Infrastructure-support for agent-based development

Ronald Ashri* Michael Luck* Mark d'Inverno†

* Department of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, UK
{R.Ashri,mml}@ecs.soton.ac.uk
† Cavendish School of Computer Science, Westminster University, London W1W 6UW, UK
dinverm@westminster.ac.uk

Abstract. As the field of agent-based computing has continued to develop, there have been several contributions to its theoretical underpinnings, and several others to supporting the efforts of practical systems development. Yet the connection between the two has been limited at best. In this paper we aim to address these limitations through a consideration of appropriate agent infrastructure that can support *principled development* of agent systems based on a strong conceptual framework. As well as a general discussion of infrastructure requirements in this context, we also describe the PARADIGMA implementation environment, based on the SMART agent framework, which represents our initial efforts in this direction.

1 Introduction

Increasingly, the distinguishing quality of current computing environments is the union of loosely-coupled, heterogeneous, networked devices to form larger structures, such as local and wide area networks, which culminate in the Internet. Not surprisingly, this development mirrors the trend amongst organisations to increase the amount of cooperation between disparate units, irrespective of geographic locations. The move is towards a more decentralised, team-based and distributed structure [5], with the use of information technology tools over the Internet acting as the main enabling force. In addition, the personal lives of individuals have also been affected by the technological advances with the use of the Internet in the home increasing daily. Perhaps the most significant change in the use of personal computing devices is the spread, and rise in influence of, embedded and mobile devices with limited computational power, which have found favour in many aspects of everyday life, from mobile phones to personal digital assistants (PDAs), providing a counterpoint to the tradition of desktop computing.

In line with this profile, there is an increasing demand for *integrating* the various different kinds of such devices in order to provide an environment where access to information and services is available in a seamless manner, while transcending physical location and computing platform. The decentralised collaboration structures of organisations need to be supported by appropriate new solutions, whilst remaining integrated with pre-existing applications, often termed *legacy applications*. Furthermore, the simple administration and effective use of existing resources has become a significant issue.

Agent-based systems, by virtue of their defining characteristics of autonomy, reactivity, proactiveness, and social ability, have been suggested as a means of providing solutions to some of these problems [10]. The power of this paradigm stems from the fact that the dynamics of social interaction, such as communication and cooperation, can be used to effectively model such heterogeneous, decentralised and loosely-coupled domains through the interaction of agents.

Nevertheless, for the agent-based systems paradigm to gain widespread use (especially in industrial settings) there are several issues that need to be resolved, a good review of which can be found in [3]. These range from low-level networking concerns such as robust network protocols (e.g. the IPv6 protocol), to appropriate middleware solutions (e.g. CORBA and Jini) and higher level agent communication language standardisation efforts (eg. FIPA ACL [4], KQML [8], etc.). All these efforts are geared towards achieving the primary aim which is, undoubtedly, application development in order to address the needs outlined above. Underpinning the success of these attempts, however, is perhaps a better understanding of the theoretical aspects of multi-agent systems. This will enable the development of applications in a principled manner leading to more robust and extensible solutions.

Theoretical research is useful because it can provide, typically through formal methods, clear concepts and definitions by tackling the ontological and epistemological issues in a research field. In the case of agent-based systems, a good theory could provide definitions of agents as well as explicate the relationships between them and other entities in the world. An appropriate, common theory also makes the comparison, evaluation and sharing of research results easier and can expedite progress in the field.

One of the problems of adopting theoretical work is that it does not easily lend itself to implementation. The reasons for this are twofold. Firstly, the theory might not take into account complications that may arise due to the limitations of the platform on which a program is to be developed. Secondly, the theory may be too abstract for a developer to see a direct connection to an implementation, or the theory might lend itself to many different interpretations at the implementation level. In a development environment where the culture of rapid application development is overpowering, theories are often seen as a hindering rather than facilitating factor. The result of this lack of reconciliation between theoretical approaches on the one hand and development and deployment on the other is that we now have a large variety of alternative concepts of what an agent is, and few means to practically evaluate the various claims made [12].

There are several ways to address this gap between theory and practice. For example, more detail could be added to a theory in order to bring it closer to implementation or, alternatively, software engineering methodologies could be developed providing a path from theoretical specification to practical implementation. In this paper, however, we propose to address the issue through the provision of appropriate infrastructure tools that interpret theoretical approaches and allow for the rapid development of applications. Through the methodical *translation* of a theory into infrastructure, developers can more readily access the overarching concepts, allowing for a more principled use of the theory, without radically changing their methods of application development. Such infrastructure tools can form the basic buildings blocks required for the development and deployment of an application. Furthermore, they can serve to verify the theory's

applicability in real world situations, possibly leading to refinements or even rejection of a theory. We adopt this approach in order to address two concerns. On the one hand there is the need to evaluate, refine and make theory more accessible, and on the other we wish to answer the question of what appropriate infrastructure for agent-based systems actually is.

