in order to avoid any damage to its components. Such a robot is autonomous because its goals are not imposed, but are generated in response to its environment. The views of the cup and robot in terms of the agent hierarchy are shown in Table 1, which provides example instantiations of the different requirements for each level.[4]

5.4.3 Autonomous Agent Perception

With autonomous agents, therefore, it is both goals and motivations that are relevant to determining what is perceived in an environment. The schema below thus specifies a modified version of the non-autonomous agent's *agperceives* function as *perceives.* That which an autonomous agent is potentially *capable* of perceiving at any time is independent of its motivations. Indeed, it will always be independent of goals and motivations, and there is consequently no equivalent increase in functionality to *canperceive.*

[4]According to our definition an autonomous agent is an object which has both motivations *and* goals. However, it may be possible to design motivated autonomous agents which, during moments of inactivity, do not have a current goal. For such entities to be described consistently within our framework it would be necessary to introduce a 'null goal" so that they maintain their *agentness.*

$$\begin{array}{l}
\textit{AutonomousAgentPerception} \\
\hline
\textit{AutonomousAgent} \\
\textit{AgentPerception} \\
\textit{perceives} : \mathbb{P}\,\textit{Motivation} \to \mathbb{P}\,\textit{Goal} \\
\qquad\qquad\qquad\qquad \to \textit{Env} \to \textit{View} \\
\hline
\operatorname{dom} \textit{perceives} = \{\textit{mots}\} \\
\hline
\end{array}$$

5.4.4 Autonomous Agent Action

An autonomous agent will have some potential means of evaluating behaviour in terms of the environment and its motivations. In other words, the behaviour of the agent is determined by both external and internal factors. This is qualitatively different from an agent that merely has goals because motivations are non-derivative and governed by internal inaccessible rules, while goals are derivative but relate to motivations. Specifically, the action-selection function for an autonomous agent is produced at every instance by the motivations of the agent. The next schema defines the action-selection function, *act* and includes the *AgentAction* and *AutonomousAgent* schemas. The domain of the *act* function is equal to the motivations of the agent.

$$\begin{array}{l}
\textit{AutonomousAgentAction} \\
\hline
\textit{AutonomousAgent} \\
\textit{AgentAction} \\
\textit{act} : \mathbb{P}\,\textit{Motivation} \to \mathbb{P}\,\textit{Goal} \\
\qquad\qquad \to \textit{View} \to \textit{Env} \to \textit{Actions} \\
\hline
\operatorname{dom} \textit{act} = \{\textit{mots}\} \\
\hline
\end{array}$$

5.4.5 Autonomous Agent State

In exactly the same way that the state of an agent is defined by refining the definition of the state of an object, the state of an autonomous agent is defined using the state of an agent. The actions performed by an autonomous agent are a function of its motivations, goals, percepts and environment.

$$\begin{array}{l}
\textit{AutonomousAgentState} \\
\hline
\textit{AgentState} \\
\textit{AutonomousAgentPerception} \\
\textit{AutonomousAgentAction} \\
\hline
\textit{willdo} = \textit{act mots goals percepts env} \\
\hline
\end{array}$$

5.4.6 Autonomous Agent Operations

In considering the definition of a change in state for an autonomous agent, there are some subtle but important differences with previous schemas. Whereas previously goals were fixed for agents as capabilities were for objects, we do not explicitly state whether motivations change when actions are performed. If they do change, then the agent functions, *agperceives* and *agentact*, will also change. If they do not change, motivations may generate new and different goals for the agent to pursue. In any of these cases, the characterising features of an agent are in flux so that an autonomous agent can be regarded as a continually re-instantiated non-autonomous agent. In this sense, autonomous agents are permanently agents as opposed to transient non-autonomous agents, which may revert to being objects.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\Delta \mathit{AutonomousAgentState}} \\
\mathit{AutonomousAgentState} \\
\mathit{AutonomousAgentState}' \\
\hline
\Delta \mathit{AgentState} \\
\mathit{perceives}' = \mathit{perceives} \\
\mathit{act}' = \mathit{act} \\
\hline
\end{array}
$$

Finally, we specify the operation of an autonomous agent performing its next set of actions, a refinement of the *AgentInteracts* schema.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\mathit{AutonomousAgentInteracts}} \\
\Delta \mathit{AutonomousAgentState} \\
\mathit{AgentInteracts} \\
\hline
\mathit{posspercepts}' = \mathit{canperceive}\ \mathit{env}'\ \mathit{peractions} \\
\mathit{percepts}' = \mathit{perceives}\ \mathit{mots}'\ \mathit{goals}'\ \mathit{posspercepts}' \\
\mathit{willdo}' = \mathit{act}\ \mathit{mots}'\ \mathit{goals}'\ \mathit{percepts}'\ \mathit{env}' \\
\hline
\end{array}
$$

