

# Mobile BDI Agents

R.W. Collier, C.F.B. Rooney, R.P.S. O'Donoghue, G.M.P. O'Hare,

*PRISM Laboratory, Dept. of Computer Science, University College Dublin (UCD),  
Belfield, Dublin 4, Ireland*

*{Rem.Collier, Colm.Rooney, Ruadhan.ODonoghue, Gregory.OHare}@ucd.ie*

**Abstract.** Much work has been done in the area of Mobile Agents. However, the majority of this research has emphasized “mobile” and largely ignored “agent”. This paper attempts to rectify this through the presentation of research that combines the areas of Intentional Agents and Mobility, namely the construction of Mobile BDI Agents.

## 1 Introduction

Much work has been carried out recently on constructing *mobile agents* [GH+97] [JVS96] [KG+97] [PS97]. However, much of this research has tended to focus upon the word “mobile” rather than the word “agent” [Mil99]. This paper redresses the balance by combining traditional agents research and mobile agents research to present a new smarter breed of mobile agent.

BDI agents [RG91] [Jen93] [OJ96] are complex deliberative reasoning systems. Generally, they are considered heavyweight software entities. This paper presents a view of BDI agents, based upon Shohams Agent-Oriented Programming (AOP) Paradigm [Sho93], that are lightweight software entities, which may easily be transmitted over a network. Such a model naturally fits the mobile agents domain. Consequently, it has been used to construct a new category of agent, a *Mobile BDI Agent*.

## 2 Why Mobility?

Mobile Agents have been applied to a number of areas, including mobile computing (e.g. PDAs, laptops); information retrieval (when it is easier, perhaps because of bandwidth, to send the agent to the information); dynamic deployment of software; and real-time controls and remote instruments [HCK95]. [Mil99] notes the lack of autonomous decision-making power within such systems, and importance of addressing this issue. It further identifies the lack of any “killer” application that requires mobile agents. This paper presents a solution to the lack of decision-making power for mobile agents, and motivates the use of mobility in the delivery of self-organising agent communities (section 7). This is achieved through the use of AOP in delivering BDI agents (section 4). This presents such agents as lightweight software entities that are amenable to transmission over a network.

### 3 Mobile Agent Systems

[GH+97] characterises a mobile agent as “*a software entity that exists in a software environment. It inherits some of the characteristics of an intelligent agent. A mobile agent must contain at the very least the following models: an agent model, a life cycle model, a computational model, a security model, a communication model and finally a navigation model.*” Of the mobile agent systems currently available, perhaps five stand out above the rest: Telescript Agents, Ara Agents, D’Agents, Tacoma Agents, and Grasshopper Agents.

Telescript [Whi94] agents travel within an *electronic marketplace*. The electronic marketplace is the entirety of all *places* where a Telescript engine provides services. Telescript agents travel between places with the go instruction; services are used with the meet instruction. Ara [PS97] is a platform for the portable and secure execution of mobile agents in heterogeneous networks. Mobile agents in Ara are programs with the ability to change their host machine during execution while preserving their internal state. Ara's specific aim in comparison to similar platforms is to provide full mobile agent functionality while retaining as much as possible of established programming models and languages. D’Agents is a mobile-agent system whose agents can be written in Tcl, Java, and Scheme [KG+97]. The ultimate goal of D’Agents is to support applications that require the retrieval, organization and presentation of distributed information in arbitrary networks. An agent in TACOMA (Tromsø And Cornell Moving Agents project) [JVS96] is a piece of code that can be installed and executed on a remote computer. Such an agent may explicitly migrate to other hosts in the network during execution. Grasshopper is the first mobile agent environment that is compliant to the industry standard supporting agent mobility and management (OMG MASIF). The MASIF (Mobile Agent System Interoperability Facility) standardisation is a joint submission of GMD FOKUS, International Business Machines Corporation, Crystaliz, General Magic, and the Open Group. It sets forth a number of constraints and ideals for mobile agent systems. As with any normative standard compliance ensures compatibility with other agent environments or applications based on the same standard thus avoiding costly and time-consuming integration procedures.