Applications development support through the provision of appropriate *infrastructure* typically needs to address two important issues. Firstly, we need to identify the significant re-usable and domain independent components that can form part of the infrastructure. Secondly, an appropriate framework through which to allow the application designer to manipulate these elements must be constructed. Both of these tasks are made easier if there is good theoretical work to underpin them. Such a theory can provide suggestions as to the entities that should exist in an agent-based system and their relationship (ontological issues) as well as what can be done with those entities (epistemological issues). Conversely, through this principled application development using the derived infrastructure, we can gain a better understanding of the theory, which can enable its refinement and extension as necessary.

The challenge of developing a usable infrastructure for agent-based systems is to produce a system at the right level of generality. For example, infrastructure that provides support only for network communication is inadequate for any substantial system, while infrastructure that forces a developer to employ, for example, a certain planning algorithm, may be overly specific and consequently constraining. While it is important to realise that infrastructure support goes beyond support for general distributed systems it is equally important to recognise that it cannot be a direct translation of a theory of agent-based systems to a programming language. That can only be one component of a larger structure that attempts to relate that theory to implementation concerns such as networking communication tools, host platform operating possibilities and limitations. This suggests that an agent infrastructure should touch upon high-level issues concerning the structure of individual agents and their interaction as well as lower-level issues.

In this paper, we consider exactly these concerns, and offer an analysis of the requirements for infrastructure to support the development and operation of agent-based systems, informed through experience in developing an agent implementation environment based on a conceptual agent framework. We begin by grounding the discussion through a short description of the environments that we are considering for the application of agent-based systems and elaborate on the kind of modularity that agent infrastructure for such environments should support. We then move on to outline our initial efforts in attempting to realise this set of requirements in the development of the PARADIGMA agent implementation environment by using appropriate conceptual and technical tools. Finally, we review related work and suggest ways to proceed further.

2 Heterogeneous Environments

Increasingly, the range of devices used to access networks is diversifying. This, coupled with the increase in the numbers of users accessing such networks, creates the need for a different approach to distributed computing. While until recently the methodologies

and tools for developing distributed applications called for abstracting beyond location issues, since assumptions could be made about the reliability and performance of networks, we are now forced to take into account both physical and virtual boundaries. The former is necessary due to the latency in information transmission, and the latter due to the partitioning of networks according to the organisational needs of network ownership and administration. In addition, solutions also need to deal with constant change in such environments, which comes about due to the fluctuating nature of organisational hierarchies, changes in needs, replacement of components and the underlying infrastructure, as well as limitations of that infrastructure. More specifically, the following salient characteristics of such environments need to be considered by any attempt to develop practical agent systems in these emerging computing environments.

- The devices used to access information and services vary greatly in capability. At one end of the spectrum, powerful desktop computers typically have much better network support, while at the other end mobile devices have limited computational power, poor display capabilities and uncertain network support. In addition, a whole host of devices occupy the points in between.
- There is a multitude of operating environments and network access protocols.
- As mobile users change geographical locations, they very often also have to change service providers, raising problems of interoperability and security.
- Devices and supporting infrastructure are continuously changed and also upgraded through efforts to offer better support and increased capabilities.

Mobile devices and, more importantly, the need to support mobile users, mean that applications should be able to provide a consistent method of accessing information and services as a user changes both her geographical position and her operating platform for accessing these services. This may entail a need for agents to migrate between devices, such as from a desktop computer to a PDA, or between service providers in order to continue offering support to users. It may also be beneficial, in terms of efficient use of computational power and bandwidth conservation, for agents to migrate to more powerful platforms in order to perform more demanding tasks before returning to a user's device with results.

The main challenge in providing support for agent applications within such extremely heterogeneous environments is finding an effective means of enabling agents to adapt to the environment. This adaptive behaviour should allow the use of different execution mechanisms based on the computational platform, different channels of communication with the user and other entities in the environment (based on network and display capabilities) and, finally, the reconfiguration of agents to enhance their operational capability based on changes in user needs and upgrades to devices.

Agents must thus be able to adapt and improve through the addition or removal of the particular characteristics relating to the adoption and creation of goals to achieve on the one hand, and the ways in which they achieve these goals on the other. For example, an autonomous agent responsible for kitchen appliances might be modified to deal with new devices in the kitchen by adding new goals and (*values* of goals), with plans to achieve the goals, as well as new capabilities for the specific appliance control and interaction. Alternatively, a personal assistant agent residing on a desktop computer

might reduce its normal set of actions (or capabilities) to a minimal set of those that are essential in order to migrate to a mobile PDA while maximising the retained information relating to user preferences, profile, and other relevant and important information.