6 Consequences of the Agent Hierarchy

The agent hierarchy specified above is sufficient for satisfying the first requirement of formal frameworks identified earlier. It provides unambiguous definitions and descriptions of different classes of entity in an elegant and readable way, making clear the particular identifying features of each class. Within this framework, we can further refine our concepts to develop more concrete instances of different types of agent, corresponding both to existing agent categories, and to new categories that may be useful and interesting. We can thus consider how the framework impacts on existing notions and classes of agent, and what consequences arise from the definitions. This section addresses such issues, focusing in particular on the agent class descriptions provided by [70] and [71].

6.1 Reflexive Agents

An agent is said to be *reflexive* [70] or *tropistic* [71] if it responds only to an immediate stimulus. Thus a particular perception of the environment dictates which next action the agent will choose to perform. This kind of agent is often viewed as an agent without goals, but that is an over-simplification. Any agent will be designed with some overall goal, even if the goal itself is not explicitly encoded. This is important, for it underlies our definition of agency. The relevant feature of reflexivity is that it has no internal state representing any prior information about the environment. As such, this fits in well with the definition provided above and specified in the *Agent* and *AgentAct* schemas.

6.2 Store Agents

Such agents are, however, extremely limited since without their experience cannot direct behaviour. In order to model their environments or to evaluate competing plans, agents must be able to capture and store information as internal state. To specify an agent with such internal state, or *store*, we can simply refine the *Agent* schema by adding a collection of attributes as *memory*. The schema below defines agents with the ability to access such an internal store of attributes or memory as a *store agent*. Notice that the type of a store is *Env* which, when accessed, produces perceptions of type *View*.

$$
\begin{array}{|l}
\textit{StoreAgent} \\
\hline
\textit{Agent} \\
\textit{store} : \textit{Env} \\
\hline
\textit{store} \neq \{\} \\
\hline
\end{array}
$$

Now, since there is both an external environment and a memory, it is necessary to distinguish between internal and external perceiving actions, where internal actions can access the store, and external perceiving actions access the external environment. We therefore refine the definition of the *canperceive* function at the agent level into two distinct functions at the store-agent level.

In defining perception, *StoreAgentPerception* includes *AgentPerception* and *StoreAgent*, with *intperactions* and *extperactions* referring to the external and internal perceiving actions, respectively. The *storecanperceive* function determines the set of perceptions that can be currently generated from its internal store, while *extcanperceive* determines those percepts possible from the external environment. The internal perceiving actions must be non-empty since otherwise the store cannot be accessed, and the internal and external perceiving actions are disjoint and together comprise the set of all perceiving actions. Note that the percepts that the agent actually selects make up a subset of the available attributes and are dependent on the goals of the agent as defined previously. Thus the *perceive* function from the *AgentPerception* schema is still applicable, since the store is carried through possible percepts and actual percepts to the action-selection function, *agentact*.

$$
\begin{array}{|l}
\textit{StoreAgentPerception} \\
\hline
\textit{StoreAgent} \\
\textit{AgentPerception} \\
\textit{intactions}, \textit{extactions} : \mathbb{P}\,\textit{Action} \\
\textit{storeperceive} : \textit{Env} \to \textit{Actions} \nrightarrow \textit{View} \\
\textit{extperceive} : \textit{Env} \to \textit{Actions} \nrightarrow \textit{View} \\
\hline
\textit{intactions} \neq \{\ \} \\
\textit{intactions} \cup \textit{extactions} = \textit{peractions} \\
\textit{intactions} \cap \textit{extactions} = \{\ \} \\
\operatorname{dom} \textit{storeperceive} = \{\textit{store}\} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{StoreAgentAction} \\
\hline
\textit{StoreAgent} \\
\textit{AgentAction} \\
\hline
\end{array}
$$

In this way, we specify agents that are similar in spirit to the *hysteretic agents* of Genesereth and Nilsson [71], and the *reflex agents with internal state* of Russell and Norvig [70].

