Software Agents

Camelia Chira

IDIMS Report
21st February 2003

1. Introduction

Software agents represent an important and fast growing area of Artificial Intelligence (AI) and more generally of Computer Science [Nwana 1996; Bradshow 1997; Green, Hurst et al. 1997; Jennings 2000]. The starting point for software agents was created by Multi-Agent Systems (MAS) which form one of the three research areas (see Figure 1) of a relatively youthful branch of AI – Distributed Artificial Intelligence (DAI). Therefore, agents inherit potential benefits from both DAI e.g. modularity, speed, reliability and AI e.g. operation at knowledge level, easier maintenance, reusability, platform independence [Nwana 1996].

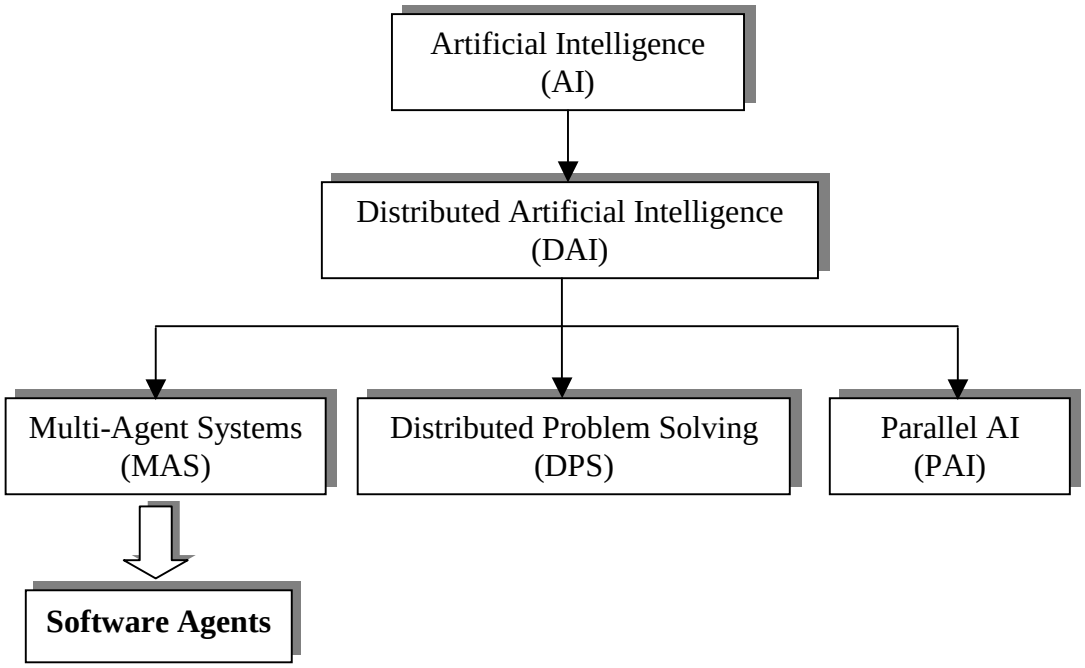

Figure 1. Agents in AI

Started in the early eighties, the research in the area of software agents and MAS evolved into what is now *"one of the most active areas of research and development activity in computing generally"* [Wooldridge and Ciancarini 2001]. Dealing with collections of interacting, coordinated knowledge-based processes [Gasser 1998], DAI demonstrates a distinct feature through the communication and coordination among intelligent and autonomous agents during a problem solving process. This approach decomposes the complexity of the domain problem (agents work together in a problem solving team as opposed to a single agent dealing with a problem) and enhances the system's performance [Chu, Srihari et al. 1996].

Based on a vast experience in using agent-based techniques, Jennings suggests the necessity of using autonomous agents for developing robust and scalable software systems. He brings two arguments to this statement [Jennings 2000]:

1. The Adequacy Hypothesis

   "Agent-oriented approaches can significantly enhance our ability to model, design and build complex, distributed software systems".

2. The Establishment Hypothesis

   "As well as being suitable for designing and building complex systems, the agent-oriented approach will succeed as a mainstream software engineering paradigm".

Jennings believes that agent-based techniques will become widely adopted since *"the agent-based approach can be viewed as a natural next step in the evolution of a whole range of approaches to software engineering"* and *"agent-based techniques are the ideal computational model for developing software for open, networked systems"* [Jennings 2000].

Also, Wooldridge and Ciancarini indicate that intelligent agents and MAS represent techniques to manage the complexity inherent in software systems. The reasons why agents are considered an important new direction in software engineering can be summarised as follows [Jennings 2000; Wooldridge and Ciancarini 2001]:

- Natural metaphor

  "Just as many domains can be conceived of consisting of a number of interacting but essentially passive objects, so many others can be conceived as interacting, active, purposeful agents".

- Distribution of data or control

  The overall control of many software systems is distributed across different computing nodes that can be geographically and temporally dispersed. *"In order to make such systems work effectively, these nodes must be capable of autonomously interacting with each other – they must be agents"*.

- Legacy systems

  "A natural way of incorporating legacy systems into modern distributed information system is to agentify them".

- Open systems

  In order to make open systems (i.e. systems for which is impossible to have all components and possible interactions among them defined at design time) work effectively, *"the ability to engage in flexible autonomous decision-making is critical"*.

Nwana splits the research and development work on software agents into two main strands as follows [Nwana 1996]:

1. Strand 1 spans the period 1977 to the current day. This strand *"concentrated mainly on deliberative-type agents with symbolic internal models"*. The emphasis is on macro issues such as *"the interaction and communication between agents, the decomposition and distribution of tasks, coordination and cooperation, conflict resolution via negotiation, etc"* and agent theories, architectures and languages.
2. Strand 2 spans the period 1990 to the current day. The emphasis is on the *"diversification in the types of agents being investigated"* which indicates that software agents are becoming mainstream [Nwana 1996; Bradshow 1997].

The second strand identified by Nwana appeared (at least partly) because *"everybody is now calling everything an agent "*. Indeed, over the last years, there has been an explosion in the use of the term "agent" without a founded motivation. This was favoured by the lack of a consensus definition for the term agent among AI researchers. Bradshow demonstrates the proliferation of many varieties of "agents" by enumerating some reasons why a number of programs are called agents e.g. some because they can be scheduled in advance to perform tasks on a remote machine, some because they perform the role of an "intelligent assistant", some because they manifest characteristics of distributed intelligence [Bradshow 1997].

This report aims to describe and define software agents and multi-agent systems by comparing various definitions and taxonomies, identifying common properties (probably differently named) proposed by different authors and reviewing agent related research. The second part presents some applications of agents and describes the notion of agent programming language.

2. Definition

Over the last years, many researchers in the area of agents and agent-based systems have offered a variety of definitions for the notion of agency. Nwana notes, *"we have as much chance of agreeing on a consensus definition for the word agent as AI researchers have of arriving at one for artificial intelligence itself"* [Nwana 1996]. This general problem in AI of defining "intelligence" led to an extensive discussion about whether *"some particular system is an agent, an intelligent agent or merely a program"* which generated as many definitions as there are researchers [Anumba, Ugwu et al. 2002]. Bradshow identifies two approaches to the definition of an agent as follows [Bradshow 1997]:

1. Agent as an ascription: this approach is based on the idea that *"agency cannot ultimately be characterized by listing a collection of attributes but rather consists fundamentally as an attribution on the part of some person"*.
2. Agent as a description: agents are defined by describing the attributes they should exhibit.

The first approach tends to define agents in a general manner offering the opportunity to many systems or components of software to be regarded as agents even though they do not present some minimal properties required by the notion of agency. Foner observes that there is *"little justification for most of the commercial offerings that call themselves agents. Most of them tend to excessively anthropomorphise the software, and then conclude that it must be an agent because of that very anthropomorphization, while simultaneously failing to provide any sort of discourse or 'social contract' between the user and the agent. Most are barely autonomous, unless a regularly-scheduled batch job counts"* [Foner 1993]. Authors in software agent technology generally agree that, even if a complete definition of the term agent is not yet possible, a good description may be given by characterising/describing the space of agent types that would result through the combination of possible attributes [Wooldridge and Jennings 1995; Franklin and Graesser 1996; Nwana 1996; Bradshow 1997]. Therefore, the second approach, i.e. agent as a description, is considered appropriate for defining software agents in the context of the current report.

Table 1 summarizes the most important definitions proposed by researchers in the area of software agents.

Author(s)	**Reference**	**Definition**
S. Russell P. Norvig	[Russell and Norvig 2003]	*An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.*
P. Maes	[Maes 1995]	*Autonomous agents are computational systems that inhabit some complex, dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks that they are designed for.*
H.S. Nwana	[Nwana 1996]	*When we really have to, we define an agent as referring to a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user.*

S. Franklin A. Graesser	[Franklin and Graesser 1996]	*An autonomous agent is a system situated within and part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.*
Y. Shoham	[Shoham 1998]	*An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments.*
N.R. Jennings M. Wooldridge	[Jennings and Wooldridge 1998] [Wooldridge 1999]	*An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.*
FIPA[1] (standard)	[Poslad, Buckle et al. 2000]	*An agent is an encapsulated software entity with its own state, behavior, thread of control, and an ability to interact and communicate with other entities – including people, other agents, and legacy systems.*

Table 1. Various definitions of an agent

Although many definitions have been proliferated, it is generally agreed in the world of research and development of software agents that an agent is characterised by the following [Nwana 1996; Wooldridge 1999; Jennings 2000]:

- An agent acts on behalf of its users.
- An agent is situated in an environment and is able to perceive that environment.
- An agent has a set of objectives and takes actions so as to accomplish these objectives.
- An agent is autonomous.

Most researchers agree that autonomy is a crucial property of an agent. Autonomous agents can take decisions without the intervention of humans or other systems. These decisions are based on the individual state and goals an agent has. An agent should act in such a manner as to pursue its internal goals. Autonomy implies that agents have control both over their internal state and over their behaviour [Wooldridge 1999; Jennings 2000].

[1] Foundation for Intelligent Physical Agents (http://www.fipa.org/) is a non-profit standard organization established in 1996, which promotes the creation of specifications of generic agent technologies.

Being embedded in a particular environment, agents receive inputs about the state of that environment through sensors and they can perform actions through effectors. Jennings et al refer to this concept using the term situatedness [Jennings 2000]. The actions of an agent will potentially affect its environment. Wooldridge refers to this ability of an agent to modify its environment through the performance of an action as the agent's effectoric capability [Wooldridge 1999]. An action has a set of associated pre-conditions that specify the possible situations when it can be performed. Wooldridge exemplifies this with an action 'lift table' which will succeed only if the weight of the table is sufficiently small that the agent can lift it. Figure 2 shows the interaction between an agent and its environment, which is usually an ongoing and non-terminating one.

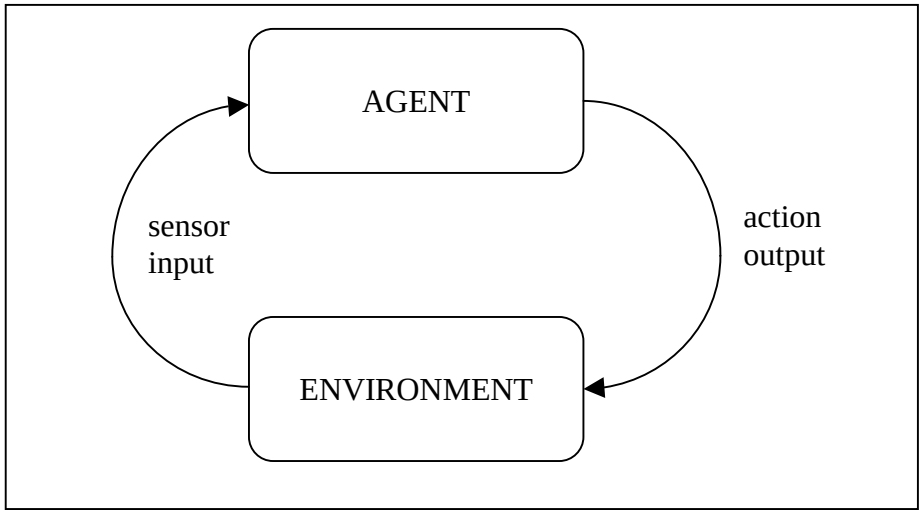

Figure 2. An agent in its environment [Wooldridge 1999]

Table 2 presents different types of environments that an agent can occupy based on various environment properties [Wooldridge 1999; Russell and Norvig 2003]. The everyday physical world can be regarded as an inaccessible, non-deterministic, non-episodic, highly dynamic and continuous environment, which is in fact the most complex general class of environments [Russell and Norvig 2003]. In most environments, an agent will not have complete control but will enjoy partial control over its environment. Therefore, the same action may fail to have the desired effect even though performed in apparently identical circumstances [Wooldridge 1999].