3 Decoupling Agent Behaviour and Description

3.1 Decoupling for Flexibility and Evaluation

One way to achieve this kind of functionality is to ensure a complete separation of architectural issues on the one hand, relating to the *behaviour* of agents, and the manner in which agents are *described* on the other. Agent descriptions provide an enumeration of the different components that make up an agent, almost in jigsaw-puzzle fashion, including attributes, goals and capabilities, for example. By contrast, agent behaviour is determined through the way in which these components come together inside an agent architecture on a particular execution platform, with a range of complex concerns such as how goals are activated, and capabilities selected. (We will say more about the details of agent description in Section 4.) Separating the *description* of an agent from concerns of control, execution environments, etc., not only makes for good software engineering in terms of modular design, which enables reuse and wide-scale development, but also enables agents to cope in the kinds of environments that we are considering.

In particular, this decoupling is crucial for the flexibility required of agents in heterogeneous and dynamically changing environments; because agent description is independent of agent behaviour, we are free to develop different types of execution platform on which to operate essentially the same agent, but using alternative architectural organisation.

The approach offers benefits to both those with a research-based focus and those with a more practical perspective aimed at real systems development. From the research side, it allows the effective comparison of different agent behaviour algorithms applied to the same agent description, providing a sensible and calibrated means of evaluation. From the development side, it allows the development of execution platforms that are tailored to their specific computing environments. For example, an agent execution platform on a mobile device is naturally more limited in available capacity and features, and might therefore use simpler or less sophisticated behavioural mechanisms than an execution platform on a powerful workstation. In both cases, the same agent description can be applied, but the resulting behaviour leveraging that description would be tailored to the environment within which the agent is executing. In principle, systems developers should eventually be able to access libraries of agent components which can be pieced together and coupled to appropriate execution platforms to achieve the desired effect.

3.2 Decoupling for Mobility

Additionally, decoupling enables agent *mobility* to be achieved in a more lightweight and secure manner. Mobile agents require packaging up through serialisation to be moved between execution platforms [2, 11]; typically this includes the state of the agent, and the agent *as is*. In the case of large agents, or those with many resources or capabilities, the transport costs can become significant, and since one of the key motivating

principles behind mobile agents is to minimise transport by focusing on code rather than data, this can be a problem.

In a decoupled system, however, agents can be packaged as a set of descriptions coupled with specific implementation of capabilities thus minimising transport overheads. Moreover, one of the main problems of mobile execution platforms is effectively securing the underlying infrastructure from malicious agents [9, 16]. Traditionally, such platforms provide the agent with an execution thread, and have minimal control over what happens within that thread other than imposing access rights to the sensitive parts of the system [17]. By imposing constraints on the structure of capabilities through the definition of generic interfaces, we can enforce tighter control over what an agent can and cannot do within an execution platform.

3.3 Conceptual Infrastructure

We argue that a strong and clear conceptual underpinning is required at the level of infrastructure so as to guide its development as well as the subsequent development of agent superstructures. In a series of papers (e.g. [7, 13, 14]), Luck and d'Inverno have provided such a conceptual foundation through the development of a framework for agent systems that supports many of the features that we listed above. Their SMART agent framework provides an encompassing structure that clearly differentiates between agent and non-agent entities in the environment, and specifies agents in a compositional way. In essence, the framework proposes a four-tiered hierarchy that includes the generic and abstract notion of an entity from which objects, agents and autonomous agents are, in turn, derived. Figure 1 shows a Venn diagram that describes the different levels in the hierarchy, and outlines the ways in which they are related. Though we will not offer a detailed exposition of the framework, we review the key concepts below.

The essential ingredients of the SMART framework are the following four types:

- attributes, which are features of the world that can potentially be perceived in an omniscient sense;
- actions, which can change the state of the environment in which they are performed by either adding or removing attributes;
- goals, which are states of affairs to be achieved in the environment; and
- motivations, which are non-derivative high-level structures that lead to the generation and adoption of goals, and affect the outcome of any task intended to satisfy those goals.

We can then define the components of the four-tiered framework using these types. The *entity* serves as an *abstraction mechanism*; it provides a template from which objects, agents and autonomous agents can be defined. Anything that is considered to be a single component is represented as an *entity*. These entities may have complex descriptions, but at the very highest level they are just collections of attributes.

Entity

$$
\begin{array}{l}
attributes : \mathbb{P}\, Attribute \\
capabilities : \mathbb{P}\, Action \\
goals : \mathbb{P}\, Goal \\
motivations : \mathbb{P}\, Motivation \\
\hline
attributes \neq \{\}
\end{array}
$$

An entity must be situated in an environment and, conversely, an environment must include all the entities within it. There may well also be other attributes that are not associated with an entity and so the union of all the attributes from each entity will only be a subset (in general) of all the attributes that comprise the environment. In the following schema, the *environment* variable is the set of all environment attributes, and the *entities* variable the set of all entities in that environment.