6.3 Agent Properties

Now, if we consider the broad characteristics reviewed in the introduction that are often cited as qualities of agents, we can see how they fit in with the framework. These qualities are summarised in

Property	**Agent Type**
reactivity	Agent
pro-activeness	(Planning) Agent
autonomy	Autonomous Agent
rationality	Autonomous Agent
benevolence	Non-autonomous Agent
veracity	Non-autonomous Agent
temporal continuity	*Not required*
personality	*Not required*
adaptability	*Not required*
mobility	*Not required*
communication, social ability	*Not required*

Table 2: Characteristic agent properties.

Table 2, and each will be addressed in turn. Note that each is a specialisation of our basic agent. First, reactivity has been defined as the ability to perceive and to respond to a changing environment. This is always true, since action-selection in all of our entities is a function of the environment at every point. Reactivity is a less pure version of reflexivity [38], but the two are often taken to be the same, and we will not pursue further a discussion of the relative qualities of them here. Pro-activeness, by which an agent behaves in a goal-directed fashion, is a direct consequence of the framework, but is more usually considered in relation to planning, and is strengthened with autonomy, discussed next, in allowing an agent to generate its own goals.

The third property, autonomy, is relatively straightforward in that it is defined explicitly as one level in the agent hierarchy. An autonomous agent is defined by its *motivations* which provide a means for self generation of goals. Rationality is a little less explicit, but arises out of the motivations of the agent. Remember that the details of the motivations are not specified, and that a motivational mechanism can be constructed in any suitable way. Whatever mechanism is adopted, however, goals are generated to mitigate motivations in an essentially rational manner. For example, motivation might amount to a measure of utility, in which case rationality can be defined in terms of maximising utility. The key point is that this notion of motivation provides constraints that guarantee rational behaviour. (For further discussion of this issue, see [67] or [72], for example.)

The property of benevolence — that agents will cooperate with other agents whenever and wherever possible — is more interesting. *B*lind benevolence has no place in modelling autonomous agents for whom cooperation will occur only when it is considered advantageous in terms of motivations to do so. More generally, benevolent behaviour is possible, but will only arise in satisfying a ‘selfish’ motivation. For example, giving to charity arises out of some internal drive such as to help others or have a good self-image; these are selfish in that they satisfy internal needs but are benevolent in that they help others. A non-autonomous agent, however, can be benevolent in that it is instantiated with a particular set of goals that do not change for the duration of the instantiation. It does not generate goals in response to motivations, and can therefore be *designed* for benevolent behaviour. The critical aspect of *blind* benevolence is that it is diametrically opposed to the concept of autonomy. Veracity, or not knowingly providing false information, can be seen as being related to benevolence in that it is not, and cannot be, guaranteed for an autonomous agent for the same reasons, but can arise naturally.

The remaining properties are all less contentious and depend on the *design* of the agent or autonomous agent. That is to say that they do not impact, or relate to, the hierarchy in any significant ways, but can be added on to the basic entities described therein through refinement. Other such refinements are also possible, leading to specifications of *knowledge-level agents* and *stepped knowledge-level agents* [71], for example. These can be constructed in a similar fashion, and we will not consider them further here. The next section does, however address the particular case of *deliberative* or *planning agents* which are especially significant in artificial intelligence for achieving the successful completion of complicated tasks, as well as considering sociological agents which construct models of others, and showing how relationship definitions can be constructed.

7 A Foundation for Further Work

As a result of this specification, we have formal definitions for agents and autonomous agents that are clear, precise and unambiguous, but which do not specify a prescribed internal architecture for agency and autonomy. This is exactly right, since it allows a variety of different architectural and design views to be accommodated within a single unifying structure. All that is required by our specification is a minimal adherence to features of, and relationships between, the entities described therein. Thus we allow a cup to be viewed as an object or an agent depending on the manner in which it functions or is used. Similarly, we allow a robot to be viewed as an object, an agent or an autonomous agent depending on the nature of its control structures. We do not specify here how those control structures should function, but instead how the control is directed.

Moreover, the framework described and specified here is not only intended to stand by itself, but also to provide a base for further development of agent architectures and agent theory in an incremental fashion through refinement and schema inclusion. In this context, the framework has been further developed to incorporate, for example, notions of planning, agent modelling and cooperation. A very brief description of each follows.

7.1 Planning

We can easily refine the components within the framework to provide, for example, a high level specification of a planning agent, which deliberates over sequences of actions that can achieve a desired goal.

As a simple example of the way types can be constructed using only the four given sets as presented in the framework, we can build a definition of a plan from our *Action* type as follows.