### 4 BDI Agents

#### 4.1 Belief Desire Intention (BDI)

Much research work has been commissioned on Multi-Agent Systems (MAS) and Distributed Artificial Intelligence (DAI) [BG88], [DLC89], [OJ96]. Significant quantities of this research has been concerned with developing practical deliberative reasoning models. Perhaps the most successful attempt at constructing such models have come from the application of a *mental state* comprising a set

of mental attitudes such as *beliefs, obligations, goals, and commitments* [Mc79]. Within this field, a consensus mental state has emerged, the Belief-Desire-intention (BDI) architecture [RG91] [Jen93] [OJ96]. In this terminology, an agent can be identified as having: a set of beliefs about its environment and about itself; a set of desires which are computational states which it wants to maintain, and a set of intentions which are computational states which the agent is trying to achieve.

#### 4.2 Agent-Oriented Programming

Agent-Oriented Programming (AOP) is a relatively new programming paradigm introduced by Yoav Shoham that “*promotes a societal view of computation, in which multiple agents interact with one another*” [Sho93]. At the conceptual level, this view presents AOP as a specialisation of the Object-Oriented Programming (OOP) Paradigm. This relationship can be further extended to a practical level with the presentation of a framework for the delivery of AOP languages as seen in figure 1.



**Figure 1: An illustration of the similarities between OOP and AOP programming environments.**

This framework supports the development and delivery of agents that use the reasoning models identified in section 4.1. Further, by presenting the decision-making machinery as a generic agent interpreter the agent is reduced to a lightweight software entity (mental state + program) that is easily transmittable over a network. Finally, of paramount interest in terms of the design of agents is the relationship between a Class and an Agent Design. This relationship supports the use of design patterns [GH+95] in the agent development process and promotes the reuse of agent code.

## 5 The Agent Factory System

### 5.1 What is Agent Factory?

The Agent Factory System is an Agent Prototyping Environment (APE) founded on Shohams’ Agent-Oriented Programming (AOP) Paradigm [Sho93]. Earlier versions of the system have been presented in [Col96][OC+98][CO99]. The Agent Factory System has been designed around two principle needs:

- The need for an environment that supports the delivery of agent-oriented applications.
- The need for a flexible and easy to use environment that supports the development and testing of heterogeneous agent designs.

These needs are met through the systems separation into two parts: *the run-time environment*, and *the development environment*. These environments are organised such that the run-time environment contains all the machinery essential for the delivery of agent-oriented applications, and the development environment that extends this run-time environment to include a toolset that simplifies the agent development process. These environments and their corresponding toolsets are illustrated in figure 2. As

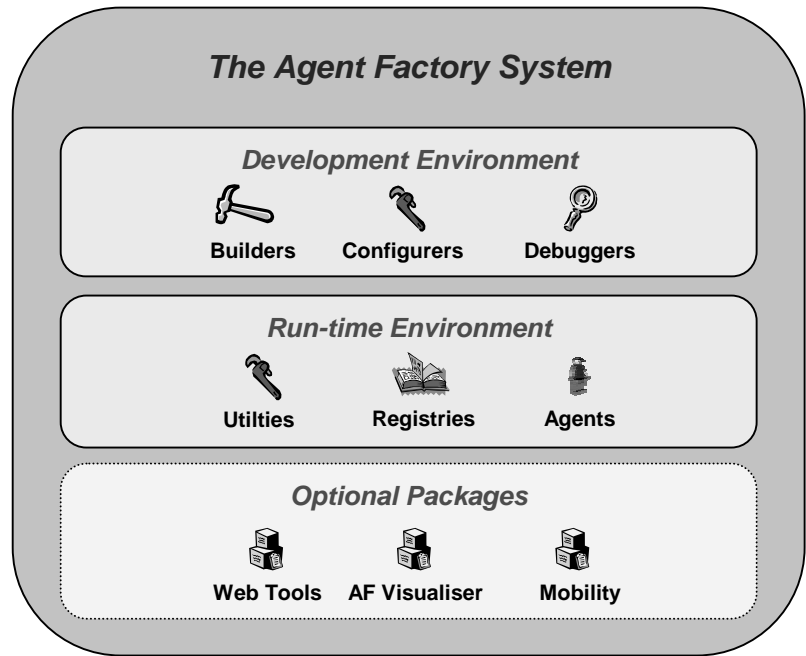


Figure 2: The Agent Factory System

can be seen, the core of the Agent Factory System is a collection of components and tools that provide the basic features needed to develop agent-oriented applications. This core can be extended by the addition of various optional packages (a package is a collection of additional components and tools) as required.