Env

$$
\begin{array}{l}
environment : \mathbb{P}\, Attribute \\
entities : \mathbb{P}\, Entity \\
\hline
environment \neq \{\,\} \\
\bigcup \{ e : entities \bullet e.attributes \} \subseteq environment
\end{array}
$$

Objects are then simply entities with sets of capabilities that can be performed to change the state of the environment.

Object

$$
\begin{array}{l}
Entity \\
\hline
capabilities \neq \{\,\}
\end{array}
$$

In turn, agents are objects with sets of goals, where goals are defined as desirable environmental states, and autonomous agents are those agents able to generate their own goals through the motivations that drive them. Here, motivations can be regarded as preferences or desires of an autonomous agent that cause it to produce goals and execute plans in an attempt to satisfy those desires.

Agent

$$
\begin{array}{l}
Object \\
\hline
goals \neq \{\,\}
\end{array}
$$

AutonomousAgent

$$
\begin{array}{l}
Agent \\
\hline
motivations \neq \{\,\}
\end{array}
$$

For each of the four high-level components we also provide a skeletal architecture to describe its interaction. In order to show this let us consider the description of agent. In general, an agent is able to perceive its environment. An agent in an environment may have a set of percepts available, which are the possible attributes that it 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.

To distinguish between representations of mental models and representations of the *actual* environment, we introduce two types, *View* and *Environment*. The first of these is defined to be the perception of an environment by an agent. This has an equivalent type to that of *Environment*, but now physical and mental components of the same type can be distinguished.

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

AgentPerception

$$Agent$$
$$perceivingactions : \mathbb{P}\, Action$$
$$canperceive : Environment \rightarrow \mathbb{P}\, Action \nrightarrow View$$
$$willperceive : \mathbb{P}\, Goal \rightarrow View \rightarrow View$$

$$perceivingactions \subseteq capabilities$$
$$\forall\, env : Environment;\ as : \mathbb{P}\, Action \bullet$$
$$\quad as \in \mathrm{dom}(canperceive\ env) \Rightarrow as = perceivingactions$$
$$\mathrm{dom}\, willperceive = \{goals\}$$

In addition, an agent will be able to perform actions determined by its goals, perceptions and the environment. This is specified by the *agentactions* function in the *AgentAction* schema below, which is dependent on the goals of the agent, the actual perceptions of the agent and the current environment. The first predicate requires that *agentactions* returns a set of actions within the agent's capabilities, while the last predicate constrains its application to the agent's goals.

AgentAction

$$Agent$$
$$ObjectAction$$
$$agentactions : \mathbb{P}\, Goal \rightarrow View \rightarrow Environment \rightarrow \mathbb{P}\, Action$$

$$\forall\, gs : \mathbb{P}\, Goal;\ v : View;\ env : Environment \bullet$$
$$\quad (agentactions\ gs\ v\ env) \subseteq capabilities$$
$$\mathrm{dom}\, agentactions = \{goals\}$$

Now that these skeletal architectures have been described it is then possible to define the *state* of an agent or autonomous agent within an environment. Once an agent is

placed in an environment, its attributes are accessible and it is possible to specify the *possible percepts* and *actual percepts* of the agent. These are denoted by the variables, *possiblepercepts* and *actualpercepts*, which are calculated using the *canperceive* and *willperceive* functions respectively. The action or actions the agent actually performs in the environment are a function of its goals, its percepts and the environment itself. The reader will notice that the schema below also includes a schema called *ObjectState* (not specified here) that defines the state of the higher-level SMART object component in an environment. This should provide an indication of how increasingly more refined and detailed concepts are built incrementally and systematically from higher level ones. The structure of the very basic framework and related model can be seen in Figure 1. An arrow here simply indicates schema inclusion.

AgentState

$$
\begin{array}{l}
\textit{AgentPerception} \\
\textit{AgentAction} \\
\textit{ObjectState} \\
\textit{posspercepts}, \textit{actualpercepts} : \textit{View} \\
\hline
\textit{actualpercepts} \subseteq \textit{posspercepts} \\
\textit{posspercepts} = \textit{canperceive environment perceivingactions} \\
\textit{actualpercepts} = \textit{willperceive goals posspercepts} \\
\textit{perceivingactions} = \{\,\} \Rightarrow \textit{posspercepts} = \{\,\} \\
\textit{willdo} = \textit{agentactions goals actualpercepts environment}
\end{array}
$$