First, we define a *complete plan* to be a *sequence* of actions.

$$\mathit{TotalPlan} == \operatorname{seq} \mathit{Action}$$

If the actions in a plan are not completely ordered, then that plan is a *partial plan.* A partial plan thus consists of some partial ordering on a set of actions as shown below. However, an action cannot occur before itself in a partial plan, and if an action a occurs before action b, then b cannot be before a. The notation, R^+, denotes the irreflexive-transitive closure of R.

$$\begin{aligned}
&\mathit{PartialPlan} == \\
&\quad \{ps : \mathit{Action} \leftrightarrow \mathit{Action} \mid \\
&\quad \forall\, a, b : \mathit{Action} \bullet (a, a) \notin ps^{+} \;\wedge \\
&\quad (a, b) \in ps^{+} \Rightarrow (b, a) \notin ps^{+} \bullet ps\}
\end{aligned}$$

A plan, *Plan*, is either a total plan or partial plan. (Other types of plan can be specified similarly.)

$$\begin{aligned} Plan &::= Partial\langle\!\langle PartialPlan\rangle\!\rangle \\ &\mid\ \ Total\langle\!\langle TotalPlan\rangle\!\rangle \end{aligned}$$

Now, consider the next schema which describes how a planning agent can be defined as a refinement of a basic agent by using schema inclusion. A planning agent is an agent with a set of plans associated with a set of goals, where each plan is a possible means of bringing about the associated goal. A subset of these goals are those the agent currently desires; there may also be plans for a goal the agent does not currently desire.

The schema states that all the plans of an agent must be associated with a goal, although it may be that the set of plans associated with a goal is the empty set. Clearly, a plan may also bring about more than one goal of the planning agent.

```
┌─ PlanningAgent ─────────────────────────────────
│ Agent
│ Plans : ℙ Plan
│ planforgoal : Goal ⇸ ℙ Plan
├──────────────
│ goals ⊆ dom planforgoal
│ ⋃(ran planforgoal) = Plans
└─────────────────────────────────────────────────
```

This relatively straightforward refinement of the basic agent model shows how more sophisticated capabilities can be added to the framework to arrive at descriptions of more specific agents. Indeed, such definitions have been used to specify architectures for BDI agents [62, 73], for example.

7.2 Cooperation

In a similar way we can investigate issues of *multi*-agent systems such as the social relationships between agents. For example, two autonomous agents are said to be *cooperating* with respect to some goal if one of the agents has adopted goals of the other. That is to say that the term *cooperation* can be used only when those involved are autonomous and, at least potentially, capable of resisting. If they are not autonomous, nor capable of resisting, then one simply *engages* the other. The difference between engagement and cooperation is in the autonomy or non-autonomy of the entities involved. It is senseless, for example, to consider a terminal cooperating with its user, but meaningful to consider the user engaging the terminal. Similarly, while it is not inconceivable for a user to engage a secretary, it makes better sense to say that the secretary is cooperating with the user, since the secretary can withdraw assistance at any point.

A *cooperation* describes a goal, the autonomous agent that generated the goal, and those autonomous agents who have adopted that goal from the generating agent as a consequence of recognising it in that agent. In this view, cooperation cannot occur unwittingly between agents, but must arise as a result of the motivations of both of the individuals involved. The definition of an *engagement* is defined in a similar fashion.

Cooperation

$goal : Goal$
$genagent : AutonomousAgent$
$coopagents : \mathbb{P}\, AutonomousAgent$

$\# coopagents \geq 1$
$\forall\, c : coopagents \bullet goal \in c.goals$
$goal \in genagent.goals$

Details of the refinement of the framework to address cooperation and other relationships can be found in [72, 74, 75]. This work is particularly illuminating, because these relationships arise naturally out of a formal specification, based on our framework, of an archetypal distributed AI system, the Contract Net Protocol [76, 77]. Thus, the link between theoretical constructs and system descriptions, when structured within the framework presented here, is natural and elegant[78, 79] and can be used to provide a basis for more refined descriptions of autonomous interaction[80].

7.3 Agent Modelling

The framework is suitable for reasoning both *about* entities in the world, and *with* entities in the world. That is to say that an agent itself can also use the entity hierarchy as a basis for reasoning about the functionality of other agents and the likelihood, for example, that they may or may not be predisposed to help in the completion of certain tasks. Thus we can describe agents who are able to model the agents they believe are in their world. Even though the types of these constructs are equivalent to those presented in the framework, it useful to distinguish physical constructs from mental constructs such as models, as it provides a conceptual aid. For example, we can use the definition of an agent to define an agent's model of another agent, and similarly for models of cooperation.