### 5.2 Agent Factory and Agent-Oriented Programming

Agent Factory implements the AOP framework presented in section 4.2. It achieves this through the delivery of three languages: (1) a mental state content language, (2) inter-agent communication carrier language, and (3) a language that permits the programming of agent behaviours. An appropriate algorithm is implemented within the agent interpreter that manages the update of the mental state based upon the given program. Classically, such algorithms follow a *perceive-deliberate-act* cycle, within Agent Factory this is achieved through the inclusion of two additional structures within the agent

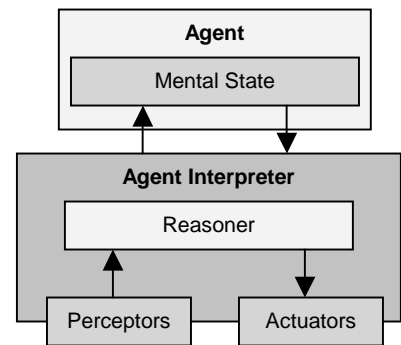


Figure 3: Extending the Agent Interpreter in Agent Factory

interpreter, namely a set of perceptors (that allow the agent to view its environment), and a set of actuators (that allow the agent to act within its environment). The algorithm invokes the perceptors and actuators as needed (see figure 3).

### 5.3 Programming Agents in Agent Factory

A default set of the languages identified in section 5.2 is provided within Agent Factory. For (1) a first-order logic is used for beliefs. This is used in conjunction with additional structures to form a Belief-Commitment-Action-Plan mental state architecture. (2) is provided in the form of the Teanga ACL [RO+99] although other languages such as the KQML [FLM97] and FIPA ACL [FIPA99] are also permitted. Finally, (3) is constructed in the form of a set of *commitment rules* that define conditions under which commitments may be adopted. Some examples of these languages are presented in figure 4.

<b>The content language (Beliefs):</b>	
<code>Bel(goto(X))</code>	agent believes it wants to go to X
<code>Bel(cango(X))</code>	agent believes it can go to X
<code>Bel(name(rem))</code>	agent believes its name is "rem"
<code>Bel(knows(X))</code>	agent believes it knows agent X
<b>The programming language (Commitment Rules):</b>	
<code>Bel(goto(X)) =&gt; Com(Self, Now, migrate(X))</code>	if agent believes it wants to go to X, then commit to migrating to X
<code>Bel(knows(X)) =&gt; Com(Self, Now, inform(X, likes(Self, icecream)))</code>	if agent believes it knows X, then commit to informing X that it likes ice-cream

Figure 4: Some examples of the default content language and programming language of Agent Factory

Within Agent Factory, programming an agent is a two-step process: firstly, add appropriate perceptors and actuators, and secondly define a set of commitment rules. This is illustrated in section 6 through the construction of the demonstrator.

## 6 Building Mobile BDI Agents

This section presents some preliminary work on constructing mobile BDI agents. It is structured into three sections. The first section identifies the main issues behind producing mobile agents, whilst the second and third sections outline how these issues have been dealt with in the Agent Factory System, in particular, the third section deals with the issue of guaranteed message delivery.

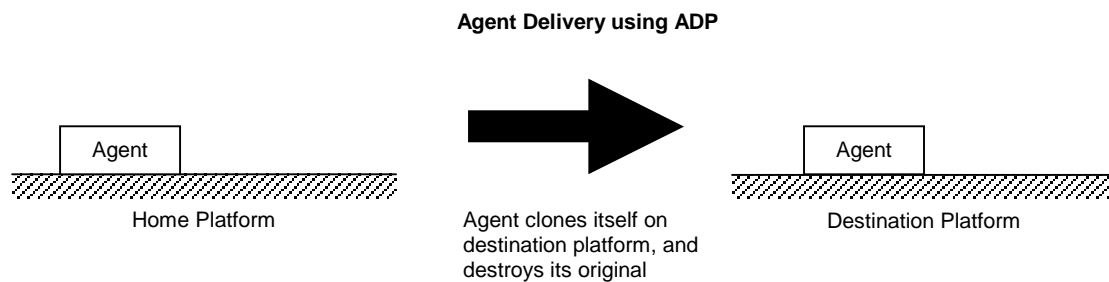
### 6.1 Some Issues

Mobile agents move by transmitting their code from one computer platform to another. Often this is termed *migration*. To ensure migration may occur, four main issues must be dealt with: (1) that there exists appropriate machinery to receive the agent at the destination, (2) that this machinery is able to reconstruct the agent, (3) that the machinery is able to run the agent, and (4) that the agent has the ability to transmit itself to the destination correctly. In addition, given the well-documented bandwidth

