Formal Agent Development: Framework to System

Mark d'Inverno[†] Michael Luck[*]

[†] Cavendish School of Computer Science, University of Westminster, London W1M 8JS, UK
dinverm@westminster.ac.uk
[*] Department of Computer Science, University of Warwick, CV4 7AL, UK
mikeluck@dcs.warwick.ac.uk

Abstract. Much work in the field of agent-based systems has tended to focus on either the development of practical applications of agent systems on the one hand, or the development of sophisticated logics for reasoning about agent systems on the other. Our own view is that work on formal models of agent-based systems are valuable inasmuch as they contribute to a fundamental goal of computing of practical agent development. In an ongoing project that has been running for several years, we have sought to do exactly that through the development of a formal framework that provides a conceptual infrastructure for the analysis and modelling of agents and multi-agent systems on the one hand, and enables implemented and deployed systems to be evaluated and compared on the other. In this paper, we describe our research programme, review its achievements to date, and suggest directions for the future.

1 Introduction

Over the course of the last ten years or so, work in agent-based systems has made dramatic progress. Indeed, there is now a relatively coherent and recognisable field of research that has experienced remarkable growth, both in the quantity of effort devoted by individual researchers and groups, and in the areas within it that have been subject to these investigations. Similarly, commercial development efforts have been increasing, with significant resources targetted at the area. It is a natural consequence of this rapid rise that there should be a range of approaches applied to the field, including both formal and practical (or empirical) methods. Yet in the rush to contribute to its development, the field of agent-based systems has experienced a fragmentation along the lines of a theoretical-practical divide, by which these two semi-distinct threads of research are advancing in parallel rather than interacting constructively.

As has been discussed elsewhere [10], much work has tended to focus on either the development of practical applications of agent systems on the one hand, or the development of sophisticated logics for reasoning about agent systems on the other. Certainly, both of these strands of research are important, but it is crucial for there to be a significant area of overlap between them for cross-fertilisation and for one strand to inform the other. Unfortunately, however, there has been a sizable gap between these formal models and implemented systems.

Our own view is that work on formal models of agent-based systems are valuable inasmuch as they contribute to a fundamental goal of computing of building real agent

systems. This is not to trivialise or denigrate the effort of formal approaches, but to direct it towards integration in a broader research programme. In an ongoing project that has been running for several years, we have sought to do exactly that through the development of a formal framework that provides a conceptual infrastructure for the analysis and modelling of agents and multi-agent systems on the one hand, and enables implemented and deployed systems to be evaluated and compared on the other. In this paper, we describe our research programme, review its achievements to date, and suggest directions for the future. After briefly considering some related work on marrying the formal and practical, we review our earlier work on a formal agent framework and show how it leads to a detailed map of agent relationships. Then we examine plans in more detail, first at an abstract level, and then adding increasingly more detail to arrive at a specification of plans in dMARS. The paper ends by considering the value of this work in moving from abstract agent specification to detailed system description.

2 Background

Though the fragmentation into theoretical and practical aspects has been noted, and several efforts made in attempting to address this fragmentation in related areas of agent-oriented systems by, for example, Goodwin [9], Luck et al. [14], and Wooldridge and Jennings [19], much remains to be done in bringing together the two strands of work.

This section draws on Luck's outline [10] of the ways in which some progress has been made with BDI agents, a well-known and effective agent architecture. Rao, in particular, has attempted to unite BDI 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 [17]. Second, he took an alternative approach by starting with an implemented system and then formalizing the operational semantics in an agent language, AgentSpeak(L), which can be viewed as an abstraction of the implemented system, and which allows agent programs to be written and interpreted [16].

In contrast to this approach, some work aims at constructing directly executable formal models. For example, Fisher's work on Concurrent MetateM [7] has attempted to use temporal logic to represent individual agent behaviours where the representations can either be executed directly, verified with respect to a logical requirement, or transformed into a more refined representation. Further work aims to use this to produce a full development framework from a single high-level agent to a cooperating multi-agent system. In a similar vein, Parsons et al. [15] aim to address the gap between specification and implementation of agent architectures by viewing an agent as a multi-context system in which each architectural component is represented as a separate unit, an encapsulated set of axioms, and an associated deductive mechanism whose interrelationships are specified using bridge rules. Since theorem-provers already exist for multi-context systems, agents specified in this way can also be directly excecuted.

As yet, the body of work aimed at bridging the gap between theory and practice is small. Fortunately, though, there seems to be a general recognition that one of the key roles of theoretical and practical work is to inform the other [2], and while this is made difficult by the almost breakneck pace of progress in the agent field, that recognition

bodes well for the future. Some sceptics remain, however, such as Nwana, who followed Russell in warning against *premature mathematization*, and the danger that lies in wait for agent research [1].

3 Overview

As stated above, we view our enterprise as that of building programs. In order to do so, however, we need to consider issues at different points along what we call the *agent development line*, identifying the various foci of research in agent-based systems in support of final deployment, as shown in Figure 1. To date, our work has concentrated on the first three of the stages identified.