$AgentModel == Agent$
$CoopModel == Cooperation$

It is then possible to describe agents who can model other agents in their environment and, for sufficiently advanced agents, the autonomy of others. Indeed the sophistication of agents can be further increased to define *sociological agents* as those with the ability not only to model the objects, agents and autonomous agents in their world but also the social relationships between them such as cooperation and engagement described above. Specifying such an agent is then just a refinement of the agent schema, using the framework schemas as types within this lower-level definition. The full definition is omitted here.

SociologicalAgent

$Agent$
$agentmodels : \mathbb{P}\, AgentModel$
$coopmodels : \mathbb{P}\, CoopModel$
$\vdots$

The schemas can be extended in this way for more and more sophisticated agents such as those able to model the plans of other agents and so on. In this way the agent hierarchy fulfils the second requirement of formal frameworks as discussed in Section 3.2 in providing a foundation for the subsequent development of more detailed and sophisticated constructs.

8 Conclusions

There exists a small body of work that provides a similar view to that presented here. For example, Covrigaru and Lindsay describe a set of properties that *characterise* autonomous systems to some "degree", relating to such factors as type and number of goals, complexity, interaction, robustness, and so on [81]. In contrast, we *define* what is necessary for a system to be autonomous in very precise terms, and we distinguish clearly between objectness, agency and autonomy. One particular consequence of the difference in views is that we allow a rock, for example, to be considered an agent *if* it is being used for some purpose, such as a hammer for tent-pegs. Covrigaru and Lindsay deny the rock the quality of autonomy because it is not goal-directed, but ignore the possibility of agency, skipping over an important part of our framework. Other work includes that by Tokoro who offers a related view in which he distinguishes objects, concurrent objects, autonomous agents and volitional agents, similar in spirit to our own view [82]. In addition, Castelfranchi also characterises autonomy through the use of motivation [83]. Our work differs in that we take autonomy to be an absolute concept which is constant regardless of the context in which it occurs. It either exists or it does not.

We have constructed a formal specification which identifies and characterises those entities that are called agents and autonomous agents. The work is not based on any existing classifications or notions because there is no consensus. Recent papers define agents in wildly different ways, if at all, and this makes it extremely difficult to be explicit about their nature and functionality.

The taxonomy described by the framework satisfies each of the requirements identified in the introduction, demonstrated over the course of the paper. First, it provides clear and precise definitions for objects, agents and autonomous agents that allow a better understanding of the functionality of different systems. It explicates those factors that are necessary for agency and autonomy, and is sufficiently abstract to cover the gamut of agents, both hardware and software, intelligent and unintelligent, and so on. This abstraction allows the framework to satisfy the remaining requirements, the second of which demands that it provides a foundation for subsequent development of more refined concepts. The previous section has shown how the framework can be used in just such a way. Finally, it must also enable alternative designs to be explicitly presented, compared and evaluated, and this, too, has been briefly shown in the previous sections by comparing different qualities of agents, and then minimally specifying related designs.

Z has thus enabled us to produce a specification that is generally accessible to researchers in AI and software engineers, as well as practitioners of formal methods. Through the use of schema inclusion, we are able to describe our framework at the highest level of abstraction and then, by incrementally increasing the detail in the specification, we add system complexity at appropriate levels. Our use of Z does not restrict us to any particular mathematical model, but instead provides a general mathematical framework within which different models, and even particular systems, can be defined and contrasted.

In particular, the nature of Z allows us to extend the framework and to refine it further to include a more varied and more inclusive set of concepts. The examples of planning, models and cooperation we have presented, outline how the original framework was extended through new schemas and schema inclusion, and indicate both how we intend to proceed in this respect, and also how appropriate Z is for this task. It enables a practitioner to choose the level of detail required to present a particular design and, furthermore, provides an environment in which the design itself can be presented in increasing levels of detail.

Acknowledgements

Thanks to John Campbell, Michael Georgeff, Jennifer Goodwin, Paul Howells, Nick Jennings, Colin Myers, Mark Priestley, John Wolstencroft and Mike Wooldridge for comments on earlier versions of this paper. This paper has been checked for type-correctness by the *fuzz* package.

References

[1] Luck, M. (1999) From definition to deployment: What next for agent-based systems? *The Knowledge Engineering Review*, 14(2):119–124.