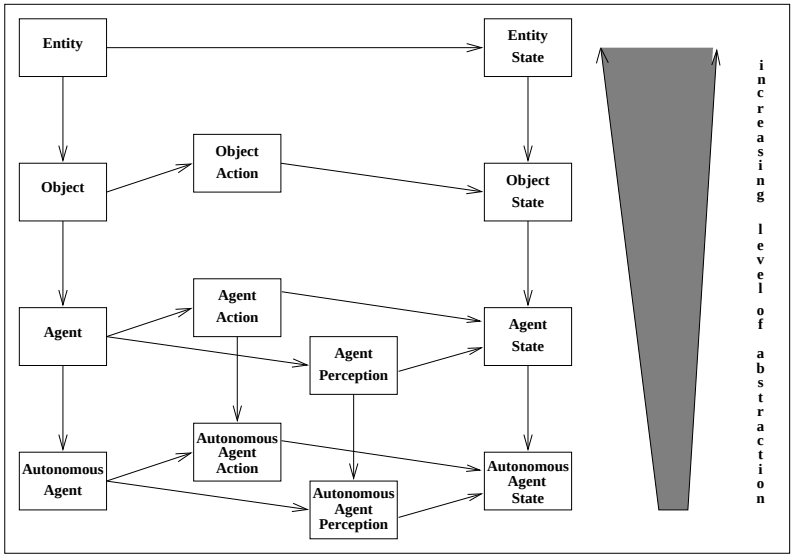

Fig. 1. Structure of the SMART Framework

In addition to these basic levels, and in order to further explicate the consequences of their framework, Luck and d'Inverno introduce two additional refinements: *neutral objects* are objects that are not agents, and *server agents* are agents that are not autonomous [14]. The relationship between neutral objects and server agents is complementary, since neutral objects give rise to server agents when they are ascribed goals by other agents in the environment. Once these goals are achieved or they are no longer feasible, server agents revert back to neutral objects.

In short, this conceptual framework provides a basis for us to use in reasoning about agent and non-agent entities within a coherent whole, while at the same time providing us with the requisite level of component differentiation to underpin the division between behaviour and description. We now move on to discuss how these concepts can be encapsulated within the technical framework that can provide an infrastructure for agent-based systems.

3.4 Technical Infrastructure

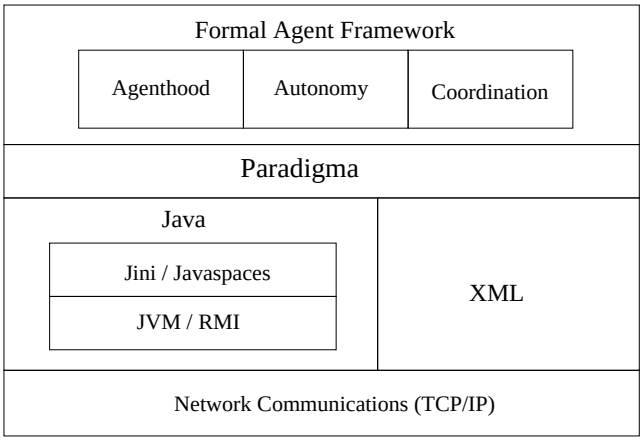

Fig. 2. Paradigma overview

In line with the aims discussed above, and based on the conceptual infrastructure outlined, we have developed an agent system, PARADIGMA, that provides a technical infrastructure for the development of agent applications. PARADIGMA unites theory with practical implementation in an attempt to provide an accessible and grounded set of tools for agent development. Key to this is ease of understanding and simplicity of use, as well as an ability for elegant expansion and adaptation to change.

An overview of PARADIGMA is presented in Figure 2. At the top level, the agent framework provides the conceptual tools that guide the design of the agents and define the relationships between them. PARADIGMA can be considered as implementing the framework through the use of the standard technologies that appear at the lower level

(and which we discuss later). We have opted for the use of standard technologies for the underlying functionality as opposed to a proprietary system not only because it provides a sensible and robust route for development, but also because it enables interaction and cross-development with others, and makes access to the overarching conceptual and theoretical issues easier. Indeed, one of the arguments advanced in justification of a certain degree of reticence on behalf of developers in relation to agent systems is, in many cases, a reliance on non-standard technologies. We seek to ensure that this is not the case here, and that recent convergence between the fields of autonomous agents, object-oriented systems and distributed systems contributes to our own efforts in the agent arena.