- We have provided a formal agent framework within which we can explore some fundamental questions relating to agent architectures, configurations of multi-agent systems, inter-agent relationships, and so on, independent of any particular model. The framework continues to be extended to cover a broader range of issues, and to provide a more complete and coherent conceptual infrastructure.
- In contrast to starting with an abstract framework and refining it down to particular system implementations, we have also attempted to start with specific deployed systems and provide formal analyses of them. In this way, we seek to move backwards to link the system specifications to the conceptual formal framework, and also to provide a means of comparing and evaluating competing agent systems.
- The third strand aims to investigate the process of moving from the abstract to the concrete, through the construction of agent development methodology, an area that has begun to receive increasing attention. In this way, we hope to marry the value of formal analysis with the imperative of systems development in a coherent fashion, leading naturally to the final stage of the development line, to *agent deployment*.

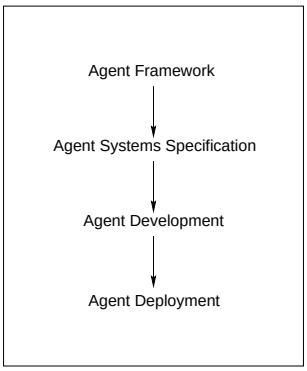

Fig. 1. The Agent Development Line

This paper can be seen as a continuation of the work contained in [5], which describes the research programme at an earlier stage of development. That work intro-

duced requirements for formal frameworks, and showed how our agent framework satisfied those requirements in relation to, in particular, goals generation and adoption, some initial inter-agent relationships, and their application to the Contract Net protocol. In this paper, we build on that work, showing further levels of analysis of agent relationships, and also describe further work on formal agent specification.

In what follows, we use the Z specification language [18], for reasons of accessibility, clarity and existing use in software development. The arguments are well-rehearsed and can be found in numerous of the references. Similarly, we assume familiarity with the notation, details of which can also be found in many of the references, and especially in [18]. The specification in this paper is not intended to be complete, nor to provide the most coherent exposition of a particular piece of work, but to show how a broad research programme in support of the aims above is progressing. Details of the different threads of work may be found in the references in each of the relevant sections.

4 The Formal Agent Framework

We begin by briefly reviewing earlier work. In short, we propose a four-tiered hierarchy comprising *entities*, *objects*, *agents* and *autonomous agents* [11]. The basic idea underlying this hierarchy is that all components of the world are entities. Of these entities, some are objects, of which some, in turn, are agents and of these, some are autonomous agents, as shown in Figure 2.

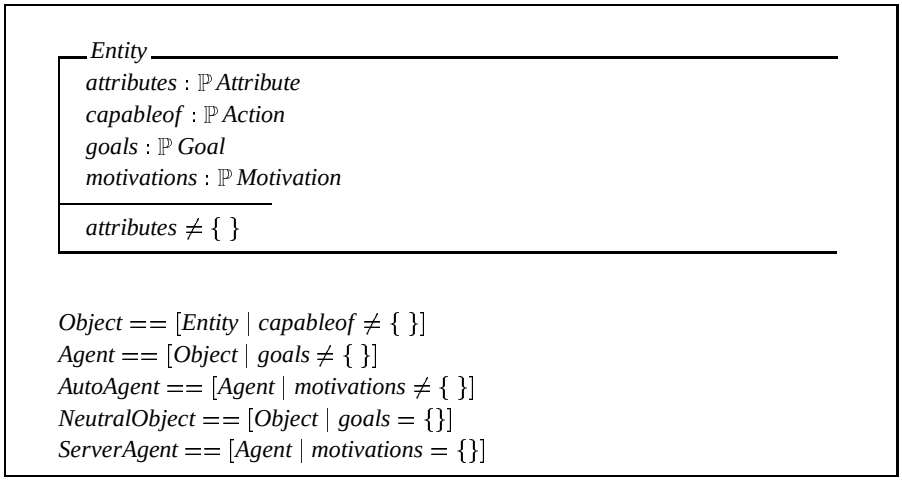

Fig. 2. Entities in the Agent Framework

Entities serve as a useful abstraction mechanism by which they are regarded as distinct from the remainder of the environment, to organise perception. An object is then an entity with abilities which can affect environments in which it is situated. An agent is just an object either that is useful to another agent where this usefulness is

defined in terms of satisfying that agent's goals, or that exhibits independent purposeful behaviour. In other words, an agent is an object with an associated set of goals. This definition of agents relies upon the existence of such other agents which provide the goals that are adopted instantiate an agent. In order to escape an infinite regress of goal adoption, we define autonomous agents, which are agents that generate their own goals from motivations. We also distinguish those objects that are not agents, and those agents that are not autonomous and refer to them as *neutral-objects* and *server-agents* respectively. An agent is then either a server-agent or an autonomous agent, and an object is either a neutral-object or an agent.