[2] Orfali, R. and Harkey, D. (1998) *Client/Server Programming with Java and CORBA*. John Wiley & Sons, New York, NY. second edition.

[3] Chaib-draa, B. (1995) Industrial applications of distributed AI. *Communications of the ACM*, 38(11):49–53.

[4] Crabtree, B. (1998) What chance software agents? *The Knowledge Engineering Review*, 13(2):131–136.

[5] Van Dyke Parunak, H. (1996) Applications of distributed artificial intelligence in industry. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 139–164. John Wiley & Sons, New York, NY.

[6] Van Dyke Parunak, H. (1998) What can agents do in industry, and why? An overview of industrially-oriented R&D at CEC. In M. Klusch and G. Weiss, editors, *Cooperative Information Agents II, Lecture Notes in Artificial Intelligence 1435*, pages 1–18. Springer-Verlag, Berlin.

[7] Johnson, L. and Hayes-Roth, B. (eds) (1997) *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA. ACM Press, New York, NY.

[8] Genesereth, M. and Ketchpel, S. (1994) Software agents. *Communications of the ACM*, 37(7):48–53.

[9] Wooldridge, M. and Jennings, N. (1995) Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2).

[10] Lashkari, Y., Metral, M. and Maes, P. (1994) Collaborative interface agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, July 1994, pages 444–449. AAAI Press, Menlo Park CA.

[11] Aylett, R. and Luck, M. (2000) Applying artificial intelligence to virtual reality: Intelligent virtual environments. *Applied Artificial Intelligence*, 14(1):3–32.

[12] Kuokka, D. and Harada, L. (1995) Matchmaking for information agents. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 672–679, Montréal, Québec, Canada. Morgan Kaufmann, San Mateo, CA.

[13] Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C. and Tsudik, G. (1995) Itinerant agents for mobile computing. *IEEE Personal Communications*, 2(5):34–49.

[14] Wong, D., Paciorek, N. and Moore, D. (1999) Java-based mobile agents. *Communications of the ACM*, 42(3):92–102.

[15] Etzioni, O., Levy, H., Segal, R. and Thekkath, C. (1995) The softbot approach to OS interfaces. *IEEE Software*, 12(4).

[16] Toomey, C. and Mark, W. (1995) Satellite image dissemination via software agents. *IEEE Expert*, 10(5):44–51.

[17] Jennings, N. and Wittig, T. (1992) ARCHON: Theory and practice. In N.M. Avouris and L. Gasser, editors, *Distributed Artificial Intelligence: Theory and Praxis*, pages 179–195. Kluwer Academic Press, Dordrecht.

[18] Kinny, D., Georgeff, M. and Rao, A. (1996) A methodology and modelling technique for systems of BDI agents. In Y. Demazeau and J.-P. Müller, 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.

[19] Jennings, N., Faratin, P., Johnson, M., O'Brien, P. and Wiegand, M. (1996) Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2 & 3):105–130.

[20] Guttman, R., Moukas, A. and Maes, P. (1998) Agent-mediated electronic commerce: a survey. *The Knowledge Engineering Review*, 13(2):147–159.

[21] Grand, S. and Cliff, D. (1998) Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems*, 1(1):39–57.

[22] Shoham, Y. (1993) Agent-oriented programming. *Artificial Intelligence*, 60:51–92.

[23] Smith, D., Cypher, A. and Spohrer, J. (1994) Programming agents without a programming language. *Communications of the ACM*, 37(7):55–67.

[24] Selker, T. (1994) A teaching agent that learns. *Communications of the ACM*, 37(7):92–99.

[25] Riecken, D. (1994) An architecture of integrated agents. *Communications of the ACM*, 37(7):107–116.

[26] Etzioni, O. and Weld, D. (1995) Intelligent agents on the internet: Fact, fiction and forecast. *IEEE Expert*, 10(4):44–49.

[27] Hedberg, S. (1995) Intelligent agents: The first harvest of softbots looks promising. *IEEE Expert*, 10(4):6–9.

[28] Franklin, S. and Graesser, A. (1997) 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, Lecture Notes in Artificial Intelligence, 1193*, pages 21–35. Springer-Verlag.

[29] Jennings, N., Sycara, K. and Wooldridge, M. (1998) A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38.

[30] Müller, J. P. (1998) Architectures and applications of intelligent agents: A survey. *The Knowledge Engineering Review*, 13(4):353–380.

[31] Petrie, C. (1997) What is an agent? 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, Lecture Notes in Artificial Intelligence, 1193,* pages 41–43. Springer-Verlag.