limitations of the Internet, an additional issue arises: (5) that the machinery should deliver mobility with the minimum network cost possible. Finally, given these agents are social (6) a transparent approach to inter-agent communication is paramount.

## 6.2 Integrating AOP and Mobility

As shown in section 4.2, AOP delivers a programming framework in which agents are *lightweight* software entities, satisfying issue (5) from section 6.1. By adopting AOP, both the host and the destination platforms must contain agent interpreters satisfying (3). Further, agents may be transmitted across a network with minimal cost through a simple ASCII network protocol, the *Agent Delivery Protocol (ADP)*. This approach is illustrated in figure 5 below.



**Figure 5: Agent Migration**

Using such a protocol requires that AOP environments be supplemented with an ADP compliant server, and that functionality be provided for the conversion of agents into a collection of ADP statements. The ADP server must have sufficient functionality to receive ADP statements (see figure 6) and to construct agents based upon these statements. The use of ADP and its corresponding features satisfy issues (1), (2), and (4). This approach has been applied to the Agent Factory System with the inclusion of a *mobility package* containing a *Migration Server Utility* and a *MobileAgent* design.

```

Creating the Clone:
CREATE_AGENT <class> <controller> <name>

Initialising the mental state:
INITIALIZE <name> <attitude> [<items>*]

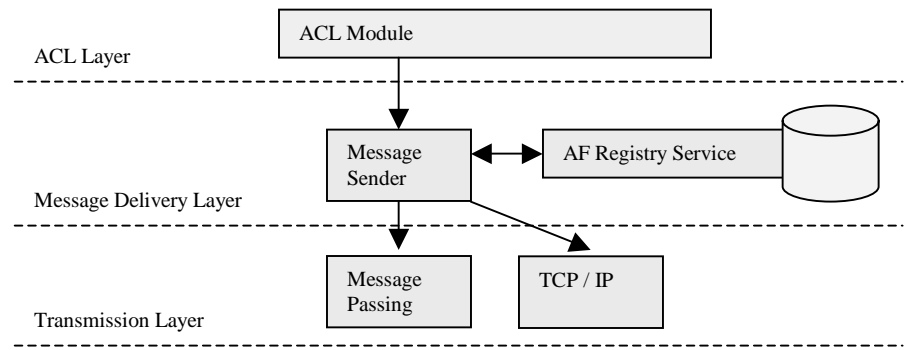
```

**Figure 6: A Simple Agent Delivery Protocol**

## 6.3 Building Transparent Communication

Constructing agents that are mobile, presents additional problems in terms of communication, in particular, guaranteeing the delivery of a message to an agent, issue (6). Problems emerge as a consequence of the dynamic nature of mobile agents in that an agent wishing to communicate does not know where this agent is, and even if it did know where the agent was earlier, cannot guarantee that it is still there.

The solution is to provide a transparent communication model, integrated into the agent interpreter, which allows agents to interact no matter where they are physically located. This model is presented in figure 7. It combines three distinct

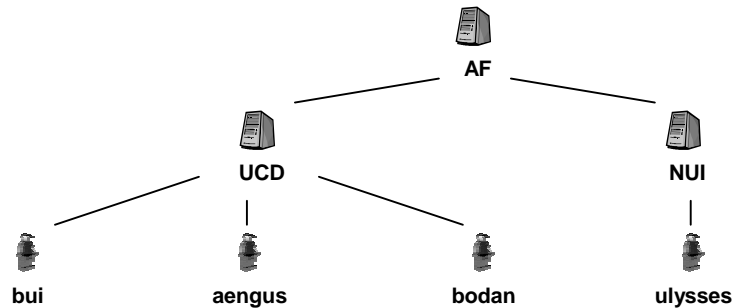


**Figure 7: The 3-layer inter-agent communication model**

layers: the *ACL Layer* that deals with message construction, the *Message Delivery Layer* that deals with the resolution of agent names into addresses, and the *Transmission Layer* that deals with the mechanics of transmitting a message using some transmission medium (i.e. TCP/IP, UDP, Object-Oriented message passing).