5 Inter-Agent Relationships

Agents and autonomous agents are thus defined in terms of goals. Agents *satisfy* goals, while autonomous agents may, additionally, generate them. Goals may be adopted by either autonomous agents, non-autonomous agents or objects without goals. Since non-autonomous agents satisfy goals for others they *rely* on other agents for purposeful existence, indicating that goal adoption creates critical inter-agent relationships.

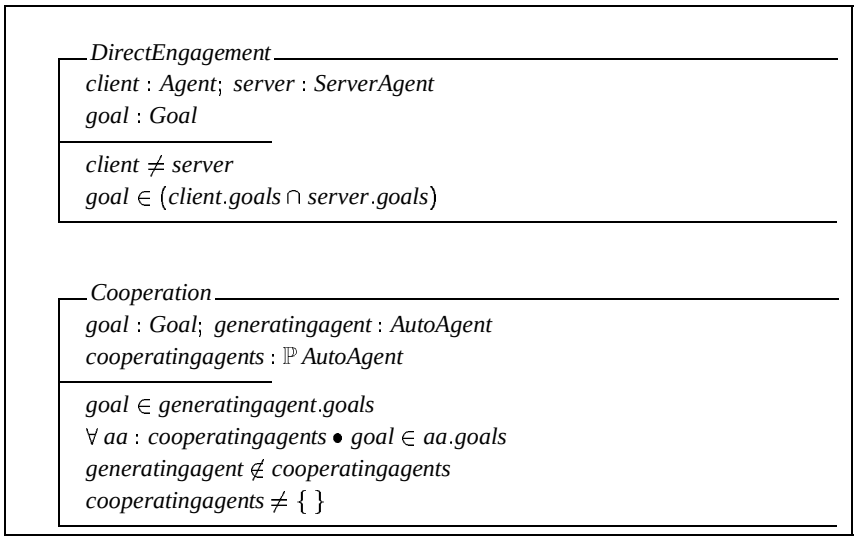

Fig. 3. Engagement and Cooperation

Key relationships are direct engagements, engagement chains and cooperations [12]. In a direct engagement, a *client*-agent with some goals uses another *server*-agent to assist them in the achievement of those goals. A server-agent either exists already as a result of some other engagement, or is instantiated from a neutral-object for the current engagement. (No restriction is placed on a client-agent.) We define a *direct engagement* in Figure 3 to consist of a client agent, *client*, a server agent, *server*, and the goal that

server is satisfying for *client*. An agent cannot engage itself, and both agents must have the goal of the engagement. Once autonomous agents have generated goals and engaged other server-agents, the server-agents may, in turn, engage other non-autonomous entities with the purpose of achieving or pursuing the original goal. These *engagement chains* provide more information with which to analyse multi-agent systems than using engagements alone, since the flow of goal adoption is explicitly represented. They represent the goal and all the agents involved in the sequence of direct engagements. Since goals are grounded by motivations, the agent at the head of the chain must be autonomous. (This is not specified here.) By contrast, two autonomous agents are said to be *cooperating* with respect to some goal if one of the agents has adopted goals of the other.

The combined total of direct engagements, engagement chains and cooperations (*denegagements*, *engchains* and *cooperations*) defines a social organisation that is not artificially or externally imposed but arises as a natural and elegant consequence of our definitions of agents and autonomous agents. Thus the agent framework allows an explicit and precise analysis of multi-agent systems with no more conceptual primitives than were introduced for the initial framework to describe individual agents. Using these fundamental forms of interaction, we can proceed to define a more detailed taxonomy of inter-agent relationships that allows a richer understanding of the social configuration of agents, suggesting different possibilities for interaction, as shown in Figure 4, taken from [13]. Importantly, the relationships identified are not imposed on multi-agent systems, but arise naturally from agents interacting, and therefore underlie all multi-agent systems.

The direct engagement relationship specifies the situation in which there is a direct engagement for which the first agent is the client and the second agent is the server. In general, however, any agent involved in an engagement chain engages all those agents that appear subsequently in the chain. To distinguish engagements involving an intermediate agent we introduce the indirect engagement relation *indengages*; an agent *indirectly* engages another if it engages it, but does not *directly* engage it. If many agents directly engage the same entity, then no single agent has complete control over it. It is important to understand *when* the behaviour of an engaged entity can be modified without any deleterious effect (such as when no other agent uses the entity for a *different* purpose). In this case we say that the agent *owns* the entity. An agent, *c*, owns another agent, *s*, if, for every sequence of server-agents in an engagement chain, *ec*, in which *s* appears, *c* precedes it, or *c* is the autonomous client-agent that initiated the chain. An agent *directly owns* another if it owns it and directly engages it. We can further distinguish the *uniquely owns* relation, which holds when an agent *directly* and *solely* owns another, and *specifically owns*, which holds when it owns it, and has only one goal.