Classification	Explanation	Greater problem for the agent designer
Accessible vs. Inaccessible	Agents can obtain complete, accurate, up-to-date information about the environment's state in an accessible environment. Most complex environments are inaccessible.	Inaccessible Environment
Deterministic vs. Non-deterministic	Any action of the agent has a single guaranteed effect in a deterministic environment as opposed to some uncertainty about the resulting state after an action is performed in a non-deterministic environment.	Non-deterministic Environment
Episodic vs. Non-episodic	The performance of an agent depends on a number of discrete episodes in an episodic environment. There is no link between the performance of an agent in different scenarios.	Non-episodic Environment
Static vs. Dynamic	A static environment can be changed only by the performance of the actions of the agent while a dynamic environment has other processes operating in it, which are not under the control of the agent.	Dynamic Environment
Discrete vs. Continuous	There are a fixed, finite number of actions and percepts in a discrete environment.	Continuous Environment

Table 2. Classification of environment properties [Russell and Norvig 2003]

Other than autonomy, many researchers consider that an agent should also be characterised by one or more of the following properties [Nwana 1996; Wooldridge 1999]:

- *Reactivity*

  An agent is situated in an environment and is able to perceive this environment and to respond to changes that occur in it (reactive behavior).
- *Pro-activeness*

An agent should have the ability to take the initiative in order to pursue its individual goals (goal-directed behavior).

- *Cooperation*

  An agent should have the capability of interacting with other agents and possibly humans via an agent-communication language.

Nwana considers pro-activeness a key element of an agent's autonomy [Nwana 1996] while Wooldridge and Ciancarini list autonomy and pro-activeness as two separate properties an agent should have [Wooldridge and Ciancarini 2001]. Furthermore, these properties are more challenging than they seem. Agents should attempt to achieve their goals but not continuously: that is, agents should cancel actions when it is clear that those actions will not work (because of some factors that modified the environment for example) or when the goal of the action is not longer valid. In such a situation, reactivity should be demonstrated: the agent should react to the events that occur in its dynamic environment. While pro-activeness (in a system that exhibits goal-directed behavior) and reactivity (in a purely reactive system) can be easily implemented independently, integrating goal-directed and reactive behavior within a system turns out to be very difficult. This problem of achieving an effective balance between pro-activeness and reactivity represents one of the key problems of the agent designer and is basically still open to discussion [Wooldridge and Ciancarini 2001].

The third property of an agent i.e. cooperation involves the ability of an agent to dynamically negotiate and coordinate [Wooldridge 1999]. Nwana considers cooperation to be the reason for having multiple agents situated in an environment instead of having just one agent in an environment. Because of their social ability, agents can cooperate with other agents and humans. However, Nwana notes that coordination among different agents is possible without cooperation [Nwana 1996].

Wooldridge and Ciancarini use these properties in their definition of an agent: *"By an agent, we mean a system that enjoys the following properties: autonomy […], reactivity […], pro-activeness […], social ability […]"* [Wooldridge and Ciancarini 2001]. Therefore, these properties i.e. autonomy, reactivity, pro-activeness and cooperation are not optional but instead they *define* the term agent.

Jennings et al include three key concepts in their definition of an agent: situatedness, autonomy and flexibility. The latter concept is defined using three properties as follows: reactivity (an agent should be responsive), pro-activeness and social ability [Jennings, Sycara et al. 1998].

Some researchers in the area of agents and MAS add a number of other properties to characterise agents as follows [Franklin and Graesser 1996; Nwana 1996; Bradshow 1997]:

- *Learning*

  An agent should have the ability to learn while acting and reacting in its environment.

- *Mobility*

  A mobile agent has the ability to move around a network (even from one platform to another) in a self-directed way.

- *Temporal continuity*

  The actions of an agent are performed through a continuous running process (over long periods of time).

- *Personality*

  An agent should manifest a believable character and emotional state.

The ability of an agent to learn can increase performance over time. Nwana considers that a truly intelligent agent (the ideal agent) should equally be characterised by three primary attributes i.e. autonomy, cooperation and learning while any system that does not exhibit these three properties (more or less emphasized) should not be considered an agent at all [Nwana 1996].

Table 3 summarizes all the properties associated with the notion of agency and provides an explanation for each.

Property	**Other name(s)**	**Explanation**	**Comments**
Autonomy		An agent can operate on its own without the intervention of humans or other systems.	Generally accepted
Reactivity	Situatedness *or* Sensing and acting	An agent perceives its environment: it receives input from the environment and is able to change the environment by performing some actions.	
Pro-activeness	Goal-directed behavior	An agent has the ability to take the initiative in order to accomplish its design objectives.	Considered by Nwana a key element of autonomy [Nwana

			1996]
Cooperation	Communication *or* Social Ability	An agent is capable of interacting with other agents and/or humans in order to accomplish its design objectives.	Viewed by most researchers as a crucial attribute of an intelligent agent.
Flexibility		Defined by Wooldridge and Jennings using three other properties i.e. reactivity, pro-activeness and cooperation [Jennings, Sycara et al. 1998] [Wooldridge 1999]	Flexibility is not a new property but instead incorporates three properties already defined.
Learning	Adaptivity	An agent can learn and improve with experience.	Considered by Nwana a key attribute of an intelligent agent.
Mobility		An agent has the ability to move from one machine to another in a network.	
Temporal continuity		An agent persists over long periods of time.	
Personality	Character	An agent demonstrates a "believable" personality and emotional state.	

Table 3. Properties of an agent

Wooldridge (as well as other researchers) indicates that the various properties of an agent (described above) are of differing importance for different domains. Therefore, learning can be considered very important for some applications while others may consider it really detrimental [Wooldridge 1999].

Table 4 presents the definition of an agent based on a list of the properties the agent should have proposed by different authors.

Author	Reference	Properties used in the definition of an *agent*
H.S. Nwana	[Nwana 1996]	Autonomy (which includes pro-activeness) Cooperation Learning
S. Franklin A. Graesser	[Franklin and Graesser 1996]	Autonomy Reactivity Pro-activeness Temporal Continuity
N.R. Jennings M. Wooldridge	[Jennings and Wooldridge 1998] [Wooldridge 1999]	Autonomy Reactivity Pro-activeness } Flexibility Social Ability

Table 4. Various definitions of an agent using a list of properties

Ndumu and Nwana use an abstract context diagram to represent an agent from a conceptual point of view (see Figure 3). A generic agent is composed of three main layers as follows [Ndumu and Nwana 1996]:

- The *definition layer* identifies the agent as an autonomous rational entity.
- The *organisation layer* defines the agent's relationships with other agents.
- The *coordination layer* specifies the social abilities of the agent (e.g. known coordination/negotiation techniques - see section 5).

The other two layers included in this abstract structure of an agent, i.e. the communication layer and the API (Application Program Interface) layer, play a less important role [Ndumu and Nwana 1996].

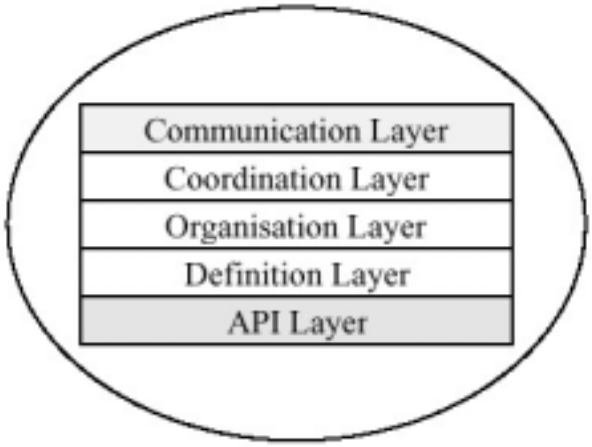

Figure 3. A layered abstraction of a generic agent [Ndumu and Nwana 1996]

To summarise, a software agent is a computer system situated in an environment that acts on behalf of its user and is characterised by the properties enumerated in the above table i.e. autonomy, cooperation, reactivity, pro-activeness, temporal continuity and learning. Autonomy is definitely the most important property of an agent without which the notion of agency would not exist. Furthermore, cooperation among different software agents may be very useful in achieving the objectives an agent has. The ideal is an agent characterised by all the above-mentioned properties but the design and implementation of such an agent is yet a very difficult and complex task.

3. Typology

As indicated in the previous section, the term "agent" is an elusive one. Psychology, sociology, economics and AI (and Computer Science more generally) are using it with equivalent meanings but through different perspectives. There are several classification schemes or taxonomies proposed in the agent research community from which the following three are well acknowledged:

1. Gilbert's scope of intelligent agents [Bradshow 1997]
2. Nwana's primary attribute dimension typology [Nwana 1996]
3. Franklin and Graesser's agent taxonomy [Franklin and Graesser 1996]

Figure 4 presents Gilbert's scope of intelligent agents [Bradshow 1997].

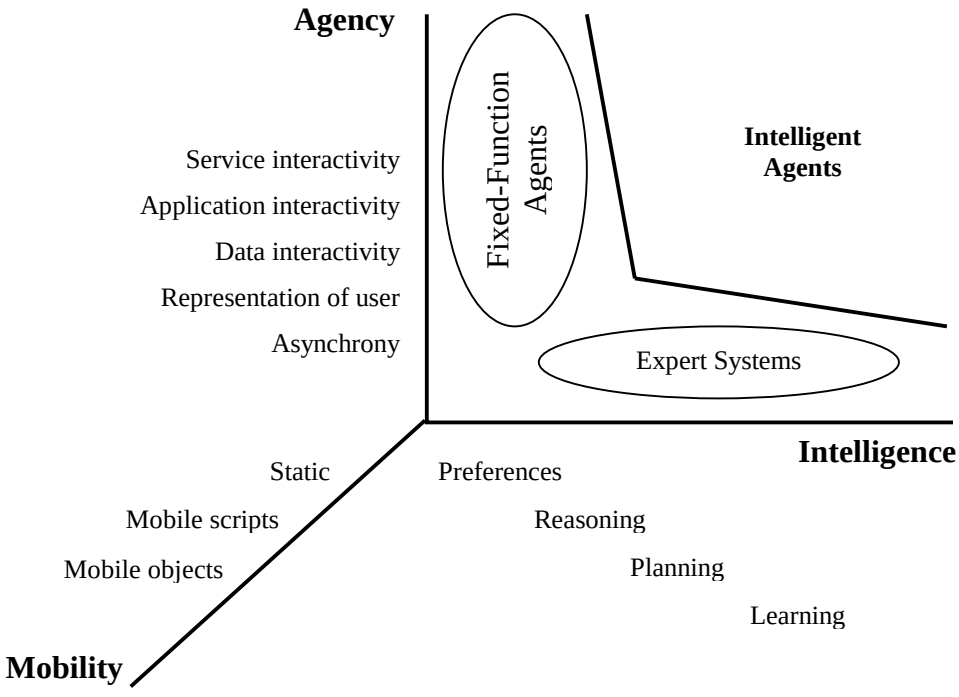

Figure 4. Scope of intelligent agents (adapted from Gilbert et al. by [Bradshow 1997])

Gilbert at al described intelligent agents using the following three dimensions [Bradshow 1997]:

- ***Agency*** *"is the degree of autonomy and authority vested in the agent, and can be measured at least qualitatively by the nature of the interaction between the agent and other entities in the system. At minimum, an agent must run asynchronously. The degree of agency is enhanced if an agent represents a user in some."* (Gilbert et al. 1995 as cited by [Bradshow 1997]).
- ***Intelligence*** is the degree of reasoning and learned behaviour. Furthermore, intelligent agents should learn and adapt to their environment in terms of the user's objectives and the resources available.
- ***Mobility*** is the degree to which the agents travel through the network.

Nwana uses the three minimal characteristics an agent should exhibit i.e. autonomy, cooperation and learning to classify agents in four categories as follows (see Figure 5) [Nwana 1996]:

- *Collaborative agents*

  There is more emphasis on cooperation and autonomy than on learning
- *Collaborative learning agents*

  There is more emphasis on cooperation and learning than on autonomy.
- *Interface agents*

  There is more emphasis on autonomy and learning than on cooperation.
- *Smart agents*

  These agents implement all three properties equally.

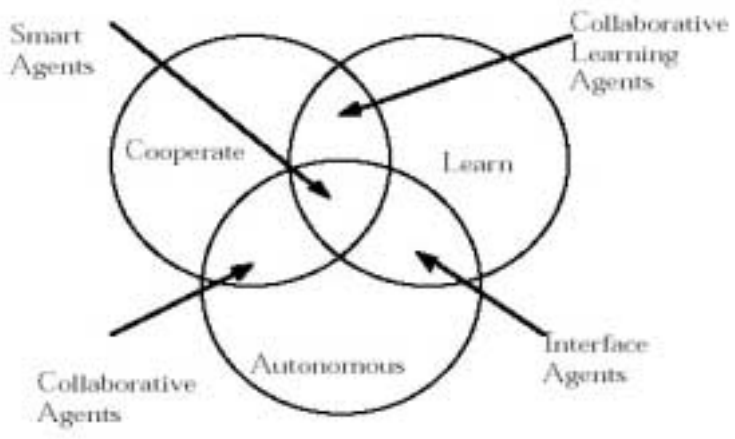

Figure 5. Nwana's agent typology [Nwana 1996]

Mobility can also be used to classify agents in *static* or *mobile* while the presence of a symbolic reasoning model results in *deliberative* or *reactive* agents. Combining these types of agents with the ones already identified based on the ideal and primary attributes of an agent (as considered by Nwana) can produce other categories of agents such as static deliberative collaborative agents, mobile reactive collaborative agents, static deliberative interface agents, mobile reactive interface agents, etc. Another classification proposed by Nwana uses the roles of agents and is exemplified with *information* or *internet* agents. This category of agents manages large databases in wide area networks like the internet. The last category of agents identified by Nwana is *hybrid* agents, which combine two or more agent philosophies. Furthermore, Nwana uses these agent typologies to identify only seven types of agents as follows (see Figure 6) [Nwana 1996]:

1. *Collaborative agents* are "able to act rationally and autonomously in open and time-constrained multi-agent environments".
   Key characteristics: autonomy, social ability, responsiveness and pro-activeness.
2. *Interface agents* support and assist the user when interacting with one or more computer applications by learning during the collaboration process with the user and with other software agents.
   Key characteristics: autonomy, learning (mainly from the user but also from other agents), and cooperation with the user and/or other agents.
3. *Mobile agents* are autonomous software programs capable of roaming wide area networks (such as WWW) and cooperation while performing duties (e.g. flight reservation, managing a telecommunications network) on behalf of its user.
   Key characteristics: mobility, autonomy and cooperation (with other agents – for example, to exchange data or information).
4. *Information/Internet agents* are designed to manage, manipulate or collate the vast amount of information available from many distributed sources (information explosion). These agents "have varying characteristics: they may be static or mobile; they may be non-cooperative or social; and they may or may not learn".
5. *Reactive agents* act/respond to the current state of their environment based on a stimulus-response scheme. These agents are relatively simple and interact with other agents in basic ways but they have the potential to form more robust and fault tolerant agent-based systems.
   Key characteristics: autonomy and reactivity.