[32] Nwana, H. (1996) Software agents: an overview. *The Knowledge Engineering Review*, 11(3):205–244.

[33] Wooldridge, M. (1997) Agents as a Rorschach test: A response to Franklin and Graesser. 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, Lecture Notes in Artificial Intelligence, 1193*, pages 47–48. Springer-Verlag.

[34] Luck, M. and d'Inverno, M. (1995) 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.

[35] Wooldridge, M. and Jennings, N. (1999) Cooperative problem solving. *Journal of Logic and Computation*, 9(4):563–592.

[36] Rao, A. and Georgeff, M. (1992) An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning,* Cambridge, MA, October 1992, pages 439–449. Morgan Kaufmann.

[37] Rao, A. (1996) 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.

[38] Goodwin, R. (1995) A formal specification of agent properties. *Journal of Logic and Computation*, 5(6):763–781.

[39] Good, D. and Young, W. (1991) Mathematical methods for digital systems development. In S. Prehn and W. J. Toetenel, editors, *VDM '91: Formal Software Development Methods, Proceedings of the Fourth International Symposium of VDM Europe, LNCS, 552,* volume 2, pages 406–430. Springer-Verlag.

[40] Krause, P., Fox, J., O'Neill, M. and Glowinski, A. (1993) Can we formally specify a medical decision support system. *IEEE Expert*, 8(3):56–61.

[41] Craig, I. (1991) *The Formal Specification of Advanced AI Architectures*. Ellis Horwood, Chichester.

[42] Craig, I. (1994) The formal specification of ELEKTRA. Research Report RR261, Department of Computer Science, University of Warwick.

[43] Wooldridge, M. (1995) This is MYWORLD: The logic of an agent-oriented DAI testbed. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages, LNAI 890*, pages 160–178. Springer-Verlag.

[44] Milnes, B. (1992) A specification of the Soar architecture in Z. Technical Report CMU-CS-92-169, School of Computer Science, Carnegie Mellon University.

[45] Fensel, D. (1995) Formal specification languages in knowledge and software engineering. *The Knowledge Engineering Review*, 10(4):361–404.

[46] Fensel, D. and van Harmelen, F. (1994) A comparison of languages which operationalise and formalise KADS models of expertise. *The Knowledge Engineering Review*, 9:105–146.

[47] van Harmelen, F. and Fensel, D. (1995) Formal methods in knowledge engineering. *The Knowledge Engineering Review*, 10(4):345–360.

[48] Garlan, D. and Notkin, D. (1991) Formalizing design spaces: Implicit invocation mechanisms. In S. Prehn and W. J. Toetenel, editors, *VDM '91: Formal Software Development Methods, Proceedings of the Fourth International Symposium of VDM Europe, LNCS, 551*, volume 1, pages 31–44. Springer-Verlag.

[49] d'Inverno, M., Justo, G. and Howells, P. (1996) A formal framework for specifying design methodologies. *Software Process: Improvement and Practice*, 2(3):181–195.

[50] d'Inverno, M., Priestley, M. and Luck, M. (1997) A formal framework for hypertext systems. *IEE Proceedings on Software Engineering*, 144(3):175–184.

[51] d'Inverno, M. and Hu, M. (1997) A Z specification of the soft-link hypertext model. In M. G. Hinchey, editor, *ZUM'97: The Z Formal Specification Notation, 10th International Conference of Z Users, Lecture Notes in Computer Science*, pages 297–316.

[52] Garlan, D. (1990) The role of formal reusable frameworks. *ACM SIGSOFT: Software Engineering Notes*, 15(4):42–44.

[53] Spivey, J. (1992) *The Z Notation: A Reference Manual.* Prentice Hall, Hemel Hempstead, 2nd edition.

[54] Bowen, J. (1996) *Formal Specification and Documentation using Z: A Case Study Approach*. International Thomson Computer Press, London.

[55] Hayes, I. (ed) (1993) *Specification Case Studies.* Prentice Hall, second edition.

[56] Spivey, J. (1988) *Understanding Z: A Specification Language and its Formal Semantics.* Cambridge University Press.

[57] Bowen, J., Fett, S. and Hinchey, M. (eds) (1998) *ZUM'98: The Z Formal Specification Notation, 11th International Conference of Z Users, Lecture Notes in Computer Science, 1493.* Springer-Verlag.