6 Sociological Behaviour

Social behaviour involves an agent interacting with others; sociological behaviour requires more, that an agent understand its relationships with others. In order to do so, it must model them, their relationships, and their plans.

directly engages	$\forall c : Agent;\ s : ServerAgent \bullet (c, s) \in dengages \Leftrightarrow$ $\exists d : dengagements \bullet d.client = c \wedge d.server = s$
engages	$\forall c : Agent, s : ServerAgent \bullet (c, s) \in engages \Leftrightarrow$ $\exists ec : engchains \bullet (s \in (\mathrm{ran}\, ec.agentchain) \wedge c = ec.autoagent) \vee$ $((c, s), ec.agentchain) \in before)$
indirectly engages	$indengages = engages \setminus dengages$
owns	$\forall c : Agent;\ s : ServerAgent \bullet (c, s) \in owns \Leftrightarrow$ $(\forall ec : engchains \mid s \in \mathrm{ran}\, ec.agentchain \bullet$ $ec.autoagent = c \ \vee\ ((c, s), ec.agentchain) \in before)$
directly owns	$downs = owns \cap dengages$
uniquely owns	$\forall c : Agent;\ s : ServerAgent \bullet (c, s) \in uowns \Leftrightarrow$ $(c, s) \in downs \wedge \neg (\exists a : Agent \mid a \neq c \bullet (a, s) \in engages)$
specifically owns	$\forall c : Agent;\ s : ServerAgent \bullet (c, s) \in sowns \Leftrightarrow$ $(c, s) \in owns \wedge \#(s.goals) = 1$

Fig. 4. Inter-agent Relationships

6.1 Models and Plans

To model their environment, agents require an *internal store*, without which their past experience could not direct behaviour, resulting in reflexive action alone. A store exists as part of an agent's state in an environment but it must also have existed *prior* to that state. We call this feature an *internal store* or *memory*, and define *store agents* as those with memories. Unlike *social* agents that engage in interaction with others, *sociological* agents model relationships as well as agents. It is a simple matter to define the model an agent has of another agent (*AgentModel*), by re-using the agent framework components as shown in Figure 5. Even though the types of these constructs are equivalent to those presented earlier, it is useful to distinguish physical constructs from mental constructs such as models, as it provides a conceptual aid. We can similarly define models of other components and relationships so that specifying a sociological agent amounts to a refinement of the *Agent* schema as outlined in Figure 5.

Now, in order to consider sociological agents, or agents that can model others, we can construct a high-level model of *plan-agents* that applies equally well to reactive or deliberative, single-agent or multi-agent, planners. It represents a high-level of abstraction without committing to the nature of an agent, the plan representation, or of the agent's environment; we simply distinguish *categories* of plan and possible relationships between an agent's plans and goals. Specifically, we define *active* plans as those identified as candidate plans not yet selected for execution; and *executable* plans as those active plans that have been selected for execution.

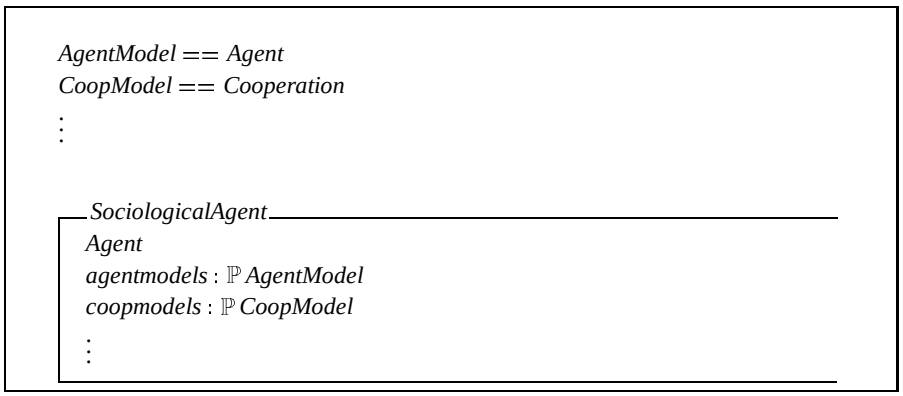

Fig. 5. Sociological Agent

Formally, we initially define the set of all agent plans to be a given set ($[Plan]$), so that at this stage we abstract out any information about the nature of plans themselves. Our highest-level description of a *plan-agent* can then be formalised in the *PlanAgent* schema of Figure 6.