6. *Hybrid agents* combine two or more agent philosophies into a single agent in order to maximise the strengths and minimise the deficiencies of the most relevant techniques (for a particular purpose).
7. *Smart agents* are equally characterised by autonomy, cooperation and learning.

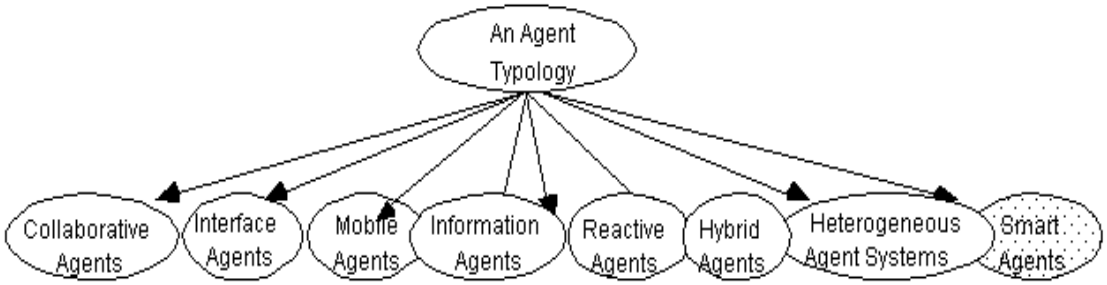

Figure 6. A classification of software agents proposed by Nwana [Nwana 1996]

Heterogeneous agent systems are obtained by combining agents from two or more of these categories. Unlike hybrid agent architectures, this agent category refers to an integrated set-up of at least two or more types of agents (including hybrid agents). Agent-based software engineering facilitates the interoperation of miscellaneous software agents. An agent communication language is necessary for the communication process among different agents [Nwana 1996]. This category of agent systems is generally referred to (by most researchers) as multi-agent systems and is discussed in more detail in the next section of this report.

Franklin and Graesser proposed the taxonomy of autonomous agents presented in Figure 7 [Franklin and Graesser 1996].

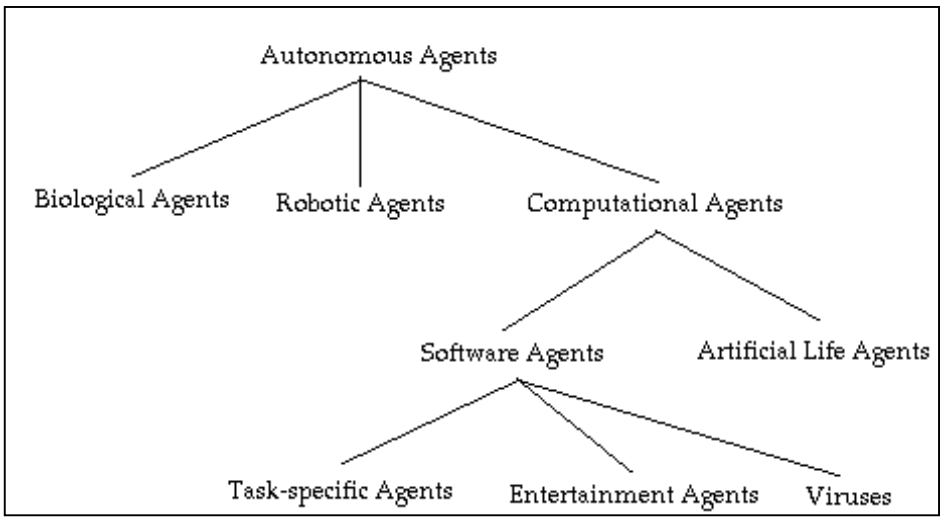

Figure 7. The taxonomy of agents proposed by Franklin and Graesser [Franklin and Graesser 1996]

This taxonomy includes biological, robotic and computational agents at the kingdom level, software agents and artificial life agents at the phylum level and task-specific agents, entertainment agents and computer viruses at the class level. A further taxonomy can be performed using schemes such as classification via the agent's control structures (e.g. regulation, planning and adaptive), via environments (e.g. database, file system, network, internet), via languages (in which the agent is written) and via applications. These subclassification schemes provide a collection of features for an agent and therefore a possible category of classification [Franklin and Graesser 1996].

Figure 8 presents a comprehensive list of classifying properties of agents [Horn, Kupries et al. 1999], from which the agent taxonomy is straightforward.

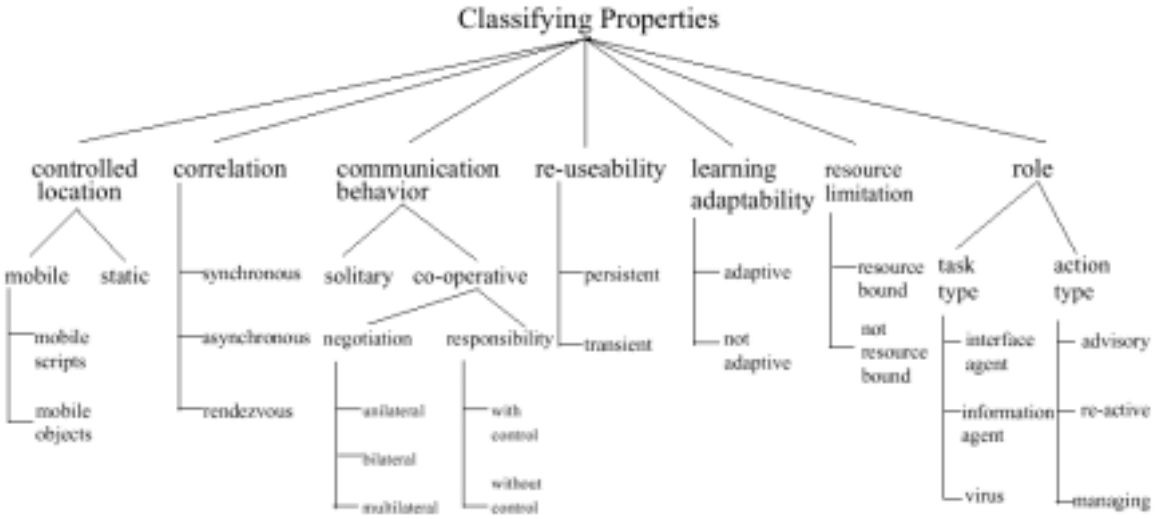

Figure 8. The classification of software agents proposed by Horn et al [Horn, Kupries et al. 1999]

Horn et al use some classifying properties such as locality affiliation, role, service capacity, communication behaviour, negotiation ability, delegation ability, correlation, learning adaptability, resource limitation and re-usability to categorize agents. These properties are partly interdependent.

4. Agent Architectures

Agent architectures address the issues of designing and creating computer-based systems that satisfy the agent properties (proposed by agent theorists) e.g. autonomy, reactivity, pro-activeness, social ability [Wooldridge 1998]. *"An agent architecture is essentially a map of the internals of an agent – its data structures, the operations that may be performed*

on these data structures, and the control flow between these data structures" [Wooldridge 1999]. Wooldridge and Jennings indicate that agent architectures can be viewed as software engineering models of agents [Wooldridge and Jennings 1995]. They identify the following classes of agent architectures [Wooldridge and Jennings 1995; Wooldridge 1999]:

1. Deliberative (or symbolic) architectures
2. Reactive (or behavioural or situate) architectures
3. Hybrid architectures

Deliberative architectures adopt the traditional AI (called symbolic AI) approach to designing intelligent systems by viewing them as a type of knowledge-based system. Wooldridge defines a deliberative agent as one that *"contains an explicitly represented, symbolic model of the world"* and *"makes decisions (for example about what actions to perform) via symbolic reasoning"* [Wooldridge 1999]. The agent-based system that has to be designed receives a symbolic representation of its environment and its desired behaviour, which can be syntactically manipulated. Figure 9 shows how deliberative agents adopt the sense-plan-act problem-solving paradigm of classical AI planning systems [Helin 2003].

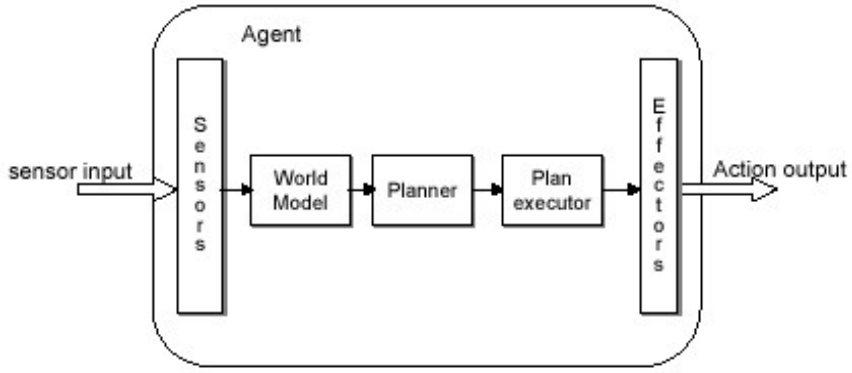

Figure 9. The basic architecture of a deliberative agent [Helin 2003]

The disadvantages associated with deliberative architectures can be summarised as follows [Wooldridge 1999; Helin 2003]:

- *The transduction problem*

  It is time consuming to translate the information into its symbolic representation. The assumption of calculative rationality (i.e. *"the assumption that the world will not change in any significant way while the agent is deciding what to do, and that an action which is rational when decision making begins will be rational when it*

concludes" [Wooldridge 1999]) might result in an ineffective operation of agents in time-constrained environments.

- *The representation/reasoning problem*

  This problem refers to representing and reasoning about complex, dynamic, possibly physical environments (so as to achieve useful results).

Logic based agents are identified by Wooldridge as those agents that use logical formulae as the symbolic representations and logical deduction or theorem proving as the syntactic manipulation. In this approach to building agents, decision making is realized via logical deduction. The program of an agent is encoded as logical theory and the selection of an action is reduced to a problem of proof [Wooldridge 1999].

Much of the research and developments work on deliberative agents has focused on the agent-oriented programming paradigm [Wooldridge 1998]. The state of an agent is characterised in terms of its mental attitudes of belief, desire and intention [Rao and Georgeff 1995]. Agent-oriented programming uses these intentional notions to directly program agents. Shoham developed an experimental language called AGENT0 [Shoham 1998] in order to demonstrate the agent-oriented programming paradigm. Other examples of logical approaches to agent programming include the ConGolog [Giacomo, Lespérance et al. 2000] and the Concurrent METATEM [Fisher 1994] programming languages. These agent languages are all described later in this report.

Reactive architectures are an alternative to the symbolic AI paradigm. They involve developing and combining individual behaviours of reactive agents situated in some environment [Wooldridge 1999]. Reactive agents have a very simple representation of the world but provide tight coupling of perception and action. The behaviour-based paradigm informs the reactive approach to building agents. Each individual behavior continually maps perceptual input to action output. Figure 10 presents the basic architecture of a reactive agent [Helin 2003].

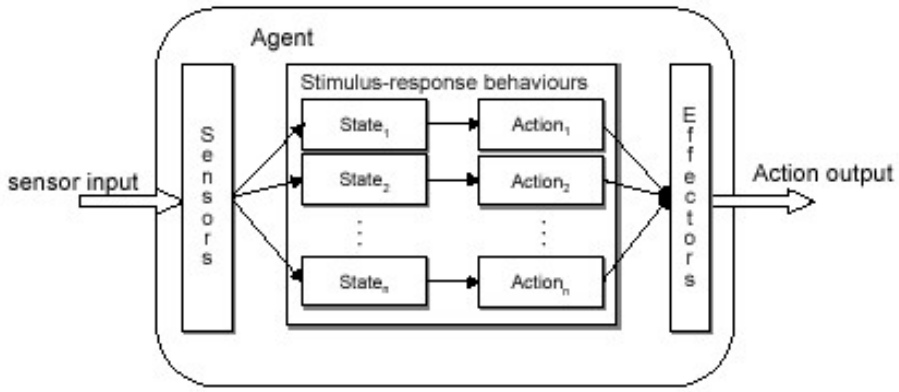

Figure 10. The basic architecture of a reactive agent [Helin 2003]

In the reactive approach, intelligent behaviour emerges from the interaction of various simpler behaviours as well as from the interaction between an agent and its environment. The main disadvantage of this architecture relates to the fact that agents do not employ models of their environment. This means that they need a great deal of local information to determine an acceptable action. Decision making is realised in the agent's local environment without necessarily taking into account non-local information [Wooldridge 1999]. A well-known example of reactive agent architecture is the subsumption architecture developed by Brooks [Brooks 1986].