Communication transparency is achieved through the use of the *Agent Factory Registry Service (AFRS)*. This registry includes a comprehensive list of what agents exist and where they are. It is organised into a logical hierarchy of domains as seen in figure 8.

The domain-oriented view is adopted to provide a context for agent names, which must be unique within a domain. It is based on the Domain Name Service used on the Internet. The top-level domain is the Agent



**Figure 8: The Agent Factory Registry Service**

Factory (AF) domain, and additional sub-domains can be attached to it (in the figure, there are 2: UCD and NUI). Intra-domain communication requires the sender know the receivers name (e.g. “bui”); inter-domain communication requires that the sender know both the receivers name and the path of the domain in which that agent is situated (e.g. “bui@UCD.AF”). The AFRS also maintains listings of resources and services (abilities offered by agents).

## 7 A Simple Demonstrator

### 7.1 Self-Organising Networks

Mobility presents the opportunity for constructing self-organising agent communities [Mil99], where agents, distributed over a network of agent platforms are able to migrate across the network in order to

achieve tasks required of them. Migration could occur for a variety of reasons: due to an overcrowding of a platform (load balancing), to access a resource locally (where distributed access is not possible, or not preferable), or simply to gain quicker access to a resource (rather than wait in a queue for access to a resource, an agent may move to an alternative resource that is free or has a smaller waiting list).

## 7.2 The Problem

The problem presented in this paper concerns an agent, *Bui*, requiring access to a resource, *X*, to complete a task, *T*, for another agent, *Bodan*, which can proceed no further with its task until *Bui* completes *T*.

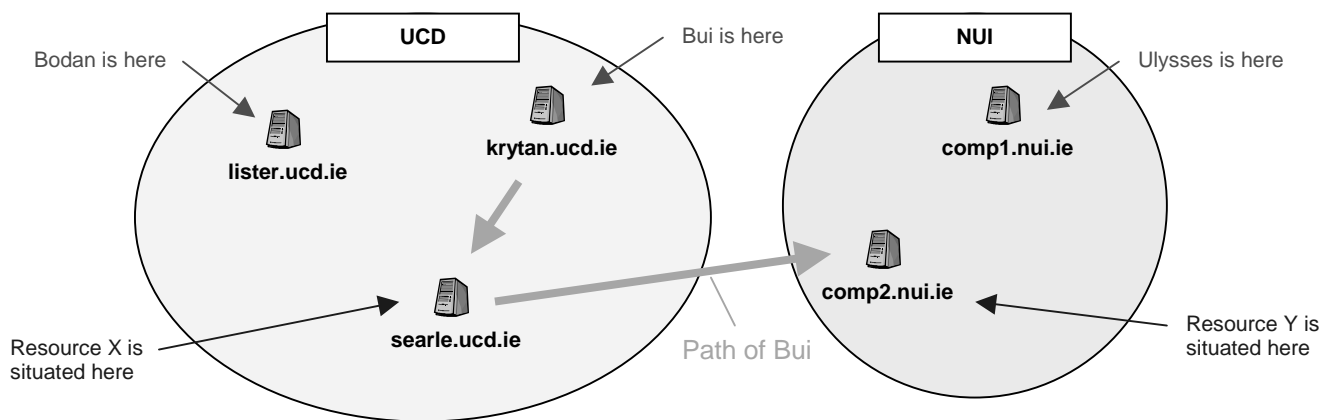


Figure 9: A Self-Organising Agent Community

In the scenario presented in figure 9 the initial state is as follows:

- *Bui* resides on its home platform *krytan.ucd.ie*.
- *Bodan* has migrated to *lister.ucd.ie*.
- *Bui* believes *X* is available on *searle.ucd.ie*.
- *X* is currently unavailable on *searle.ucd.ie*, but *Y*, which is equivalent, is available on *comp2.nui.ie*. This information is available at *krytan.ucd.ie*.
- Access to *Y* is managed by *Ulysses@comp1.nui.ie*.