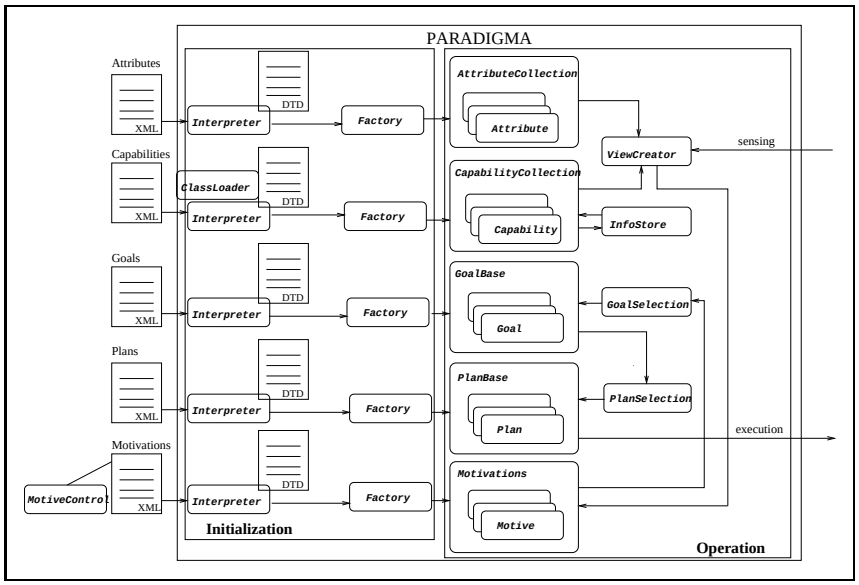

Fig. 3. Agent creation and agent operation

Decoupling Behaviour and Description In order to achieve the desired decoupling of description and behaviour at the implementation level we have made clear distinctions between the task of composing an agent by assembling the required building blocks, such as attributes, capabilities and goals and relating these components via decision mechanisms.

A description of an entity in PARADIGMA is, in essence, a collection of XML documents. Each document contains within it a set of components of the same type. For example, an *attribute* document, which is the simplest structure, contains a series of type-value definitions that can be declared either constant or variable. A *capabilities* document, on the other hand, contains a description and type of the capability and also a

link to the code that implements the capability (in the same spirit as dMARS plans [6], for example). This enables the implementations to vary in order to suit executing platforms, or so as to provide newer versions of capabilities. It is envisaged that eventually the developer will have access to libraries of capabilities that can be linked to the agent descriptions. Goals, plans and motivations are more elaborate structures and can vary according to the desired level of complexity required by the developer. For example, a simple plan structure may just define a series of capabilities that an entity should perform, while a more complicated structure may also include invocation conditions and postconditions, as well as elements that should remain true during the execution of the plan.

Once such a description has been pieced together based on the requirements of the application, the developer can insert it into an execution platform. At this stage, the XML documents will be interpreted and the appropriate capabilities will be retrieved. The executing environment then couples the entity to decision mechanisms in order to effect execution.

The complete process of agent creation and execution is illustrated in Figure 3, which is divided into two stages, initialization and operation. An agent is created by supplying the required building blocks of attributes, capabilities, goals, plans and motivations. As mentioned earlier, capabilities require specific implementations, whose location is made part of their description. The last element required at this stage, especially for autonomous agents, is some form of control mechanism that will dictate, for example, how motivations change as the state of the environment changes. In the figure this is illustrated by the *MotiveControl* component, to enable it to adjust its motivations as the state of the environment changes. In the current implementation of PARADIGMA, motivations are seen as a tuple of three variables: an identifying name, a strength or salience rating and a boolean indicating whether the strength is variable. The control component could be as simple as a set of rules indicating the values motivation strengths should take as attributes of the environment change, though it could equally provide a more sophisticated set of constraints.

All this information is interpreted and checked before the agent is constructed by the *Factory* components, based on the requirements of the execution environment in question. Following a successful initialisation stage, the execution environment becomes responsible for executing the supplied agent description. In the figure, we show some of the essential components for these tasks, such as *ViewCreator* for collecting information about the environment, *InfoStore* for maintaining acquired information so that it may be shared with others if appropriate and, finally, plan and goal selection units.

As can be seen, by taking this approach we have a complete decoupling of all the components that comprise an agent from the agent development and execution platform. Furthermore, it becomes trivial to change platforms in order to suit particular situations, or in order to incorporate other desired changes and advances. For example, if we wish to provide different descriptions of *attributes*, all that is required is to develop a new DTD or XML Schema and replace the current *Factory* component with a new one. We can thus allow for the evaluation of different implementations of capabilities, decision mechanisms, etc, while still remaining within the environment that is provided by our conceptual infrastructure.