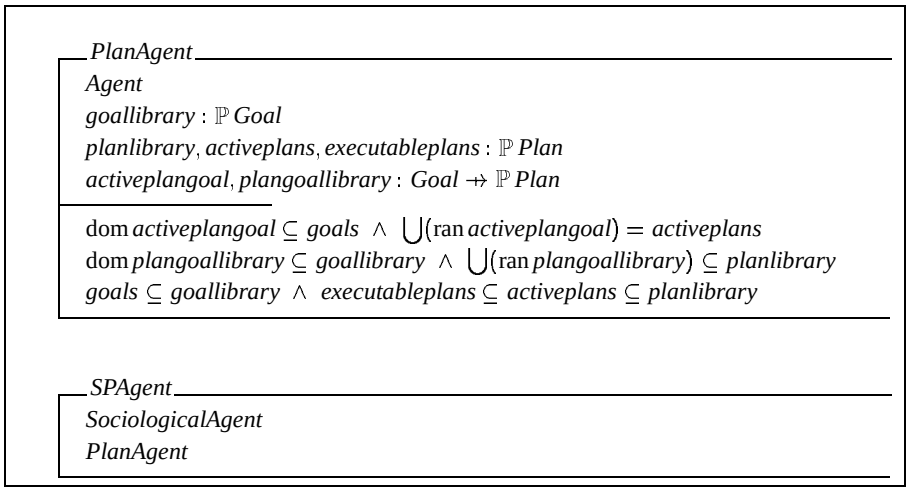

Fig. 6. Sociological Plan Agent

The variables *goallibrary*, *planlibrary*, *activeplans* and *executableplans* represent the agent's repository of goals, repository of plans, active plans and executable plans, respectively. Each active plan is necessarily associated with one or more of the agent's current goals as specified by *activeplangoal*. For example, if the function contains the pair $(g, \{p_1, p_2, p_3\})$, it indicates that p_1, p_2 and p_3 are competing active plans for g. Whilst active plans must be associated with at least one active goal the converse is not

true, since agents may have goals for which no plans have been considered. Analogously the *plangoallibrary* function relates the repository of goals, *goallibrary*, to the repository of plans, *planlibrary*. However, not necessarily all library plans and goals are related by this function.

6.2 Plan and Agent Categories

Now, using these notions, we can describe some example categories of goals, agents and plans (with respect to the models of the sociological plan-agent), that may be relevant to an agent's understanding of its environment. Each of the categories is formally defined in Figure 7, in which the sociological plan-agent is denoted as *spa*. Any variable preceded by *model* denotes the models that *spa* has of some specific type of entity or relationship. For example, *spa.modelneutralobjects* and *spa.modelowns* are the neutral objects and ownership relations the sociological agent models.

A *self-sufficient plan* is any plan that involves only neutral-objects, server-agents the plan-agent owns, and the plan-agent itself. Self-sufficient plans can therefore be executed without regard to other agents, and exploit current agent relationships. (The formal definition makes use of the relational image operator: in general, the relational image $R(\!| S |\!)$ of a set S through a relation R is the set of all objects y to which R relates to some member x of S.) A *self-sufficient goal* is any goal in the goal library that has an associated self-sufficient plan. These goals can then, according to the agent's model, be achieved independently of the existing social configuration. A *reliant-goal* is any goal that has a non-empty set of associated plans that is not self-sufficient.

self-suff plan

$$\forall\, p \in spa.planlibrary \bullet selfsuff(p) \Leftrightarrow spa.planentities(p) \subseteq spa.modelneutralobjects \cup spa.modelself \cup spa.modelowns(\!| spa.modelself |\!) \bullet p$$

self-suff goal

$$\forall\, g \in spa.goallibrary \bullet selfsuffgoal(g) \Leftrightarrow (\exists\, p \in spa.plangoallibrary(g) \bullet p \in selfsuff)$$

reliant goal

$$\forall\, g \in spa.goallibrary \bullet reliantgoal(g) \Leftrightarrow spa.plangoallibrary\, g \neq \{\} \;\wedge\; \neg\, (\exists\, p : spa.plangoallibrary\, g \bullet p \in selfsuff)$$

Fig. 7. Sociological Agent Categories I

For each plan that is not self-sufficient, a sociological plan-agent can establish the autonomous agents that may be affected by its execution, which is an important criterion in selecting a plan from competing active plans. An autonomous agent *A* may be affected by a plan in one of two ways: either it is required to perform an action directly, or it is engaging a server-agent *S* required by the plan. In this latter case, a sociological plan-agent can reason about either persuading *A* to share or release *S*, taking *S* without

permission, or finding an alternative server-agent or plan. To facilitate such an analysis, we can define further categories of agents and plans, as described in [13],but we do not consider them further here.

7 Agent Plans

Now, the rather abstract notion of plans above can be refined further, and we can develop a more detailed specification that will be useful when considering different forms of agents and planning abilities. We begin by introducing some general concepts, and then move to a detailed description of plans in dMARS.