Hybrid architectures combine the deliberative and reactive approaches (see Figure 11). An agent consists of several subsystems that manifest characteristics of both deliberative and reactive approaches as follows [Helin 2003]:

- *Deliberative component*: subsystems develop plans and make decisions using symbolic reasoning
- *Reactive component*: subsystems are able to react quickly to events without complex reasoning.

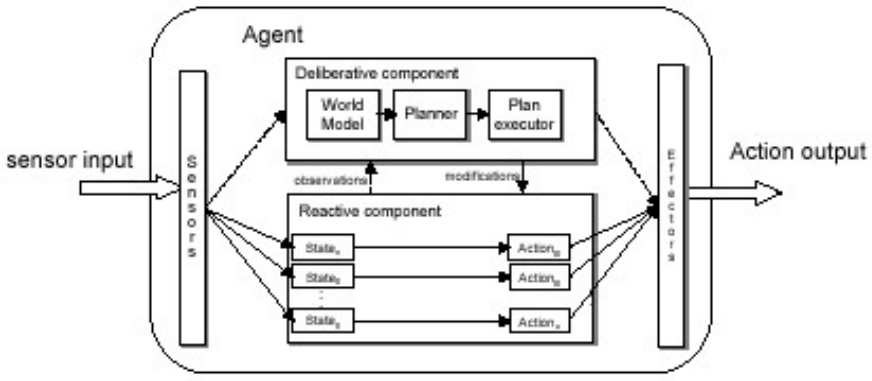

Figure 11. The basic architecture of a hybrid agent [Helin 2003]

A popular approach to the design of hybrid agents is the use of *layered architectures* [Wooldridge 1998]. The various subsystems of the architecture are arranged into a hierarchy of interacting layers each of which is reasoning about the environment at different levels of abstraction. Two types of information and control flow have been identified within layered architectures i.e. horizontal and vertical. In horizontally layered architectures, each layer is directly connected to the sensory input and action output (see Figure 12).

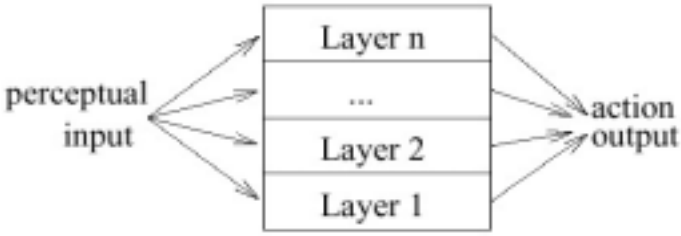

Figure 12. Control flow in horizontally layered agent architecture [Wooldridge 1999]

Acting like an agent, each layer produces action suggestions. However, this means that the layers are competing with one another creating the *"danger that the overall behaviour of the agent will not be coherent"* [Wooldridge 1999]. The main advantage of this approach is the inherent conceptual simplicity allowing an agent to exhibit many different types of behaviour. In vertically layered architectures, at most one layer is connected to the sensory input and action output (see Figure 13).

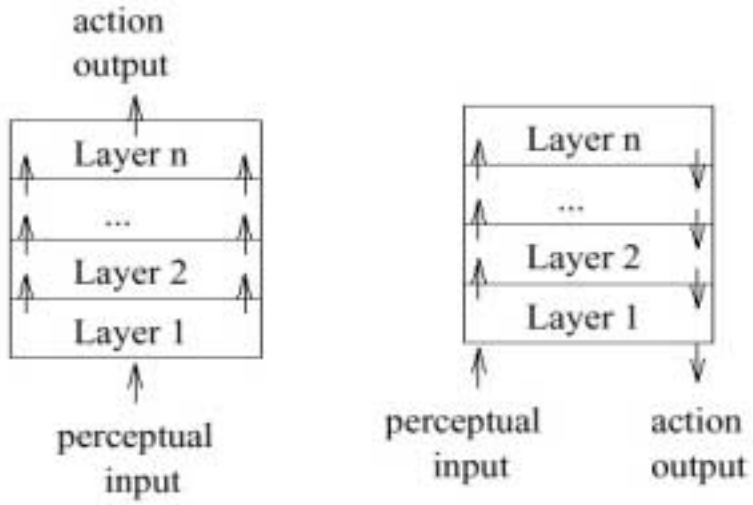

Figure 13. Control flow in vertically layered agent architecture (one pass control and two pass control) [Wooldridge 1999]

These architectures can be further classified into one pass architectures (information and control flows sequentially through each layer) and two pass architectures (information flows up through each layer and then control sequentially flows down). The lack of flexibility is the main disadvantage of vertical layering: control must flow through each different layer before a decision is made [Wooldridge 1999]. Some examples of layered architectures include the TouringMachines [Ferguson 1992] and INTERRAP [Muller and Pischel 1993] architectures.

Another well-known agent architecture is the Procedural Reasoning System (PRS) [Ingrand, Georgeff et al. 1992] based on the *belief-desire-intention* (BDI) model [Rao and

Georgeff 1995]. Inspired by the philosophical tradition of understanding practical reasoning, BDI architectures have become very popular over the last years [Georgeff, Pell et al. 1999; Wooldridge 1999]. Moreover, Wooldridge lists BDI and layered architectures as two separate classes of concrete agent architectures. Furthermore, hybrid agents are omitted from the agent architecture typology and the following four classes of agents are considered by Wooldridge [Wooldridge 1999]:

- Logic-based agents
- Reactive agents
- BDI agents
- Layered architectures

Indeed, most agents fall in one of these four categories from an architectural point of view [Jennings, Sycara et al. 1998; Devedzic 2001].

The BDI architecture represents an agent in terms of its beliefs, desires (or goals) and intentions. The basic components of a BDI agent are data structures (that represent beliefs, desires and intentions) and functions for representing and reasoning about them. As shown in Figure 14, there are seven key components of a generic BDI architecture as follows [Wooldridge 1999]:

- The agent's *beliefs* correspond to the information the agent has about the environment it occupies.
- The *belief revision function (brf)* determines new beliefs based on a perceptual input and the agent's current beliefs.
- The agent's *desires* correspond to the available actions (intuitively, allocated tasks).
- The agent's *intentions* correspond to those desires to which the agent has committed to achieving (the agent's current focus).
- The *option generation* function determines the agent's desires based on the agent's current beliefs and intentions.
- The *filter* function determines the agent's intentions based on the agent's current beliefs, desires and intentions.
- The *action* selection function determines the action to be performed based on the agent's current intentions.

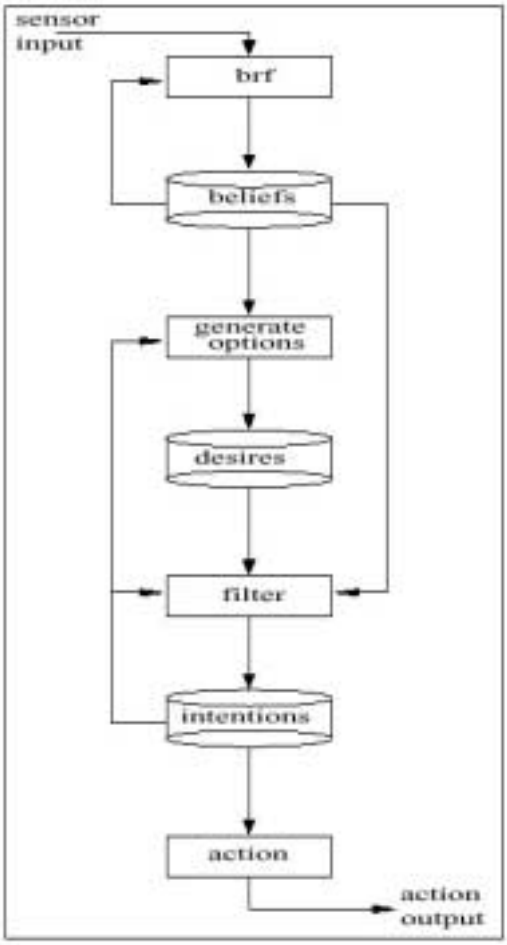

Figure 14. A generic architecture of a BDI agent [Wooldridge 1999]

The BDI architecture presents some attractive benefits such as intuitiveness and clear functional decomposition [Wooldridge 1999]. Another positive aspect is that many researchers focused on the formalisation of the BDI model. Rao and Georgeff developed a family of BDI logics for the formal semantics of BDI architectures, which are mainly based on possible relationships between the three mental components of BDI agents or on proof methods for restricted forms of the logics [Rao and Georgeff 1995]. An important problem in BDI architectures is that of *"striking a balance between being committed to and overcommitted to one's intentions: the deliberation process must be finely tuned to its environment, ensuring that in more dynamic, highly unpredictable domains, it reconsiders its intentions relatively frequently – in more static environments, less frequent reconsideration is necessary"* [Wooldridge 1999]. Also, the BDI model should consider systems that must learn and adapt their behaviour, which are becoming more and more important [Georgeff, Pell et al. 1999].

5. Multi-Agent Systems

As noted in the first section of this report, systems composed of multiple agents are studied under the banners of Multi-Agent Systems (MAS) and Distributed Problem Solving (DPS), the two main fields of Distributed Artificial Intelligence [Green, Hurst et al. 1997; Sen 1997; Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999]. A DPS system incorporates interaction strategies in order to solve a particular problem through cooperation (by dividing and sharing knowledge about the problem) among different modules. On the other hand, MAS researchers study the behavior of a group of autonomous agents (possibly pre-existing), which are working together towards a common goal. Including several interacting agents, MAS systems represent a great potential of agent-based systems. An *agent-based system* can be defined as *"one in which the key abstraction used is that of an agent"* [Wooldridge and Ciancarini 2001] meaning that a single agent can form an agent-based system [Jennings, Sycara et al. 1998]. MAS systems are ideal for solving complex problems for which some or all of the following apply [Jennings, Sycara et al. 1998]:

- Multiple problem solving methods
- Multiple perspectives
- Multiple problem solving entities

The increasing interest in MAS research is motivated by many potential advantages including the following [Bradshow 1997; Green, Hurst et al. 1997; Gasser 1998; Jennings, Sycara et al. 1998; Martin, Plaza et al. 1998]:

- MAS systems provide robustness, efficiency, flexibility, adaptivity and scalability.
- MAS systems allow inter-operation of multiple existing legacy systems (e.g. expert systems, decision support systems).
- MAS systems have the ability to solve problems that are too large or complex for a single centralised agent
- MAS systems can cope with domains in which data, expertise, or control is distributed (e.g. in domains such as distributed sensing, medical diagnosis or air-traffic control, knowledge or activity is inherently distributed).
- MAS systems can enhance speed, reliability and extensibility.
- MAS systems offer conceptual clarity and simplicity of design.

Jennings et al define the term MAS as a *"loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver"* [Jennings, Sycara et al. 1998]. The problem solvers from this definition are autonomous and possibly heterogeneous agents. A MAS system is

characterised by the following [Green, Hurst et al. 1997; Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999; Lazansky, Stepankova et al. 2001]:

- A MAS system consists of a collection of agents.
- Each agent acts autonomously.
- The agents in a MAS system are able to interact in order to reach an overall goal.
- Each agent has a limited set of problem solving capabilities.
- There is no global system control.
- Data is decentralized.
- Computation is asynchronous.

It is clear from the definition and main characteristics of a MAS system that inter-operation among autonomous agents is essential to successfully find a solution to a given problem. Agent-oriented interactions include simple information interchanges as well as planning of interdependent activities for which cooperation, coordination and negotiation are fundamental. Jennings notes that these agent interactions differ from those that occur in other computational models from two perspectives. Firstly, an agent knows which goals should be followed and, therefore, agent-oriented interactions are conceptualised as taking place at the knowledge level. Secondly, agents are flexible entities in an environment over which they have partial control and, therefore, they have to make run-time decisions about their interactions that were not foreseen at design time. Jennings also identifies the organisational relationships inherent in an agent-based system because agents act towards a goal on behalf of individuals/companies or as part of a wider problem solving initiative. The organisational context needs to be explicitly represented as it defines the nature of the relationships between agents and influences their behaviour [Jennings 2000]. All these concepts are represented by Jennings using a canonical view of an agent based system (see Figure 15).