The next stage in the development of PARADIGMA is to reverse this process and capture the state of an agent back in a set of XML files. The new set of XML descriptions would reflect the changes that the agent has gone through during execution, and would allow for the easy transport of the agent to another platform. There, the agent may make use of different decision mechanisms and, crucially, different implementations of capabilities that may be optimized for the new platform. This provides an interesting departure from current mobile code systems, since we are not limited to any particular programming language in order to achieve agent mobility. Furthermore, because the actual code that will need to be loaded is reduced to the capabilities of the agent, while the integration with decision mechanisms is up to the platform, security concerns are slightly different. For example, although the XML descriptions may move from untrusted to trusted environments, the code that implements capabilities may always come from trusted environments since it is not inextricably attached to the agent. These issues, of course, require further consideration since the problem of untrusted platforms always remains open.

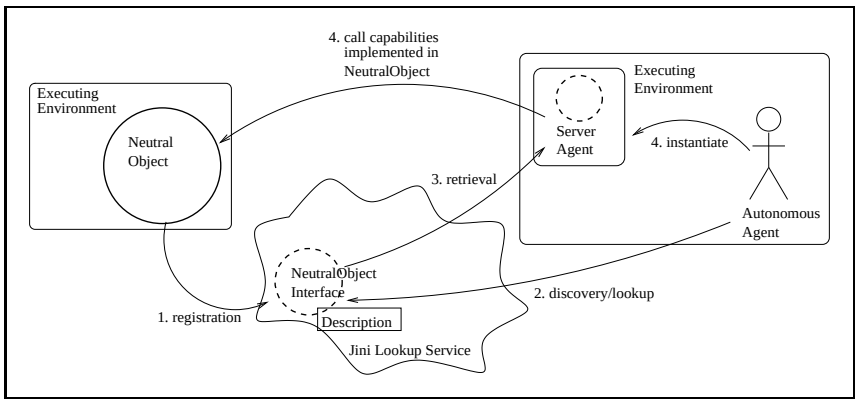

Fig. 4. Using neutral objects

Enabling Agent Communities The main challenge at the level of distribution and support of agents involves the provision of a middleware infrastructure layer that is able to support dynamic communities of entities where constant change is always part of the agenda. For the purposes of PARADIGMA, we have chosen Jini because of the features that come closest to fulfilling all the requirements discussed earlier. A more thorough review than is possible here of the Jini infrastructure to support implementation of Luck and d'Inverno's framework is given in [1], but we outline and illustrate the key points below.

Entities executing in a PARADIGMA platform can at any time make use of available facilities in order to announce their existence on the network. Note that this is not a requirement but an option, since it may not always be desired or even feasible to per-

form such announcements. This is important in terms of separating the issues related to cooperation with, and discovery of, other agents from issues related to the operation of a single agent. Nevertheless, if a decision to make an announcement has been taken, then PARADIGMA will attempt to discover the available registries, represented by Jini lookup services. Once such lookup services are discovered, the entity will guide the platform as to the information it wants to make known about itself. This information will be registered in the Jini lookup service along with a proxy that will allow interested parties to make direct contact with the entity. Lookup services are managed through a leasing mechanism that requires registered entities to renew their interest in retaining their information within the lookup service or have their information discarded. In essence, Jini provides the required network connectivity and administration infrastructure for the support of heterogeneous communities of entities, thus making it suitable as an environment for implementation of the conceptual framework described above.

By way of example, Figure 4 illustrates how neutral objects can be discovered and used by other agents in a Jini-supported environment, and in particular PARADIGMA. A device or software component, represent by a neutral object (drawn using a solid circle line), creates an appropriate description of itself and registers the required information relating to the attributes and capabilities in a Jini lookup service along with a proxy (drawn using a dotted circle) that can be used to access it. If an agent (represented by the stick figure) decides that the device is useful for its needs, it downloads the proxy and creates a server agent with the relevant goals, and which wraps around the proxy. Once the server agent has achieved its goals it is discarded and the neutral object is disengaged.

In the case of autonomous agents, the registered proxy can be an interface that implements appropriate communication protocols. Other agents could then retrieve this implementation so as to communication with the agent. An interesting dynamic here is that the communication interfaces can act as *translators* from one communication protocol to another, and can vary according to the entities the autonomous agent wishes to communicate with. For example, in environments where bandwidth and reliability are important, the implemented interface could direct messages to appropriate messaging routes that would ensure the messages are not lost.

4 Discussion

4.1 Related Work

PARADIGMA attempts to address a wide range of issues starting with identifying the appropriate concepts to support agent-based systems infrastructure, and ranging to consider the appropriate technologies for implementing such concepts. In terms of the approach we have adopted, which clearly distinguishes the relationship between agent and non-agent entities, and separates issues of description from issues of behaviour, PARADIGMA can be thought of as a system that integrates several different strands of agent research. Similar work has been done with the DARPA-funded Control of Agent-Based Systems (CoABS) program [15], whose main goal is to provide the appropriate infrastructure to enable integration of heterogeneous agent-based systems. At the middleware layer it makes use of Jini network technology and, similar to PARADIGMA, it

allows for the registration of agents to the Jini lookup service along with appropriate descriptions. CoABS also provides mechanisms for agent communication through RMI. In terms of the layers of required infrastructure discussed earlier, CoABS addresses the middleware layer by facilitating management and communication of agents, but it does not address the higher level issues of mobility and intelligent agents. As such, it takes a different approach to PARADIGMA by transferring the burden of addressing these concerns to application developers. CoABS could, therefore, act as an integrator of other infrastructures but, does not provide the required functionality to allow mainstream developers to use agent concepts directly.