7.1 Modelling Plans

This involves defining first the *components* of a plan, and then the *structure* of a plan, as shown in Figure 8. The components, which we call *plan-actions*, each consist of a *composite-action* and a set of related entities as described below. The structure of plans defines the relationship of the component plan-actions to one another. For example, plans may be *total* and define a sequence of plan-actions, *partial* and place a partial order on the performance of plan-actions, or *trees* and, for example, allow choice between alternative plan-actions at every stage in the plan's execution.

$$
\begin{array}{l}
Primitive == Action \\
Template == \mathbb{P}\, Action \\
ConcPrimitive == \mathbb{P}\, Action \\
ConcTemplate == \mathbb{P}(\mathbb{P}\, Action) \\
ActnComp ::= Prim\langle\langle Primitive\rangle\rangle \\
\quad \mid Temp\langle\langle Template\rangle\rangle \\
\quad \mid ConcPrim\langle\langle ConcPrimitive\rangle\rangle \\
\quad \mid ConcTemp\langle\langle ConcTemplate\rangle\rangle
\end{array}
\qquad
\begin{array}{l}
TotalPlan == \mathrm{seq}\, PlanAction \\
TreePlan ::= Tip\langle\langle PlanAction\rangle\rangle \\
\qquad \mid Fork\langle\langle \mathbb{P}_1 (PlanAction \times TreePlan)\rangle\rangle \\
Plan ::= Part\langle\langle PartialPlan\rangle\rangle \\
\qquad \mid Total\langle\langle TotalPlan\rangle\rangle \\
\qquad \mid Tree\langle\langle TreePlan\rangle\rangle
\end{array}
$$

$$
\begin{array}{l}
planpairs : Plan \rightarrow \mathbb{P}\, PlanAction \\
planentities : Plan \rightarrow \mathbb{P}\, EntityModel \\
planactions : Plan \rightarrow \mathbb{P}\, Action
\end{array}
$$

$$
PlanAction == \mathbb{P}(ActnComp \times \mathbb{P}\, EntityModel)
$$

$$
\begin{array}{l}
PartialPlan == \{ps : PlanAction \leftrightarrow PlanAction \mid \forall a, b : PlanAction \bullet \\
\qquad (a, a) \notin ps^+ \wedge (a, b) \in ps^+ \Rightarrow (b, a) \notin ps^+ \bullet ps\}
\end{array}
$$

Fig. 8. Plan components and structure

We identify four types of action that may be contained in plans, called *primitive*, *template*, *concurrent-primitive* and *concurrent-template*. There may be other categories and variations on those we have chosen, but not only do they provide a starting point for specifying systems, they also illustrate how different representations can be formalised

and incorporated within the same model. A primitive action is simply a base action as defined in the agent framework, and an action template provides a high-level description of what is required by an action, defined as the set of all primitive actions that may result through an instantiation of that action-template. An example where the distinction is manifest is in dMARS (see Section 7.2), where template actions would represent action formulae containing free variables. Once all the free variables are bound to values, the action is then a primitive action and can be performed. We also define a concurrent-primitive action as a set of primitive actions to be performed concurrently and a concurrent action-template as a set of template actions that are performed concurrently. A new type, *ActnComp*, is then defined as a *compound-action* to include all four of these types.

Actions must be performed by entities, so we associate every composite-action in a plan with a set of entities, such that each entity in the set can potentially perform the action. At some stage in the planning process this set may be empty, indicating that no choice of entity has yet been made. We define a *plan-action* as a set of pairs, where each pair contains a composite-action and a set of those entities that could potentially perform the action. Plan-actions are defined as a *set of pairs* rather than a *single pair* so that plans containing simultaneous actions can be represented.

We specify three commonly-found categories of plan according to their structure as discussed earlier, though other types may be specified similarly. A partial plan imposes a partial order on the execution of actions, subject to two constraints. First, an action cannot be performed before itself and, second, if plan-action a is before b, b cannot be before a. A plan consisting of a total order of plan-actions is a total plan. Formally, this is represented as a sequence of plan-actions. A plan that allows a choice between actions at every stage is a tree. In general, a tree is either a leaf node containing a plan-action, or a fork containing a node, and a (non-empty) set of branches each leading to a tree. These are formalised in Figure 8, replacing the definition of *Plan* as a given set by a free-type definition to include the three plan categories thus defined.

7.2 Application to BDI Agents

While many different and contrasting single-agent architectures have been proposed, perhaps the most successful are those based on the belief-desire-intention (BDI) framework. In particular, the Procedural Reasoning System (PRS), has progressed from an experimental LISP version to a full C++ implementation known as the distributed Multi-Agent Reasoning System (dMARS). It has been applied in perhaps the most significant multi-agent applications to date. As described in Section 2 above, PRS, which has its conceptual roots in the belief-desire-intention (BDI) model of practical reasoning, has been the subject of a dual approach by which a significant commercial system has been produced while the theoretical foundations of the BDI model continue to be closely investigated.

As part of our work, we have sought to formalise these BDI systems through the direct representation of the implementations on the one hand, and through refinement of the detailed models constructed through the abstract agent framework on the other. This work has included the formal specification [6] of the AgentSpeak(L) language developed by Rao [16], which is a programming language based on an abstraction of

the PRS architecture; irrelevant implementation detail is removed, and PRS is stripped to its bare essentials. Our specification reformalises Rao's original description so that it is couched in terms of state and operations on state that can be easily refined into an implemented system. In addition, being based on a simplified version of dMARS, the specification provides a starting point for actual specifications of these more sophisticated systems. Subsequent work continued this theme by moving to produce an abstract formal specification of dMARS itself, through which an operational semantics for dMARS was provided, offering a benchmark against which future BDI systems and PRS-like implementations can be compared.