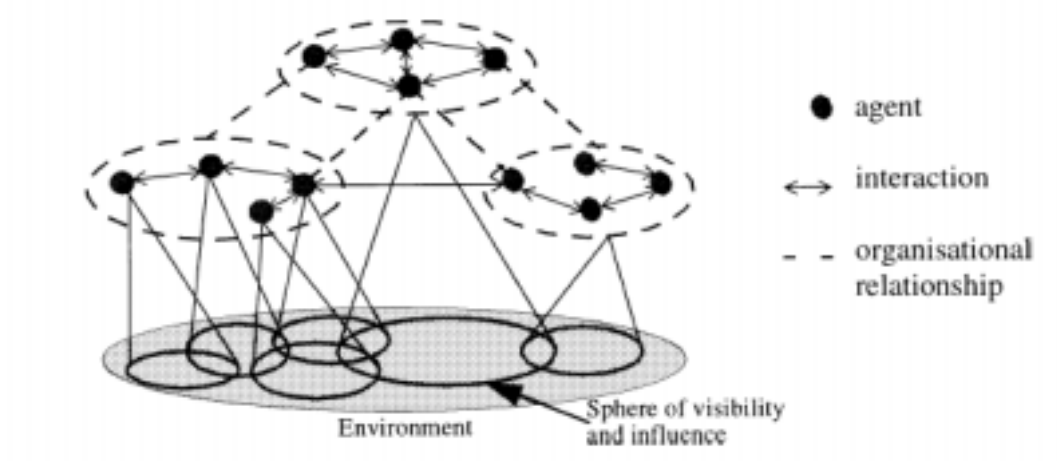

Figure 15. Canonical view of an agent-based system [Jennings 2000]

Depending on the degree of cooperation demonstrated by individual agents, two types of MAS systems have been identified by researchers as follows [Green, Hurst et al. 1997]:

1. *Cooperative Multi-Agent Systems (CMAS)*

   The general performance of the system is important and, therefore, all agents in the system act cooperatively in an appropriate manner. The designer of such a system is not concerned with the performance of individual agents.

2. *Self-Interested Multi-Agent Systems (SMAS)*

   Interested only on the benefit derived from individual agents, independent designers implement individually motivated agents. Such agents are considered self-interested, competitive or non-cooperative.

The situation of total cooperation known as the *benevolent agent assumption* is generally accepted in DAI research but agents may have conflicting goals resulting in this cooperative to antagonistic spectrum in a MAS system [Green, Hurst et al. 1997]. Cooperation is the primary element in a CMAS while negotiation is seen as the method for coordination and conflict resolutions in SMAS [Jennings, Sycara et al. 1998].

Although MAS systems present many potential benefits, some inherent problems to the design and implementation of such systems have also been identified by researchers. These problems, which have intertwined solutions, can be summarised as follows [Gasser 1998; Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999]:

- Formulation, description, decomposition and allocation of the problem and synthesis of the results among a group of intelligent agents.
- Communication and interaction among agents: what communication languages or protocols to use in order to enable a meaningful agent interaction, and what and when to communicate?
- Coordination among agents: how to enable individual agents to reason about the actions, plans, strategies and beliefs of other agents and about their coordinated process?
- Identification and reconciliation of disparate viewpoints and conflicts among agents trying to coordinate their actions.
- Balance of local computation and communication: how to avoid computational overload by the means of load balancing strategies?
- Implementation of a MAS system: how to engineer and construct practical MAS systems and what are the technology platforms and development methodologies to support MAS design and implementation?

- Verification and correction of MAS applications using formal and practical approaches.

Since agents in a MAS system have to exchange information and knowledge in order to solve a problem coherently, the following areas have become of crucial importance in MAS research [Green, Hurst et al. 1997]:

- *Coordination*

  The agents in a MAS system must coordinate their activities (to determine the organisational structure in a group of agents and to allocate tasks and resources).
- *Negotiation*

  Agents must negotiate if a conflict occurs.
- *Communication*

  Any agent in a MAS system must be able to communicate with other agents. An agent communication language (ACL) enables agents to collaborate with each other providing them with the means of exchanging information and knowledge [Labrou, Finin et al. 1999].

Coordination

Considered a central issue to MAS research, coordination has also been studied by researchers in various areas such as organisation theory, political science, social psychology, anthropology, law and sociology. Agent-oriented interaction would be ineffective without a valuable coordination among cooperative agents working together towards a common goal. Nwana et al define coordination as *"a process in which agents engage in order to ensure a community of individual agents acts in a coherent manner"* [Nwana, Lee et al. 1996]. Coordination is considered an essential aspect of MAS systems for several reasons as follows [Nwana, Lee et al. 1996; Green, Hurst et al. 1997]:

- Coordination prevents anarchy or chaos during conflicts. Such a situation is possible because each agent has a partial view over its environment and therefore, its actions might interfere with rather than support other agents' actions.
- Agents' behaviours have to be coordinated to meet global constraints e.g. a MAS system constructing a design has to work within the constraints of a pre-specified budget.
- Coordination is necessary because agents in a MAS system have different and limited capabilities and expertise (distributed expertise, resources or information).
- Interdependent activities require coordination (an agent's action might depend on the completion of another agent's task).

- Coordination enables efficiency. One agent can discover information that is of sufficient use to another agent even if their activities are independent.

Nwana et al indicate that coordination may require *cooperation* (although coordination can also occur without cooperation) but cooperation among agents does not necessarily result in coordination. Also, *communication* among agents may be required for coordination but agents can also be coordinated without communication via organisation provided they possess models of each other's behaviours [Nwana, Lee et al. 1996]. Furthermore, coordination does not imply reciprocation since an agent can coordinate its activities with those of another agent unaware of its presence [Durfee 2001].

Researchers have proposed various coordination techniques as follows [Nwana, Lee et al. 1996; Green, Hurst et al. 1997; Oliveira, Fischer et al. 1999]:

- Organisational structuring
- Contract Net Protocol (CNP)
- Multi-agent planning
- Social laws
- Computational market-based mechanisms

Organisational structuring is the simplest coordination technique and exploits the a priori organisational structure: the system of agents is provided with an agent with a wider perspective of the system. Hierarchical structuring yields the classic master-slave or client-server coordination technique. Many researchers adopted the blackboard strategy to implement this technique: scheduled by a master agent, agents can read/write to/from the blackboard. Systems that exploit this architecture include the Designer Fabricator Interpreter (DFI) system proposed by Werkman [Werkman 1990], the Sharp Multi-Agent Kernel (SMAK), the Distributed Vehicle Monitoring Testbed (DVMT) system and the DRESUN testbed for research on distributed situation assessment (DSA) which explores the implications of agents with more sophisticated representations and control capabilities than those in DVMT [Carver, Lesser et al. 1993] and the free-market agent architecture MAGMA [Tsvetovatyy, Gini et al. 1997]. Although useful where the master-slave relationships are inherent to the modelled MAS, this technique is impractical in many realistic applications (because it presumes that at least one agent has a global view over the entire environment).

The *Contract Net Protocol* is a high-level coordination strategy proposed by Smith and Davis (as cited in [Nwana, Lee et al. 1996]) and used in many applications. This approach assumes a decentralised market structure in which agents can have two roles: a manager or a contractor. While monitoring the problem's overall solution, the manager breaks the

problem in sub-problems and assigns them to contractors which in turn solve them or may recursively become managers and further decompose the sub-problem. Best applications of this technique include well-defined hierarchical tasks, problems with a coarse-grained decomposition and applications characterised by minimal coupling among subtasks. The contract net strategy presents various advantages (e.g. better agreements due to dynamic task allocation, dynamic introduction/removal of agents, natural load-balancing, reliable mechanism for distributed control and failure recovery) as well as some limitations such as communication-intensity and, more important, the fact that it does not detect or resolve conflicts presuming only passive, benevolent and non-antagonistic agents (which is unrealistic for many real-world problems) [Nwana, Lee et al. 1996; Green, Hurst et al. 1997].

Multi-agent planning employs a detailed plan of agents' future actions and interactions (needed to achieve their goals) to avoid inconsistence and conflicts [Nwana, Lee et al. 1996]. There are two types of multi-agent planning i.e. centralised and distributed. The centralised multi-agent planning uses a coordinating agent to identify potential inconsistencies and conflicting interactions from the local plans sent by individual agents. Applications of this approach include the air-traffic control domain (implemented by Cammarata, McArthur and Steeb) and the MATPEN model proposed by Jin and Koyama (as cited in [Nwana, Lee et al. 1996]). The distributed multi-agent planning allows agents to build and update their individual plans as well as to model other agents' plans until all plan conflicts are resolved. This technique was used by Lesser and Corkill in their functionally accurate, cooperative (FA/C) approach for structuring distributed processing systems [Lesser and Corkill 1981]. Durfee implemented a framework for coordinating multiple AI systems cooperating in a distributed environment called partial global planning [Durfee and Lesser 1991]. The main disadvantage of the multi-agent planning technique for coordination is that it requires more computation and communication than other approaches because agents have to share and process substantial amounts of information [Green, Hurst et al. 1997].

Social laws is another technique that can be applied for coordination among intelligent agents. Conflicts among agents' actions can be avoided if any agent would have complete knowledge of the goals and intentions of all agents [Green, Hurst et al. 1997]. Chaib-draa proposes a framework for designing MAS systems in which agents *"are capable of coordinating their activities in routine, familiar, and unfamiliar situations"* [Chaib-draa 1996]. The guiding principles of this strategy are that coordination is easier in routine than

in unfamiliar situations and that all agents adopt and obey social laws such as social regularities and social collectivities.

Computational market-based mechanisms facilitate distribution of tasks and resource allocation through the use of auction-inspired protocols. This strategy can enhance the adaptivity, robustness and flexibility of MAS systems. However, Oliveira et al note that this technique "should include some risk assessment and risk management features" and the overall system performance needs to be further studied [Oliveira, Fischer et al. 1999].

Negotiation

Many researchers have studied the subject of negotiation providing many and diverse techniques and definitions for this term through a vast literature [Nwana, Lee et al. 1996; Green, Hurst et al. 1997; Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999]. Used for conflict resolution, negotiation is a significant aspect of the coordination process among autonomous agents in a system. Furthermore, negotiation is seen by many agent researchers as a key coordination technique also used to address several DAI issues [Nwana, Lee et al. 1996; Jennings, Sycara et al. 1998]. Bussman and Muller define negotiation as *"the communication process of a group of agents in order to reach a mutually accepted agreement on some matter"* (as cited in [Green, Hurst et al. 1997]). Jennings et al consider the following to be the main characteristics of negotiation [Jennings, Sycara et al. 1998]:

- The existence of a conflict.
- Self-interested agents have to resolve the conflict in a decentralised manner.
- Bounded rationality.
- Incomplete information.

Many researchers argue that agents must reason about beliefs, desires and intentions of other agents for an effective negotiation process [Rao and Georgeff 1995; Nwana, Lee et al. 1996]. This approach resulted in the attention to other research areas such as logic, case-based reasoning, belief revisions, distributed truth maintenance, model-based reasoning, optimisation and game theory [Nwana, Lee et al. 1996; Green, Hurst et al. 1997].

Game theory-based negotiation involves the application of concepts such as utility functions, space of deals and strategies and negotiation protocols. Agents use payoff matrices to represent common knowledge (each agent knows the *utility value* of the outcome of some interaction). Following a set of rules that govern the negotiation (the negotiation protocol), agents exchange their offers during an interactive process until an acceptable deal is reached [Nwana, Lee et al. 1996; Oliveira, Fischer et al. 1999]. The key

researchers of this area are Zlotkin and Rosenschein who use game theory to achieve coordination among autonomous agents in cooperative domains [Zlotkin and Rosenschein 1989; Zlotkin and Rosenschein 1996]. The main critique to this technique stresses the lack of realism due to the fact that agents are presumed to be fully rational and to have full knowledge of other agents' values [Nwana, Lee et al. 1996].

Many negotiation techniques are inspired from the human negotiation strategies [Nwana, Lee et al. 1996; Jennings, Sycara et al. 1998]. Motivated by theoretical analysis and observations of human interactions, Sycara and her research team adopt the logical model of the mental states of the agents (beliefs, desires, intentions and goals) to enable communication and negotiation among agents. Based on case-based reasoning and multi-attribute utility theory, Sycara proposed a system in which conflicts are resolved in labour relations with the aid of two practising negotiators. More recently, a new persuasive method for multiple-agent negotiation called *multiple negotiations* was proposed by Sycara and her team [Shintani, Ito et al. 2000].

Communication

To any MAS system, cooperation is essential in order to benefit from the added value provided by a collection of agents. Because agents generally have only a partial view over their environment, they will probably be required to communicate (to exchange information and knowledge in a distributed environment or to request the performance of a task) with each other in order to effectively cooperate. Nwana and Ndumu view communication *"at the heart of cooperation and competition germane to multi-agent systems"* [Nwana and Ndumu 1999]. An *agent communication language* (ACL) should facilitate the interactions between two or more agents (through communication by exchanging messages) [Genesereth and Ketchpel 1994; Nwana and Wooldridge 1996; Green, Hurst et al. 1997; Labrou, Finin et al. 1999; Chaib-draa and Dignum 2002]. Chaib-draa notes, *"the main objective of an ACL is to model a suitable framework that allows heterogeneous agents to interact, to communicate with meaningful statements that convey information about their environment or knowledge"* [Chaib-draa and Dignum 2002].

Nwana et al classify ACLs in two categories i.e. *ad hoc* and *standard* [Nwana and Wooldridge 1996]. Many agent-based applications contain collaborative agents that communicate using an ad hoc set of performatives within ad hoc ACLs or by depositing information in a shared database. However, this approach does not support interoperation between agent applications created by different developers.