In locating the resource, *Bui* must make use of the ADRS (section 6.3) to migrate between the platforms in order to access the resource. Throughout the solution to this problem, it should be recognised that *Bui* must interact transparently with both *Bodan* and *Ulysses* irrespective of its physical location. This is achieved through the use of the AFRS. The state of *Bui* at each time-step in this process is illustrated in figure 10 below.

Time	Location	Bui State	Commitment	Message Sent
t0	krytan.ucd.ie	Bel(req(X)) & Bel(at(searle.ucd.ie, X))	Com(bui, t0, migrate(searle.ucd.ie))	
t1	searle.ucd.ie	Bel(empty(X)) & Bel(at(comp2.nui.ie, Y)) & Bel(equal(X, Y)) & Bel(require(permission(Y))) & Bel(authority(Y, ulysses@nui.af))	Com(bui, t1, request(permission(Y), ulysses@nui.af))	<b>SPEECH ACT</b> (bui@ucd.af ulysses@nui.af) <b>REQUEST ACHIEVE</b> permission(Y) t1
t2	searle.ucd.ie	Bel(received(permission(Y), ulysses@nui.af))	Com(bui, t2, migrate(comp2.nui.ie))	
t3	comp2.nui.ie	Bel(available(Y))	Com(bui, t3, process(Y))	
t4	comp2.nui.ie	Bel(processed(Y))	Com(bui, t4, inform(done(T), bodan@ucd.af))	<b>SPEECH ACT</b> (bui bodan) <b>INFORM</b> <b>BELIEF</b> Bel(done(T)) t4

Figure10: State of Bui whilst achieving its task

### 7.3 Constructing the Agents in Agent Factory

The agents developed for the above problem are delivered through the development of two Agent Designs: the *MobileAgent* of section 6.2, and an *ExampleAgent* that inherits from *MobileAgent* (see figure 11). The first design introduces a *migrate* actuator and a *platform* perceptor. The perceptor introduces beliefs of the form:  $Bel(platform(X))$ , where X is the IP address of an agent platform. In addition, a commitment rule is added of the form:  $Bel(goto(X)) \Rightarrow Com(Self, Now, migrate(X))$ . Hence, if an agent believes it wants to go to platform X, then it does so. The *ExampleAgent* design extends this to include an actuator for checking the AFRS for resources, *checkResource*. Additional actuators are added for manipulating resources X and Y, and communicative actuators are inherited from a generic *Agent* design.

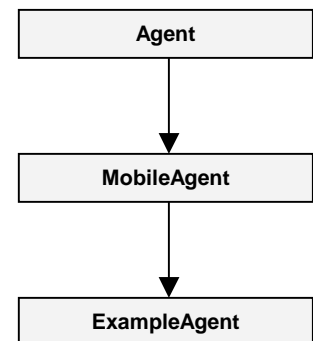


Figure 11: The Agent Design Hierarchy

## 8 Discussion

This paper delivers mobile BDI agents through an AOP framework, which delivers the BDI reasoning apparatus in a lightweight software entity. The AOP approach is advocated as a solution to the well-documented problems of the mobile agents community [Mil99]. Furthermore, this paper illustrates how this approach has been implemented within the Agent Factory System, through the development of a *mobility package*. Subsequently, some preliminary work with self-organising communities has been presented. The approach advocated here successfully ties various strands of research together in the delivery of a new generation of intelligent mobile agent.

## Acknowledgements

We gratefully acknowledge the support of Enterprise Ireland through grant number SP/97/08 IMPACT (Initiative for Mobile Programmable Agent-based Computing Technologies).