4.2 Conclusions and Further Work

Agent-based systems have a vital role to play in the immediate development of applications and services across the distributed and increasingly pervasive computing fabric of our everyday environments. The convergence of related fields of distributed computing and object-oriented development also provides extra support and impetus for the adoption of agent technology into the mainstream. Yet this provides an opportunity that can only be taken if two conditions hold. First, mainstream technologies must be used for infrastructural underpinning of agent applications to enable accessibility, further development, and, importantly, *integration*. Second, the kinds of applications that we build must be constructed in ways that facilitate flexibility, evaluation, and the potential for secondary capabilities (that are still critical for many applications and environments) like mobility.

One of the main problems that have delayed the wide deployment of agent-based systems has been the lack of integration between different systems. The agreement on common infrastructure would enable that integration, especially if the infrastructure made use of other standards and systems that have already found a wider acceptance, such as Jini at the middleware level.

In constructing PARADIGMA, we have done just this, through our two-levels of technical infrastructure and conceptual infrastructure, which support the decoupling of agent behaviour from agent description to achieve exactly these aims. PARADIGMA is a fully functional execution and development platform with which to build real applications, and all the work described in this paper is fully implemented. The next stage in its development is, at one level to build a broad range of applications to demonstrate its suitability, and at another to examine the mechanisms required for dynamic self-modification of agent capabilities.

References

1. Ronald Ashri and Michael Luck. Paradigma: Agent implementation through Jini. In A. M. Tjoa, R.R. Wagner, and A. Al-Zobaidie, editors, *Eleventh International Workshop on Databases and Expert System Application*, pages 453–457. IEEE Computer Society, 2000.
2. J. Baumann, F. Hohl, K. Rothermel, and M. Straer. Mole - concepts of a mobile agent system. *World Wide Web*, 1(3):123–137, 1998.
3. J. Bradshaw. Agents for the masses. *IEEE Intelligent Systems*, 14(2):53–63, 1999.

4. Bernard Burg, Jonathan Dale, and Steven Willmott. Open standards and open source for agent-based systems. *Agentlink News*, (6):2–5, 2001.
5. D. DeSanctis and B. Jackson. Co-ordination of information technology management: Team based structures and computer-based communication systems. *Journal of Management Information Sciences*, 4(10):85–110, 1994.
6. 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*, pages 155–176. Springer-Verlag, 1365, 1998.
7. M. d'Inverno and M. Luck. A formal view of social dependence networks. 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*, volume 1087, pages 115–129. Springer Verlag, 1996.
8. T. Finin, Y. Labrou, and J. Mayfield. Kqml as an agent communication language. In J. Bradshaw, editor, *Software Agents*. MIT Press, Cambridge, 1997.
9. Robert Gray, David Kotz, George Cybenko, and Daniela Rus. D'agents: Security in a multiple-language, mobile agent system. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 154–187. Springer-Verlag, 1998.
10. Nicholas R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.
11. Danny Lange and Mitsuru Oshima. *Programming and Deploying Java(tm) Mobile Agents with Aglets(tm)*. Addisson-Wesley, 1998.
12. M. Luck. From definition to development: What next for agent-based systems. *Knowledge Engineering Review*, 14(2):119–124, 1999.
13. M. Luck and M. d'Inverno. A formal framework for agency and autonomy. In *95*. 254–260, 1995.
14. M. Luck and M. d'Inverno. Engagement and cooperation in motivated agent modelling. In *Proceedings of the First Australian DAI Workshop*, volume 1087 of *Lecture Notes in Artificial Intelligence*, pages 70–84. Springer Verlag, 1996.
15. C. Thompson, T. Bannon, T. Pazandak, and V.Vasudevan. Agents for the masses. In *Workshop on Agent-based high Performance Computing: Problem Solving Applications and Practical Deployment*, 1999.
16. Christian F. Tschudin. Mobile agent security. In Matthias Klusch, editor, *Intelligent Information Agents*, pages 431–446. Springer-Verlag, 1999.
17. Tom Walsh, Noemi Paciorek, and David Wong. Security and reliability in concordia. In *31st Annual Hawai'i International Conference on System Sciences (HICSS31)*, 1998.