Therefore, standard ACLs are essential to the cooperation process among various autonomous agents. Designed to support interactions among intelligent software agents, the Knowledge Query and Manipulation Language (KQML) is such a standard ACL proposed by the Knowledge Sharing Effort (KSE) consortium [Finin, Fritzson et al. 1994]. KQML is a high-level communication language and set of protocols for identifying, connecting with and exchanging information and knowledge among agents. Agents can specify the information requirements and capabilities using a set of KQML *performatives* that define the allowed "speech acts" (that agents may attempt in communicating with each other) and support the creation of more complex co-ordination and negotiation strategies. Run-time knowledge sharing is facilitated by a special type of agents called *communication facilitators* that coordinate the actions of other agents. KQML consists of three layers as follows [Finin, Labrou et al. 1997]:

1. The *content layer* specifies the content of the message.
2. The *message layer* encodes a message using the set of performatives provided by the language. It determines the kinds of KQML agent interactions and specifies the protocol for delivering the message.
3. The *communication layer* is used for encoding low level communication parameters such as the identity of the sender and recipient and a unique identifier for the communication.

Figure 16 shows this layered organization of a KQML message.

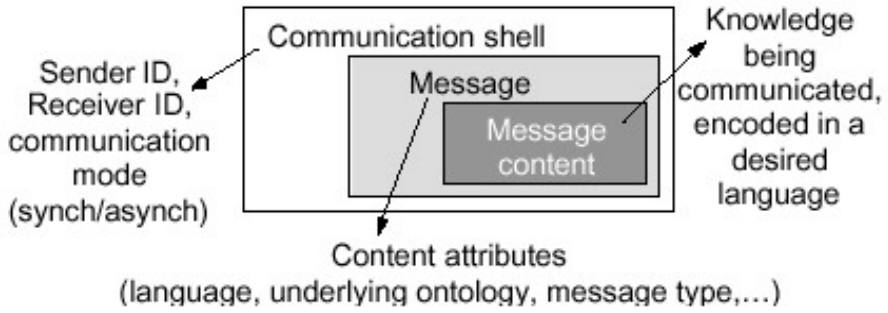

Figure 16. KQML layered organization (after [Devedzic 2001])

KSE researchers also designed a representation language for the contents of the messages called Knowledge Interchange Format (KIF) as an extension of first-order logic [Finin, Labrou et al. 1997]. However, KQML is independent of the content language (KIF, SQL, etc) and of the ontology assumed by the content [Labrou, Finin et al. 1999].

Although KQML is probably the most used agent communication language/protocol in the agent community, many researchers have identified various limitations of the language [Cohen and Levesque 1995; Nwana and Wooldridge 1996; Labrou, Finin et al. 1999]. Labrou et al indicate that *"different KQML implementations cannot interoperate"* and that *"there is no fixed specification sanctioned by a consensus-creating body"* and *"no agreed-upon semantics foundation"* [Labrou, Finin et al. 1999].

FIPA ACL is another standard ACL proposed by the Foundation for Intelligent Physical Agents (FIPA). FIPA is a standards organisation in the area of software agents whose goal is to develop specifications that maximize interoperability within and across agent-based systems [http://www.fipa.org; Labrou, Finin et al. 1999; Poslad, Buckle et al. 2000]. Similarly to KQML, communication between FIPA agents relies on the speech act theory. FIPA ACL is based on a set of communicative acts (also called performatives) such as request, inform and refuse that are specified by FIPA independently from the overall content of the message [Dale and Mamdani 2001]. The FIPA ACL also focuses on the effects on the mental attitudes (such as beliefs, desires, intentions) of the sender and receiver agents [Poslad, Buckle et al. 2000]. The FIPA ACL message structure includes the identity of sender and receiver as well as the ontology and interaction protocol of the message. The content of the message supplied with a communicative act is expressed in a content language such as the FIPA semantic language (FIPA SL). To achieve the desired agent interaction, a set of FIPA interaction protocols (including requesting an action, contract net and several kinds of auctions) was created to describe entire conversations between agents [http://www.fipa.org].

In a comparison of KQML and FIPA ACLs, Labrou et al find the two languages almost identical with the primary difference in the details of their semantic frameworks [Labrou, Finin et al. 1999]. However, Kumar et al indicate that *"most contemporary agent communication languages, notably FIPA and KQML, have either no provision or no well-defined semantics for group communication"* [Kumar, Huber et al. 2000]. More recently, Chaib-draa and Dignum identified a set of issues in the development of ACLs and agent communication theory using KQML and FIPA ACLs as examples. Potential problems of these ACLs include the following [Chaib-draa and Dignum 2002]:

- The linkage between the semantic theory and the theory of agency (which have to be aligned so that the ACL messages to be formally coherent)
- The semantics of KQML and FIPA-ACL (which are based on the mental agency while agents are almost never developed to use mental states)

- The verification of the semantics of an ACL and of an instantiation of a protocol to a protocol specification.
- The use of ontologies to interpret components of an ACL message.
- Limited coverage of communicative acts which are either assertives or directives (both KQML and FIPA ACL are extensible ACLs but the addition of performatives by different developers would lead to different incompatible dialects of these ACLs).
- The gap between individual messages and the extended message sequences (or conversations) that arise between agents.

Besides an ACL, a common understanding of the concepts used among agents is necessary for a meaningful agent communication. This is because agents may have different terms for the same concept or identical terms for different concepts [Odell 2000]. Therefore, ontologies are used for representing the knowledge from various application domains. The ACL remains just syntax without a shared common ontology containing the terms used in agent communication and the knowledge (e.g. definitions, attributes, relationships between terms and constraints) associated with them [Nwana and Wooldridge 1996]. Chaib-draa indicates that many ACLs need an ontology that should be characterised by the following [Chaib-draa and Dignum 2002]:

- Broad coverage (for allowing multiple agents to share knowledge in several contexts)
- Extensibility (for allowing designers to add new elements)
- Relevance to the domain

Both KQML and ACLs are designed to be independent of particular application vocabularies (by identifying the source of the vocabulary used in the message content). However, *"the way that an agent would make use of the KQML or FIPA-ACL ontology specification to interpret unfamiliar parts of an ACL message has never been precisely defined"* [Chaib-draa and Dignum 2002]. Active ongoing research is still focused on the general ontological problem.

Although a mature field, software agent research still lacks in universally accepted concepts from definitions to languages/protocols for coordination, negotiation and communication.

NOTE: An extensive literature review on coordination, negotiation and communication among agents can be performed. I think we should do this if we decide to concentrate on this aspect of MAS since coordination/negotiation/communication form a vast research area on its own.

6. Agent-Oriented Methodologies

Many researchers in the area of agents and MAS systems believe that agent-based computing has the potential to improve the conceptualisation, design and implementation of complex distributed software systems. A systematic methodology for the analysis and design of agent-based applications is a crucial requirement for the success of agent-oriented software engineering (AOSE) [Jennings 2000; Wooldridge and Ciancarini 2001]. Also, Ndumu and Nwana indicate that *"appropriate design methodologies for constructing the different types of agent systems for different application domains"* are needed before *"generic platforms for engineering agent-based applications"* [Ndumu and Nwana 1996].

Although there are many agent theories, languages and architectures available, very little work has been done in the area of agent-oriented methodologies to assist the developer in all the phases of the life cycle of an agent-based application. The available methodologies for the analysis and design of agent-based systems can be classified in two groups as follows [Iglesias, Garijo et al. 1999] [Wooldridge and Ciancarini 2001]:

- Methodologies that extend or adapt object-oriented methodologies e.g. AAII [Kinny, Georgeff et al. 1996], Gaia [Wooldridge, Jennings et al. 2000], MaSE [DeLoach 1999], AUML [Odell, Parunak et al. 2000].
- Methodologies that adapt knowledge engineering models or other techniques e.g. DESIRE [Brazier, Dunin-Keplicz et al. 1997].

The *Agent Modelling Technique for Systems of BDI agents* (also called the *AAII* – Australian Artificial Intelligence Institute – methodology) was developed by Kinny et al [Kinny, Georgeff et al. 1996] by building upon and adapting existing object-oriented models. Based on the BDI paradigm, this agent-oriented methodology and modelling technique provides both internal and external perspectives of MAS systems. The internal model deals with the agent's beliefs, desires and intentions. Presenting a system level view, the external model decomposes the system into agents and deals with the relationships between them.

Wooldridge et al [Wooldridge, Jennings et al. 2000] proposed a methodology for agent-oriented analysis and design called *Gaia*. Using the Gaia methodology, the designer of an agent-based application can systematically progress from a set of requirements to a detailed design ready to be implemented. Extending object-oriented analysis and design models, Gaia supports the developer to model complex systems through a process of organisational design. It provides a set of agent-specific concepts, which are of two types: abstract (i.e. roles, permissions, responsibilities, protocols, activities, liveness properties and safety properties) and concrete (i.e. agent types, services, acquaintances). Abstract

concepts are used to conceptualise the system during analysis while concrete concepts are used within the design process.

Proposed by DeLoach [DeLoach 1999], the *Multiagent Systems Engineering (MaSE)* for formal agent system synthesis is a further abstraction of the object-oriented paradigm. Two languages i.e. Agent Modeling Language (AgML) and Agent Definition Language (AgDL) are used to describe agents and MAS systems.

Odell et al [Odell, Parunak et al. 2000] explored UML (Unified Modeling Language) idioms and extensions that can be used to model agents and agent-based systems. The result is an UML-based approach to building agent applications called *Agent UML (AUML)*. It should be noted that UML is a widely accepted standard for object-oriented modelling but it is not a methodology (it is rather a language). AUML extends the UML notation by supporting concurrent threads of interaction and allowing an agent to play many roles. Both FIPA [http://www.fipa.org] and OMG [http://www.omg.org] groups are recommending the use of UML extensions for the specification of agent-based systems.

All these extensions of object-oriented methodologies share a number of advantages as follows [Iglesias, Garijo et al. 1999]:

- The benefit of the similarities between the object-oriented paradigm and the agent-oriented paradigm.
- The commonly usage of object-oriented languages to implement agent-based applications (as shown in the next section).
- The popularity of object-oriented methodologies.

However, *"object-oriented methodologies simply do not allow us to capture many aspects of agent systems"* [Wooldridge and Ciancarini 2001]. Most disadvantages of these methodologies fall out from the differences between objects and agents, as follows [Iglesias, Garijo et al. 1999]:

- Message passing used to communicate (for objects, is just method invocation while agents analyse and model these messages, can decide whether or not to execute the requested action and use complex protocols to negotiate).
- Lack of techniques for modelling the agent's mental state.
- Lack of procedures for modelling the social relationships between agents.

Because the predominant approach to implementing methodologies for agents and MAS systems is to extend object-oriented paradigms, little work has been done on agent-oriented methodologies that adapt knowledge engineering models. The *DESIRE* (DEsign and Specification of Interacting REasoning components) framework supports the specification and implementation of component-based autonomous interactive agents [Brazier, Dunin-

Keplicz et al. 1997]. Using DESIRE, the analyst can explicitly model complex reasoning within agents, communication patterns between agents as well as interactions with the external world. This high-level modelling framework supports conceptual design and specification of both dynamic and static aspects of agent behaviour. Brazier et al report the successful application of the compositional multiagent framework DESIRE to develop a conceptual specification of simple agents and to simulate the behaviour in a dynamic environment [Brazier, Eck et al. 2001].

Other approaches to modelling MAS systems (e.g. CoMoMAS and MAS-CommonKADS) extend the existing CommonKADS methodology for knowledge engineering [Iglesias, Garijo et al. 1999].

The advantages of adapting knowledge engineering methodologies for the design of agent-based systems can be summarised as follows [Iglesias, Garijo et al. 1999]:

- They provide techniques for modelling the agent's knowledge (knowledge acquisition process).
- The existing tools, ontology libraries and problem solving method libraries can be reused.

However, these methodologies *"do not address the distributed or social aspects of the agents, or their reflective and goal-oriented attitudes"* [Iglesias, Garijo et al. 1999] because a knowledge based system is conceived as a centralised one.

7. Agent Languages and Environments

The growing interest in the area of software agents and multi-agent systems motivated the development of languages that facilitate the design and construction of agent-based applications. Wooldridge and Jennings define an agent language as *"a system that allows one to program hardware or software computer systems in terms of some of the concepts developed by agent theorists"* [Wooldridge and Jennings 1995].

Although numerous languages and platforms have been created by different research groups and companies to support the development of agent-based applications, traditional languages are still used to construct agent applications. Nwana and Wooldridge indicate, *"typically, object-oriented languages such as Smalltalk, Java or C++ lend themselves more easily for the construction of agent systems"* [Nwana and Wooldridge 1996]. The reason for this is that agents and objects share some properties such as encapsulation, inheritance and message passing. However, objects may respond to the same message in different ways (polymorphism) while agents must have a common ACL. In object-oriented programming, an object can decide to invoke a method of another object but when an agent

wants to do the same thing (to request an action from another agent) the decision lies with the agent that receives the request. Jennings et al summarise this distinction by observing *"objects do it for free; agents do it for money"* [Jennings, Sycara et al. 1998]. The most used programming language for developing agent applications is Java due to its rich library of functions tackling concurrency as well as security [Huget 2002], support for object-oriented programming techniques, code portability, native support for multithreading and introspection of object properties and methods [Bigus, Schlosnagle et al. 2002].