## References

- [BG88] Bond, A.H., Gasser, L. eds., **Readings in Distributed Artificial Intelligence**, *San Mateo, CA., 1988.*
- [Col96] Collier, R., **The Realization of Agent Factory: An Environment for the Rapid Prototyping of Intelligent Agents**, *M.Phil, UMIST, Manchester, UK, 1996.*
- [CO99] Collier, R.W., O'Hare G.M.P., **Agent Factory: A Revised Agent Prototyping Environment**, in 10th Irish Conference on Artificial Intelligence and Cognitive Science (AICS), 1999.
- [DLC89] Durfee, E.H., Lesser, V.R., Corkhill, D.D., **Trends in co-operative distributed problem solving**, *IEEE: Knowl. Data Eng. 11(1), 63-8, 1989.*
- [DC+99] Duffy B.R., Collier R.W., O'Hare G.M.P., Rooney C.F.B., O'Donoghue R.P.S., **SOCIAL ROBOTICS: Reality and Virtuality in Agent-Based Robotics**, *Bar-Ilan Symposium on the foundations of Artificial Intelligence: Bridging theory and practice (BISFAI), 23-25 June 1999, Ramat Gan, Israel, 1999.*
- [FLM97] Finin T., Labrou, Y., and Mayfield, J., **KQML as an Agent Communication Language**, in *Software Agents Bradshaw J. (ed.), MIT Press, Cambridge, 1997.*
- [GH+95] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., **Design patterns: elements of reusable object-oriented software**, *Addison Wesley.*
- [GH+97] Green, S., Hurst, L., Nangle, B., Cunningham, P., Somers, F., Evans, R., **Software agents: A Review**, *Technical Document TCD-CS-97-04, Computer Science Department, Trinity College Dublin, May 1997.*
- [HCK95] Harrison, C.G., Chess, D.M., Kershenbaum, A., **Mobile Agents: Are they a good idea?**, *IBM Research Report RC 19887, 1995.*
- [Jen93] Jennings, N.R., **Specification and implementation of a Belief-Desire joint intention architecture for collaborative problem solving**, in *Int. Jour. of Intel. and Co-op. Info. Sys. Vol. II no3, 1993.*
- [JVS96] Johansen, D., Van Renesse, R., Schneider, F.B., **Supporting Broad Internet Access to TACOMA**, in *Proceedings of the Seventh ACM SIGOPS European Workshop, Connemara, Ireland, pp. 55-58, September 1996.*
- [KG+97] Kotz, D., Gray, R., Nog, S., Rus, D., Chawla, S., Cybenko, G., **Agent TCL: Targeting the Needs of Mobile Computers**, in *IEEE Internet Computing, Vol. 1, No. 4, July - August 1997.*
- [KG99] Kotz, D. and Gray, R.S., **Mobile Agents and the Future of the Internet**, in *ACM Operating Systems Review, 33(3):7-13, August 1999.*
- [Mc79] McCarthy, John, **Ascribing Mental Qualities to Machines**, 1979.
- [Mil99] Milojicic, D., **Mobile Agent Applications**, in *"Trend War" column in IEEE Concurrency, July-September 1999.*
- [OC+98] O'Hare G.M.P., Collier R.W., Conlon J., and Abbas S., **Agent Factory: An Environment for Constructing and Visualising Agent Communities**, *9th Irish Conference on Artificial Intelligence and Cognitive Science (AICS), 1998*
- [OJ96] O'Hare, G.M.P. and Jennings, N.R. (Editors.), **Foundations of Distributed Artificial Intelligence**, *Wiley Interscience Pub., New York, 1996, 296 pages. ISBN 0-471-00675.*
- [PS97] Peine, H. and Stolpmann, T., **The Architecture of the Ara Platform for Mobile Agents**, in Kurt Rothermel, Radu Popescu-Zeletin (Eds.): *Proc. of the First International Workshop on Mobile Agents MA'97 (Berlin, Germany), April 7-8th. Lecture Notes in Computer Science No. 1219, Springer Verlag, ISBN 3-540-62803-7, 1997.*
- [RG91] Rao, A.S. and Georgeff, M.P., **Modelling Rational Agents within a BDI Architecture**, *Prin. of Knowl. Rep. & Reas., San Mateo, CA., 1991*
- [RO+99] C.F.B. Rooney, R.P.S. O'Donoghue, B.R. Duffy, G.M.P. O'Hare, R.W. Collier, **The Social Robot Architecture: Towards Sociality in a Real World Domain**, *Towards Intelligent Mobile Robots 99, Bristol, UK.*
- [Sho93] Shoham, Y., **Agent-Oriented Programming**, *Artificial Intelligence, Vol. 60, 51-90, Elsevier Science Publishers, 1993.*
- [Whi94] White, J.E., **Telescript Technology: The Foundation of the Electronic Marketplace**, *General Magic Inc., 1994.*