[58] Bowen, J. and Hall, J. (eds) (1994) *Z User Workshop: Proceedings of the Eighth Z User Meeting*, Cambridge, June 1994. Springer-Verlag, London

[59] Bowen, J. and Hinchey, M. (eds) (1995) *ZUM'95: The Z Formal Specification Notation, 9th International Conference of Z Users, Lecture Notes in Computer Science, 967.* Springer-Verlag.

[60] Bowen, J., Hinchey, M. and Till, D. (eds) (1997) *ZUM'97: The Z Formal Specification Notation, 10th International Conference of Z Users, Lecture Notes in Computer Science, 1212.* Springer-Verlag.

[61] Diller, A. (1994) *Z: An Introduction to Formal Methods.* John Wiley & Sons, Chichester, second edition.

[62] d'Inverno, M., Kinny, D., Luck, M. and Wooldridge, M. (1998) A formal specification of dMARS. In *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages*, pages 155–176. Springer-Verlag, 1365.

[63] Kunda, Z. (1990) The case for motivated reasoning. *Psychological Bulletin*, 108(3):480–498.

[64] Simon, H. (1979) Motivational and emotional controls of cognition. In *Models of Thought*, pages 29–38. Yale University Press.

[65] Sloman, A. (1987) Motives, mechanisms, and emotions. *Cognition and Emotion*, 1(3):217–233.

[66] Sloman, A. and Croucher, M. (1981) Why robots will have emotions. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 197–202, Vancouver, B.C. William Kaufmann, Los Altos, CA.

[67] Norman, T. and Long, D. (1995) 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.

[68] Moffat, D. and Frijda, N. (1995) 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.

[69] Russell, S., Subramanian, D. and Parr, R. (1993) Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chamberry, September 1993, pages 338–344. Morgan Kaufmann, San Mateo, CA.

[70] Russell, S. and Norvig, P. (1995) *Artificial Intelligence: a modern approach.* Prentice-Hall, Upper Saddle River, NJ.

[71] Genesereth, M. and Nilsson, N. (1987) *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Palo Alta, CA.

[72] Luck, M. and d'Inverno, M. (1996) Engagement and cooperation in motivated agent modelling. In C. Zhang and D. Lukose, editors, *Distributed Artificial Intelligence Architecture and Modelling: Proceedings of the First Australian Workshop on Distributed Artificial Intelligence, Lecture Notes in Artificial Intelligence, 1087*, pages 70–84. Springer-Verlag.

[73] d'Inverno, M. and Luck, M. (1998) Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):233–260.

[74] d'Inverno, M. and Luck, M. (1997) Making and breaking engagements: An operational analysis of agent relationships. In C. Zhang and D. Lukose, editors, *Multi-Agent Systems Methodologies and Applications: Proceedings of the Second Australian Workshop on Distributed Artificial Intelligence, Lecture Notes in Artificial Intelligence, 1286*, pages 48–62. Springer-Verlag.

[75] Luck, M. and d'Inverno, M. (1998) Motivated behaviour for goal adoption. In C. Zhang and 1544 D. Lukose, editors, *Multi-Agent Systems: Theories, Languages and Applications — Proceedings of the Fourth Australian Workshop on Distributed Artificial Intelligence, Lecture Notes in Artificial Intelligence*, pages 58–73. Springer-Verlag.

[76] Smith, R. (1980) The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113.

[77] Smith, R. and Davis, R. (1981) Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):61–70.

[78] d'Inverno, M. and Luck, M. (1996) 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, Lecture Notes in Artificial Intelligence, 1038*, pages 72–85. Springer-Verlag.

[79] Luck, M., Griffiths, N. and d'Inverno, M. (1997) From agent theory to agent construction: A case study. In *Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Lecture Notes in Artificial Intelligence, 1193*, pages 49–63. Springer-Verlag.

[80] d'Inverno, M. and Luck, M. (1996) Understanding autonomous interaction. In W. Wahlster, editor, *ECAI'96 - Proceedings of the 13th European Conference on Artificial Intelligence*, Budapest, August 1996, pages 529–533. John Wiley & Sons, Chichester.

[81] Covrigaru, A. and Lindsay, R. (1991) Deterministic autonomous systems. *AI Magazine*, 12(3):110–117.

[82] Tokoro, M. (1993) The society of objects. Technical Report SCSL-TR-93-018, Sony CSL.

[83] Castelfranchi, C. (1995) Guarantees for autonomy in cognitive agent architecture. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages, LNAI 890*, pages 56–70. Springer-Verlag.