Due to space constraints, we cannot hope to get anywhere near a specification of either of these systems, but instead we aim to show how we can further refine the models of plans described above to get to a point at which we can specify the details of such implementations. The value of this is in the ease of comparison and analysis with the more abstract notions described earlier.

We begin our specification, shown in Figure 9 by defining the allowable *beliefs* of an agent in dMARS, which are like PROLOG facts. To start, we define a *term*, which is either a variable or a function symbol applied to a (possibly empty) sequence of terms, and an *atom*, a predicate symbol applied to a (possibly empty) sequence of terms. In turn, a *belief formula* is either an atom or the negation of an atom, and the set of *beliefs* of an agent is the set of all ground belief formulae (i.e. those containing no variables). (We assume an auxiliary function *belvars* which, given a belief formula, returns the set of variables it contains.) Similarly, a situation formula is an expression whose truth can be evaluated with respect to a set of beliefs. A goal is then a belief formula prefixed with an achieve operator or a situation formula prefixed with a query operator. Thus an agent can have a goal either of achieving a state of affairs or of determining whether the state of affairs holds.

The types of action that agents can perform may be classified as either *external* (in which case the domain of the action is the environment outside the agent) or *internal* (in which case the domain of the action is the agent itself). External actions are specified as if they are procedure calls, and comprise an external action symbol (analogous to the procedure name) taken from the set $[ActionSym]$, and a sequence of terms (analogous to the parameters of the procedure). Internal actions may be one of two types: add or remove a belief from the data base.

Plans are *adopted* by agents and, once adopted, constrain an agent's behaviour and act as *intentions*. They consists of six components: an *invocation condition* (or *triggering event*); an optional *context* (a situation formula) that defines the pre-conditions of the plan, i.e., what must be believed by the agent for a plan to be executable; the *plan body*, which is a tree representing a kind of flow-graph of actions to perform; a *maintenance condition* that must be true for the plan to continue executing; a set of *internal actions* that are performed if the plan succeeds; and finally, a set of *internal actions* that are performed if the plan fails. The tree representing the body has states as nodes, and arcs (branches) representing either a goal, an internal action or an external action as defined below. Executing a plan successfully involves traversing the tree from the root to any leaf node.

$[Var, FunSym, PredSym]$

$Term ::= var\langle\langle Var\rangle\rangle \mid functor\langle\langle FunSym \times \operatorname{seq} Term\rangle\rangle$

Atom

$head : PredSym$

$terms : \operatorname{seq} Term$

$BeliefFormula ::= pos\langle\langle Atom\rangle\rangle \mid not\langle\langle Atom\rangle\rangle$

$Belief == \{b : BeliefFormula \mid belvars\, b = \varnothing \bullet b\}$

$SituationFormula ::= belform\langle\langle BeliefFormula\rangle\rangle$
$\quad \mid and\langle\langle SituationFormula \times SituationFormula\rangle\rangle$
$\quad \mid or\langle\langle SituationFormula \times SituationFormula\rangle\rangle$
$\quad \mid true \mid false$

$Goal ::= achieve\langle\langle BeliefFormula\rangle\rangle \mid query\langle\langle SituationFormula\rangle\rangle$

ExtAction

$name : ActionSym$

$terms : \operatorname{seq} Term$

$IntAction ::= add\langle\langle BeliefFormula\rangle\rangle \mid remove\langle\langle BeliefFormula\rangle\rangle$

$TriggerEvent ::= addbelevent\langle\langle Belief\rangle\rangle$
$\quad \mid rembelevent\langle\langle Belief\rangle\rangle$
$\quad \mid toldevent\langle\langle Atom\rangle\rangle$
$\quad \mid goalevent\langle\langle Goal\rangle\rangle$

$Branch ::= extaction\langle\langle ExtAction\rangle\rangle \mid intaction\langle\langle IntAction\rangle\rangle \mid subgoal\langle\langle Goal\rangle\rangle$

$Body ::= End\langle\langle State\rangle\rangle \mid Fork\langle\langle \mathbb{P}_1 (State \times Branch \times Body)\rangle\rangle$

Plan

$inv : TriggerEvent$

$context : optional[SituationFormula]$

$body : Body$

$maint : SituationFormula$

$succ : \operatorname{seq} IntAction$

$fail : \operatorname{seq} IntAction$

Fig. 9. Plans in dMARS

A trigger event causes a plan to be adopted, and four types of events are allowable as triggers: the acquisition of a new belief; the removal of a belief; the receipt of a message; or the acquisition of a new goal. This last type of trigger event allows goal-driven as well as event-driven processing. As noted above, plan bodies are trees in which arcs are labelled with either goals or actions and states are place holders. Since states are not important in themselves, we define them using the given set $[State]$. An arc (branch) within a plan body may be labelled with either an internal or external action, or a subgoal. Finally, a dMARS plan body is either an *end tip* containing a state, or a *fork* containing a state and a non-empty set of branches each leading to another tree.