It should be noted that there is a very large number of agent languages, environments, frameworks and toolkits already available and only some of them will be reviewed in the following, as it is impossible to be exhaustive.

In the early 90s, Shoham proposed a new programming paradigm called agent-oriented programming (AOP) that *"promotes a societal view of computation"* [Shoham 1998]. A specialization of object-oriented programming(OOP), AOP allows the direct programming of agents in terms of their mental state (consisting of components such as beliefs, decisions, capabilities and obligations). Agent programs control agents and include communication primitives such as informing, requesting and offering (based on the speech act theory). Table 5 presents the relation between OOP and AOP [Shoham 1998].

	OOP	AOP
Basic unit	object	agent
Parameters defining state of basic unit	unconstrained	beliefs, commitments, capabilities, choices,…
Process of computation	message passing and response methods	message passing and response methods
Types of messages	unconstrained	inform, request, offer, promise, decline,…
Constraints on methods	none	honesty, consistency

Table 5. The relation between OOP and AOP [Shoham 1998]

Shoham indicates that a complete AOP system contains a restricted formal language for describing the mental state, an interpreted programming language for defining and programming agents and an 'agentifier' for converting neutral devices into programmable agents. Furthermore, Shoham developed the *AGENT0* programming language as an implementation of the AOP paradigm. AGENT0 allows the specification of an agent using a set of capabilities, a set of initial beliefs, a set of initial commitments and a set of

commitment rules. Each commitment rule consists of a message condition, a mental condition and an action. The agent becomes committed to the action if the message condition matches the messages received by the agent and the mental condition matches the beliefs of the agent [Shoham 1998]. Figure 17 presents the control flow in AGENT0 [Wooldridge 1999]. An operation loop of the agent consists of reading all current messages, updating beliefs and executing all necessary commitments.

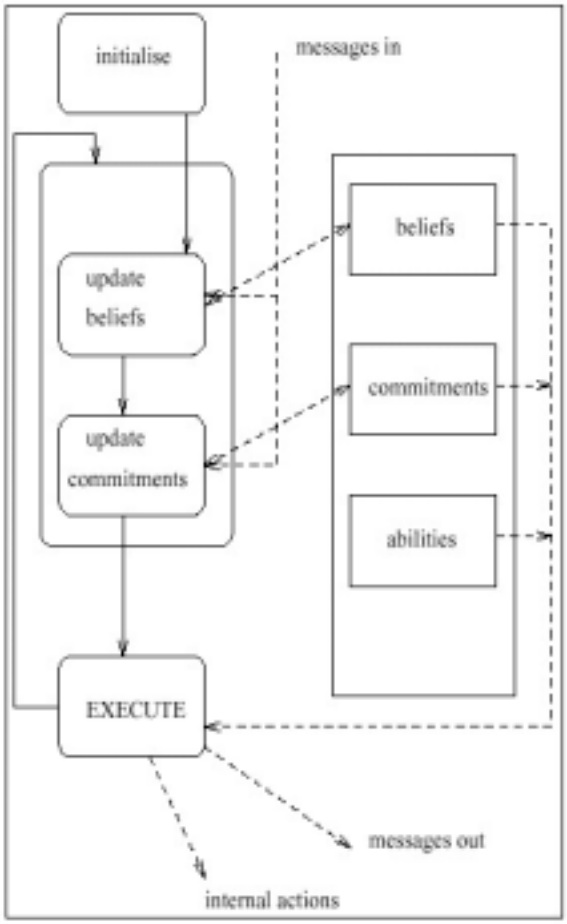

Figure 17. The flow of control in the AGENT0 language (after [Wooldridge 1999])

Although Shoham's work is of significant importance in the research area of agent languages, Wooldridge notes that the AGENT0 AOP language is only a prototype *"not intended for building anything like large-scale production systems"* and is limited because *"the relationship between the logic and interpreted programming language is only loosely defined"* [Wooldridge 1999].

Another logical approach to agent programming is *Concurrent METATEM* [Fisher 1994], a multi-agent programming language based upon the direct execution of linear time temporal logic agent specifications. A Concurrent METATEM system consists of concurrently executing agents whose behavior is implemented using executable temporal logic and which can communicate via asynchronous broadcast message passing.

Created by Rao, *AgentSpeak(L)* [Rao 1996] is a programming language that allows the formalization of BDI agents. It is based on restricted first-order language with events and actions and consists of a set of base beliefs and a set of context-sensitive plans allowing hierarchical decomposition of goals.

An extended version of Golog, *ConGolog (Concurrent Golog)* is a concurrent programming language for process specification and agent programming [Giacomo, Lespérance et al. 2000]. It handles concurrent processes with possibly different priorities, high-level interrupts and arbitrary exogenous actions. ConGolog supports the formal specification of complex MAS systems but "*lacks features for modelling the rationale behind design choices"* [Wang and Lespérance 2001].

Over the last years, a large number of toolkits and developing environments have been created to support the agent developer in the task of implementation of agent-based systems. Some of these agent technologies are reviewed in the followings.

ZEUS [Nwana, Ndumu et al. 1999] is a toolkit for constructing collaborative multi-agent applications developed by BT Laboratories. Implemented in the Java programming language, ZEUS facilitates the creation of agents by specifying the attributes and the tasks of individual agents. A generator tool can then be invoked to create the source code implementation for each agent. The developer can make use of an agent component library, visualisation tools and agent building software components in order to access the application-independent agent-level functionality required of collaborative agents, to observe the agents' behaviour and to interactively create agents by specifying their attributes and strategies. Initially, ZEUS supported only KQML as the agent communication language but it was further developed to support FIPA ACL as well. ZEUS is an open-source software freely available under an general public license.

Fully implemented in Java, *JADE - Java Agent DEvelopment Framework* - [Bellifemine, Poggi et al. 1999] is a software framework that facilitates the development of MAS systems. The JADE agent platform is compliant with the FIPA specifications and performs all agent communication through message passing (using FIPA ACL to represent messages). JADE adopts the multi-thread solution (offered by Java) and supports scheduling of cooperative behaviours. The graphical user interface facilitates the remote management, monitoring and controlling of the status of agents, the creation and execution of an agent on a remote host as well as control of other FIPA compliant agent platforms. JADE successfully participated in the FIPA interoperability tests and is currently under further development.

Developed by Nortel Networks, *FIPA-OS* [Poslad, Buckle et al. 2000] is an open agent platform designed to comply with the FIPA agent standards [http://www.fipa.org]. It supports communication between multiple agents and operates in a heterogeneous open service environment. The key aspect of FIPA-OS is openness, which is accentuated by the fact that the platform software is distributed and managed under an open-source licensing scheme.

Developed by the Agent Oriented Software group [http://www.agent-software.com], *JACK Intelligent Agents* [Howden, Ronnquist et al. 2001] is a Java framework for multi-agent system development. It supports the BDI architecture model but it can be extended to support different agent models and specific application requirements. The JACK agent language extends the Java programming language with agent-oriented concepts such as agents, capabilities, events, plans, agent knowledge bases, and resource and concurrency management. Using a component-based approach, JACK provides the core architecture and infrastructure for building, running and integrating software agents in distributed applications.

The *Open Agent Architecture (OAA)* [Cheyer and Martin 2001] is a domain-independent framework for constructing agent-based systems. The facilitator agent and libraries (in several languages) can be used for creating application agents. Coordination and communication among agents is addressed by one ore more facilitators (specialized server agents) included in each OAA-based system. Supporting flexible, dynamic configurations of autonomous agents, OAA facilitates the declaration of capabilities by service-providing agents, the construction of goals by users and service-requesting agents, the creation and maintenance of shared repositories of data and the use of triggers to instantiate commitments within and between agents.

Developed at Stanford University, *JATLite (Java Agent Template, Lite)* [Jeon, Petrie et al. 2000] is a collection of Java objects and class libraries that facilitates the creation of software agents communicating robustly over the Internet in order to perform a distributed computation. Agent messages used in the communication process are primarily based on the KQML language and protocol. JATLite features include multi-threaded operation, a message router for agent registration, connection, name and password services, storage and queuing of messages for mobile and sporadic agents, and support for both stand-alone agents in Java and C++ and applet agents.

Other available agent toolkits and multiagent platforms include the Java-based system AgentBuilder [http://www.agentbuilder.com/], the IBM Agent Building and Learning Environment (ABLE) [Bigus, Schlosnagle et al. 2002], the Java Intelligent Agent

Component Ware (JIAC) [Ballmann and Wieczorek 1998], the Java-based Grasshopper [IKV++GmbH 2001], the International Knowledge System's AgentX [http://www.iks.com/agentx.htm], Mitsubishi's Java-based mobile agent system Concordia [Kiniry and Zimmerman 1997], the Direct Intelligent Adaptation (DirectIA) system for creating adaptive agents [http://www.directia.com/], the Agent Development Toolkit (ADK) for mobile agents [http://www.tryllian.com], the iGEN toolkit for building cognitive agents [http://www.cognitiveagent.com], IBM Japan's Java-based autonomous software agent technology Aglets Software Development Kit [http://www.trl.ibm.com/aglets], the SodaBot system developed at MIT Artificial Intelligence Lab [http://www.ai.mit.edu/people/sodabot/sodabot.html]. Offering varied functionality, these systems do not necessarily adhere to any standards: an agent created using one system will not work in another. Also, "*there is no uniform support for communication protocols across these tools either*" [Odell 2000]. However, the majority of these agent systems adopted the following strategies [Odell 2000]:

- The programming language used for implementation is Java or C++.
- The communication language for agent interoperability is KQML or FIPA ACL.
- The content language used for representing knowledge is KIF or FIPA SL.

Some of the commercial agent-building systems are purchasable and some are freely available under a general public license.

8. Applications of agents

Several application areas are currently focused on the employment of agents and multi-agent-systems in complex problem solving processes. Jennings and Wooldridge indicate that the agent-based solution is appropriate for open (or at least, highly dynamic, uncertain or complex) environments in which flexible and autonomous agents may be the only solution [Jennings and Wooldridge 1998; Wooldridge 1998]. Domains in which data, control, expertise or resources are inherently distributed can be addressed using agent technology. Finally, the agent-based approach is suitable for environments that are naturally modelled as societies of autonomous cooperating components (the agent is a natural metaphor) as well as for systems that contain legacy components (see section 2) [Jennings and Wooldridge 1998]. Also, Oliveira et al identify elements such as distribution, complexity, flexible interaction, highly dynamic environments and openness as the typical properties of the application domains in which multi-agent technologies are most appropriate [Oliveira, Fischer et al. 1999].

Jennings and Wooldridge present a classification of agent applications by the domain to which they are applied [Jennings and Wooldridge 1998]:

- *Industrial applications*: process control, manufacturing, and air traffic control.
- *Commercial applications*: information management, electronic commerce, and business process management.
- *Medical applications*: patient monitoring, and health care.
- *Entertainment*: games, and interactive theatre and cinema.

To these, Wooldridge adds the following domains [Wooldridge 1998]: industrial systems management, distributed sensing, space shuttle fault diagnosis, factory process control (distributed applications of agent technology). Also, in the area of agent applications for the Internet, there is special attention accorded to mobile agents that can migrate around the Internet working on the user's behalf. Other than the already mentioned electronic commerce domain, work has been done in the information gathering area and Personal Digital Assistant (PDA) systems. Another application area for agents is represented by interfaces: the agent assists the user during tasks by anticipating requirements [Wooldridge 1998].

Jennings and Wooldridge [Jennings and Wooldridge 1998] identify three dimensions along which these agent applications can be analysed: sophistication of the agents, role of the agents and granularity of view. There are three levels of sophistication based on the type of the agent's behavior as follows [Jennings and Wooldridge 1998]:

1. The *gopher* agent – executes simple tasks based on well-defined and pre-specified information
2. The *service performing* agent – executes high-level tasks based on well-defined information
3. The *predictive/proactive* agent – is capable of executing tasks in a flexible and autonomous manner on behalf of its user

Secondly, agent based systems can be analysed by considering the role of the agents e.g. in industrial and commercial applications, the role is to provide decision support functionality (the final decision belongs to the user) while in the entertainment domain, the role is to completely solve the problem. Finally, granularity of view refers to the use of the individual agent as opposed to a society of agents for specific domains. Some applications adopt the single-agent approach, others use MAS systems (these are probably more complicated since issues such as communication, coordination and negotiation have to be considered by the developer) [Jennings and Wooldridge 1998].

Other classification schemes for agent applications are available from different authors or groups of researchers. For example, the OMG group [http://www.omg.org] classify the applications that use agents as follows [Odell 2000]:

- Enterprise applications e.g. smart documents, goal-oriented enterprise, role and personnel management.
- Business-to-business applications e.g. market making for goods and services, team management.
- Process control e.g. intelligent buildings, plant management, robots.
- Personal agents e.g. email and news filters, personal schedule management, personal automatic secretary.
- Information management tasks e.g. searching for information, information filtering, information monitoring, data source mediation, interface agents/personal assistants.
- Nomadic computing applications (agents for mobile computing)

Although agents and MAS systems are increasingly employed in various application domains, there are several problems associated with the agent-based approach to building computer-based systems as follows [Jennings and Wooldridge 1998]:

- *No overall system controller*

  Agents may not be the answer for domains with global constraints or that require a guaranteed real-time response.
- *No global perspective*

  A basic issue in MAS research is that of integrating the agent's decisions based on local knowledge with the desire to achieve globally optimal performance.
- *Trust and delegation*

  In order to delegate tasks to agents, users must trust agents that they work indeed in their behalf. Furthermore, *"the agent must strike a balance between continually seeking guidance (and needlessly distracting the user) and never seeking guidance (and exceeding its authority). Put crudely, an agent must know its limitations."* [Jennings and Wooldridge 1998].

