Programming Cognitive Agents  
(Invited Talk)  

John-Jules Ch. Meyer  
Institute of Information and Computing Sciences,  
Utrecht University, The Netherlands  

1 Introduction  

Although there is a lot of theory around about cognitive agents since the seminal work by researchers such as Bratman, Cohen & Levesque and Rao & Georgeff practice of programming ‘truly’ cognitive agents is still in its infancy. Of course, several architectures have been proposed and even occasionally been implemented, and there is a prospect of many potential applications of agent-based systems, but is there a truly systematic way of programming agents with cognitive / mental attitudes such beliefs, desires, intentions, goals, plans, commitments, emotions...? We believe that for this dedicated agent-oriented languages are needed. A number of these have been developed in the last decade or so. But programming in them is still hard. Is there a methodology for agent-oriented programming? Can one structure agent programs better making use of cognitive notions? And how to verify that an agent program is correct? And how is this combined with programming multi-agent systems and agent societies where co-ordination of these autonomous agents and more generally social notions such as norms seem most important? In this paper a number of the issues related to programming cognitive (multi) agents will be discussed on the basis of work done in Utrecht around the agent language 3APL.  

2 The Philosophical Origins of Cognitive Agents  

Although already in older literature, e.g. on philosophical logic, the term ‘agent’ is employed, I consider as the start of the present agent-oriented paradigm the seminal work of the philosopher Michael Bratman [7] who gives a treatment of the behaviour of a rational (human) agent in terms of mental/cognitive notions such as beliefs, desires and intentions (BDI). Since then he himself [8] and others have tried to get a more precise understanding of cognitive notions, either by trying to formalise these in some logical framework [10][37][26][46] or by devising architectures for intelligent agents [37][32].

3 Agent-Oriented Software Engineering  

Although there is not complete agreement how, it is generally recognised that by their very nature the design of programs using agents requires new software
engineering methods, techniques and tools \((47,3)\). Some say that we do need agent-related concepts (such as BDI) in the analysis and design phase of an agent system \((48)\), but for the implementation we can just use generic high-level programming languages such as JAVA without any in-built agent facilities. To the other extreme, and we are in this camp, some say that it only makes sense to analyse and design in an agent-oriented way if also the implementation is realised using agent concepts \((12)\), and to a certain extent also \((9)\).

### 3.1 Agent-Oriented Programming

In his pioneering paper \((44)\), Shoham introduced the first agent-oriented language in which agent concepts such as beliefs, commitments and commitment rules were employed. Since then there have been proposed a number of programming languages that may be called agent-oriented since they have some typical agent-like features. These include (Concurrent) METATEM \((20)\), Congolog \((22)\), and AgentSpeak(L) \((36)\). At the moment there are quite a number of agent-oriented languages such as Jason (an interpreter for AgentSpeak), JACK, Jade, Jadex \((5)\), the latter three based on JAVA, and our own 3APL \((25)\). It is my impression that at present among researchers in the field it is not yet completely clear to what extent these languages must be based on mainstream ones like JAVA and which agent concepts are really adequate or needed for 'true' agent programming. For instance, in the logical frameworks mentioned above one mostly has some notion of declarative goal while in most agent programming languages - if they have goals - goals are mostly procedural. We have tried to remedy this situation in 3APL \((13)\) by including both declarative and procedural goals. By doing this it becomes really possible to program agents such that if they have certain declarative goals they may adopt plans for these to achieve them and while executing these plans they may find reasons to revise them. We feel this is truly how an agent should behave.

### 3.2 Programming Agent Societies: How to Socialise Agents

Of course, for programming multi-agents of agent societies we need more. We need communication between agents, but, more importantly, we need to co-ordinate more or less autonomous agents! For this we can for a part draw on theory and techniques from concurrency and distributed computing, and process algebra in particular. For instance, we can adopt and adapt several communication / synchronisation mechanisms from Communicating Sequential Processes (CSP) and Concurrent Constraint Programming (CCP) to describe agent communication and co-ordination, and employ Structural Operational Semantics (SOS) to specify this in a formal way, as we have done recently in \((4)\).

Another problem related with agent communication, particularly within heterogeneous agent societies, concerns the language (ontology) agents use to reason about their beliefs and communicate with each other. Of course, if agents stem from different sources (designers) and have different tasks they will generally employ different and distinct ontologies (concepts and their representations) for