All these components can then be brought together into the definition of a plan. The basic execution mechanism for dMARS agents involves an agent matching the trigger and context of each plan against the chosen event in the event queue and the current set of beliefs, respectively, and then generating a set of candidate, matching plans, selecting one, and making a *plan instance* for it. Space constraints prohibit going into further details of the various aspects of this work, but we hope that it has been possible to show how increasing levels of analysis and detail enable transition between abstract conceptual infrastructure and implemented system.

8 Summary

Our efforts with BDI agents [4, 3] have provided formal computational models of implemented systems and idealised systems, using the Z specification language [18], a standard (and commonly-used) formal method of software engineering. The benefit of this work is that the formal model is much more strongly related to the implementation, in that it can be checked for type-correctness, it can be animated to provide a prototype system, and it can be formally and systematically refined to produce a provably correct implementation. In this vein, related work has sought to contribute to the conceptual and theoretical foundations of agent-based systems through the use of such specification languages (used in traditional software engineering) that enable formal modelling yet provide a basis for implementation of practical systems,

Indeed, as the fields of intelligent agents and multi-agent systems move relentlessly forwards, it is becoming increasingly more important to maintain a coherent world view that both structures existing work and provides a base on which to keep pace with the latest advances. Our framework has allowed us to do just that. By elaborating the agent hierarchy in different ways, we have been able to detail both individual agent functionality and develop models of evolving social relationships between agents with, for example, our analyses of goal generation and adoption, and our treatment of engagement and cooperation. Not only does this provide a clear conceptual foundation, it also allows us to refine our level of description to particular systems and theories.

Acknowledgement Thanks to Simon Miles and Kevin Bryson for working through this paper and identifying some errors at very short notice.

References

1. R. Aylett, F. Brazier, N. Jennings, M. Luck, C. Preist, and H. Nwana. Agent systems and applications. *Knowledge Engineering Review*, 13(3):303–308, 1998.
2. M. d'Inverno, M. Fisher, A. Lomuscio, M. Luck, M. de Rijke, M. Ryan, and M. Wooldridge. Formalisms for multi-agent systems. *Knowledge Engineering Review*, 12(3):315–321, 1997.
3. M. d'Inverno, D. Kinny, and M. Luck. Interaction protocols in agentis. In *ICMAS'98, Third International Conference on Multi-Agent Systems*, pages 112–119, Paris, France, 1998. IEEE Computer Society.
4. M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures and Languages*, volume 1365, pages 155–176. Springer-Verlag, 1998.
5. M. d'Inverno and M. Luck. Development and application of a formal agent framework. In M. G. Hinchey and L. Shaoying, editors, *ICFEM'97: First IEEE International Conference on Formal Engineering Methods*, pages 222–231. IEEE Press, 1997.
6. M. d'Inverno and M. Luck. A formal specification of agentspeak(l). *Logic and Computation*, 8(3), 1998.
7. M. Fisher. Representing and executing agent-based systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI 890)*, pages 307–323. Springer, 1995.
8. R. Goodwin. A formal specification of agent properties. *Journal of Logic and Computation*, 5(6):763–781, 1995.
9. M. Luck. From definition to deployment: What next for agent-based systems? *The Knowledge Engineering Review*, 14(2):119–124, 1999.
10. M. Luck and M. d'Inverno. A formal framework for agency and autonomy. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press / MIT Press, 1995.
11. M. Luck and M. d'Inverno. Engagement and cooperation in motivated agent modelling. In *Proceedings of the First Australian DAI Workshop, Lecture Notes in Artificial Intelligence, 1087*, pages 70–84. Springer Verlag, 1996.
12. M. Luck and M. d'Inverno. Plan analysis for autonomous sociological agents. In *Submitted to the Seventh International Workshop on Agent Theories, Architectures and Languages*, 2000.
13. M. Luck, N. Griffiths, and M. d'Inverno. From agent theory to agent construction: A case study. In *Intelligent Agents III, LNAI 1193*, pages 49–63. Springer Verlag, 1997.
14. S. Parsons, C. Sierra, and N. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.
15. A. S. Rao. Agentspeak(l): BDI agents speak out in a logical computable language. In W. Van de Velde and J. W. Perram, editors, *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, LNAI 1038*, pages 42–55. Springer-Verlag, 1996.
16. A. S. Rao and M. P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *Proceedings of Knowledge Representation and Reasoning*, pages 439–449, 1992.
17. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, Hemel Hempstead, 2nd edition, 1992.
18. M. J. Wooldridge and N. R. Jennings. Formalizing the cooperative problem solving process. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, 1994.