While there are still many problems associated with the design and implementation of agent-based applications, MAS systems *"provide a powerful model for computing in the 21st century, in which networks of interacting, real-time, intelligent agents seamlessly integrate the work of people and machines, and dynamically adapt their problem solving to effectively deal with changing usage patterns, resource configurations and available sources of expertise and information"* [Lesser 1999].

References

Anumba, C. J., O. O. Ugwu, L.Newnham and A.Thorpe (2002). "Collaborative design of structures using intelligent agents." Automation in Construction 11: 89-103.

Ballmann, S. and D. Wieczorek (1998). Java Intelligent Agent Component Ware (JIAC) - technical documentation. Berlin, DAI Laboratory
Technical University of Berlin.

Bellifemine, F., A. Poggi and G. Rimassa (1999). JADE - A FIPA-compliant agent framework. Proceedings of PAAM'99, London.

Bigus, J. P., D. A. Schlosnagle, J. R. Pilgrim, W. N. M. III and Y. Diao (2002). "ABLE: A toolkit for building multiagent autonomic systems." IBM Systems Journal 41(3).

Bradshow, J. M. (1997). An Introduction to Software Agents. Software Agents. J. M. Bradshow. Cambridge, MIT Press.

Brazier, F. M. T., B. M. Dunin-Keplicz, N. R. Jennings and J. Treur (1997). "DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework." International Journal of Cooperative Information Systems 6(Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems): 67-94.

Brazier, F. M. T., P. A. T. v. Eck and J. Treur (2001). "Modelling a Society of Simple Agents: From Conceptual Specification to Experimentation." Journal of Applied Intelligence 14: 161-178.

Brooks, R. A. (1986). "A robust layered control system for a mobile robot." IEEE Journal of Robotics and Automation 2: 14-23.

Carver, N., V. Lesser and Q. Long (1993). Distributed sensor Interpretation: Modeling Agent Interpretations in DRESUN, UMass Technical Report, UMCS 93-75.

Chaib-draa, B. (1996). "Interaction Between Agents in Routine, Familiar and Unfamiliar Situations." International Journal of Intelligent & Cooperative Information Systems 5(1): 1-25.

Chaib-draa, B. and F. Dignum (2002). "Trends in Agent Communication Language." Computational Intelligence 18(2).

Cheyer, A. and D. Martin (2001). "The Open Agent Architecture." Journal of Autonomous Agents and Multi-Agent Systems 4(1): 143-148.

Chu, E., K. Srihari and C. R. Emerson (1996). "Distributed Artificial Intelligence in Process Control." 19th International Conference on Computers and Industrial Engineering.

Cohen, P. R. and H. J. Levesque (1995). Communicative actions for artificial agents. Proceedings of the International Conference on Multi-Agent Systems, San Francisco, AAAI Press.

Dale, J. and E. Mamdani (2001). "Open Standards for Interoperating Agent-Based Systems." Software Focus, Wiley.

DeLoach, S. A. (1999). Multiagent Systems Engineering: A Methodology And Language for Designing Agent Systems. Agent-Oriented Information Systems (AOIS) '99.

Devedzic, V. (2001). "Knowledge Modeling - State of the Art." Integrated Computer-Aided Engineering 8(3): 257-281.

Durfee, E. H. (2001). "Scaling Up Agent Coordination Strategies." IEEE Computer 34(7): 39-46.

Durfee, E. H. and V. R. Lesser (1991). "Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation." IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks SMC-21(5): 1167-1183.

Ferguson, I. A. (1992). "TouringMachines: Autonomous Agents with Attitudes." IEEE Computer 25(5).

Finin, T., R. Fritzson, D. McKay and R. McEntire (1994). KQML as an Agent Communication Language. Proceedings of the Third International Conference on Information and Kknowledge Management.

Finin, T., Y. Labrou and J. Mayfield (1997). KQML as an agent communication language. Software Agents. B. M. Jeffrey, MIT Press.

Fisher, M. (1994). A Survey of Concurrent METATEM - The Language and its Applications. Proceedings of First International Conference on Temporal Logic (ICTL), Bonn, Germany, Springer-Verlag.

Foner, L. N. (1993). What's An Agent, Anyway? A Sociological Case Study, Media Laboratory, Massachusetts Institute of Technology.

Franklin, S. and A. Graesser (1996). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996, Berlin, Germany.

Gasser, L. (1998). Social conceptions of knowledge and action: DAI foundations and open systems dynamics. Readings in Agents. M. N. Huhns and M. P. Singh, Morgan Kaufmann Publishers.

Genesereth, M. R. and S. P. Ketchpel (1994). "Software Agents." Communications of the ACM, ACM Press.

Georgeff, M., B. Pell, M. Pollack, M. Tambe and M. Wooldridge (1999). The Belief-Desire-Intention Model of Agency. Intelligent Agents. J. P. Muller, M. Singh and A. Rao, Springer-Verlag. 1365.

Giacomo, G. D., Y. Lespérance and H. J. Levesque (2000). "ConGolog, a concurrent programming language based on the situation calculus." Artficial Intelligence 121: 109-169.

Green, S., L. Hurst, B. Nangle, P. Cunningham, F. Somers and R. Evans (1997). Software Agents: A review. Dublin, Intelligent Agents Group
Trinity College Dublin

Broadcom Eireann Research Ltd.

Helin, H. (2003). Agent Architectures & Languages, http://www.cs.helsinki.fi/u/hhelin/opetus/oat/. 2003.

Horn, E., M. Kupries and T. Reinke (1999). Properties and Models of Software Agents and Prefabrication for Agent Application Systems. Proceedings of the 32nd Hawaii International Conference on System Sciences.

Howden, N., R. Ronnquist, A. Hodgson and A. Lucas (2001). JACK Intelligent Agents - Summary of an Agent Infrastructure. 5th International Conference on Autonomous Agents.

http://www.agentbuilder.com/.

http://www.agent-software.com.

http://www.ai.mit.edu/people/sodabot/sodabot.html The SodaBot System.

http://www.cognitiveagent.com iGEN Overview.

http://www.directia.com/.

http://www.fipa.org Foundation for Intelligent Physical Agents.

http://www.iks.com/agentx.htm.

http://www.omg.org Object Management Group.

http://www.trl.ibm.com/aglets Aglets.

http://www.tryllian.com The Agent Development Kit (ADK).

Huget, M.-P. (2002). Desiderata for Agent Oriented Programming Languages, University of Liverpool.

Iglesias, C. A., M. Garijo and J. C. Gonzalez (1999). A Survey of Agent-Oriented Methodologies. Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages.

IKV++GmbH (2001). Grasshopper Basics And Concepts. http://www.grasshopper.de/.

Ingrand, F. F., M. P. Georgeff and A. S. Rao (1992). "An Architecture for Real-Time Reasoning and System Control." IEEE Expert 7(6): 33-44.

Jennings, N. R. (2000). "On agend-based software engineering." Artficial Intelligence.

Jennings, N. R., K. P. Sycara and M. Wooldridge (1998). "A Roadmap of Agent Reasearch and Development." Journal of Autonomous Agents and Multi-Agent Systems 1(1): 7-36.

Jennings, N. R. and M. Wooldridge (1998). Applications of Agent Technology. Agent Technology: Foundations, Applications, and Markets. N. R. Jennings and M. Wooldridge, Springer-Verlag.

Jeon, H., C. Petrie and M. R. Cutkosky (2000). "JATLite: A Java Agent Infrastructure with Message Routing." IEEE Internet Computing.

Kiniry, J. and D. Zimmerman (1997). "A Look at Mitsubishi's Concordia." IEEE Internet Computing online.

Kinny, D., M. Georgeff and A. Rao (1996). A Methodology and Modelling Technique for Systems of BDI Agents. Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Springer.

Kumar, S., M. J. Huber, D. R. McGee, P. R. Cohen and H. J. Levesque (2000). Semantics of Agent Communication Languages for Group Interaction. The Seventeenth National Conference on Artificial Intelligence (AAAI 2000), Austin, Texas, AAIT Press/The MIT Press.

Labrou, Y., T. Finin and Y. Peng (1999). "Agent Communication Languages: The Current Landscape." IEEE Intelligent Systems.

Lazansky, J., O. Stepankova, V. Marik and M. Pechoucek (2001). "Application of the multi-agent approach in production planning and modelling." Engineering Applications of Artificial Intelligence 14(3): 369-376.

Lesser, V. and D. Corkill (1981). "Functionally Accurate, Cooperative Distributed Systems." IEEE Transactions on Systems, Man, and Cybernetics SMC-11(1): 81-96.

Lesser, V. R. (1999). "Cooperative Multiagent Systems: A Personal View of the State of the Art." IEEE Transactions on Knowledge and Data Engineering 11(1).

Maes, P. (1995). "Artificial Life meets Entertainment: Lifelike Autonomous Agents." Communications of the ACM, ACM Press 38(11): 108-114.

Martin, F. J., E. Plaza, J. A. Rodriguez-Aguilar and J. Sabater (1998). Java Interagents for Multi-Agent Systems. Software Tools for Developing Agents.

Muller, J. P. and M. Pischel (1993). The Agent Architecture InteRRaP: Concept and Application, DFKI Saarbrucken.

Ndumu, D. and H. Nwana (1996). "Research and Development Challenges for Agent-Based Systems." IEE/BCS Software Engineering Journal.

Nwana, H., L. Lee and N. Jennings (1996). "Coordination in Software Agent Systems." BT Technology Journal 14(4): 79-88.

Nwana, H. and M. Wooldridge (1996). "Software Agent Technologies." BT Technology Journal 14(4): 68-78.

Nwana, H. S. (1996). "Software Agents: An Overview." Knowledge Engineering Review 11(3): 1-40.

Nwana, H. S. and D. T. Ndumu (1999). A Perspective on Software Agents Research. Ipswich, British Telecommunications Laboratories.

Nwana, H. S., D. T. Ndumu, L. C. Lee and J. C. Collis (1999). "ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems." Applied Artifical Intelligence Journal 13(1): 129-186.

Odell, J. (2000). Agent Technology - Green Paper, OMG - Agent Platform Special Interest Group.

Odell, J., H. V. D. Parunak and B. Bauer (2000). Extending UML for Agents. Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence.

Oliveira, E., K. Fischer and O. Stepankova (1999). "Multi-agent systems: which research for which applications." Robotics and Autonomous Systems 27: 91-106.

Poslad, S., P. Buckle and R. Hadingham (2000). The FIPA-OS Agent Platform: Open Source for Open Standards. Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, UK.

Rao, A. S. (1996). AgentSpeak(L): BDI Agents speak out in a logical computable language. Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World.

Rao, A. S. and M. P. Georgeff (1995). BDI Agents: From Theory to Practice. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA.

Russell, S. and P. Norvig (2003). Artificial Intelligence: A Modern Approach, 2/E, Prentice Hall.

Sen, S. (1997). "Multiagent systems: milestones and new horizons." Trends in Cognitive Sciences 1(9).

Shintani, T., T. Ito and K. Sycara (2000). Multiple negotiations among agents for a distributed meeting scheduler. Proceedings of the Fourth International Conference on MultiAgent Systems.

Shoham, Y. (1998). Agent-oriented programming. Readings in Agents, Elsevier Science. Artificial Intelligence 60 (1993).

Tsvetovatyy, M., M. Gini, B. Mobasher and Z. Wieckowski (1997). "MAGMA: An agent-based virtual market for electronic commerce." Journal of Applied Artificial Intelligence.

Wang, X. and Y. Lespérance (2001). Agent-Oriented Requirements Engineering Using ConGolog and i*. Proceedings of the 3rd International Bi-Conference Workshop AOIS-2001, Berlin, iCue Publishing.

Werkman, K. J. (1990). Multiagent Cooperative Problem-Solving through Negotiation and Sharing of Perspectives. DAI-List, http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/pubs/lists/dai-list/dailist/006.10may90.

Wooldridge, M. (1998). "Agent-based computing." Interoperable Communication Networks 1(1): 71-97.

Wooldridge, M. (1999). Intelligent Agents, The MIT Press.

Wooldridge, M. and P. Ciancarini (2001). Agent-Oriented Software Engineering: The State of the Art. Agent-Oriented Software Engineering. P. Ciancarini and M. Wooldridge, Springer-Verlag. AI Volume 1957.

Wooldridge, M. and N. R. Jennings (1995). "Intelligent Agents: Theory and Practice." Knowledge Engineering Review 10(2).

Wooldridge, M., N. R. Jennings and D. Kinny (2000). "The gaia Methodology for Agent-Oriented Analysis and Design." Autonomous Agents and Multi-Agent Systems Kluwer Academic Publishers(3): 285-312.

Wooldridge, M. J. and N. R. Jennings (1995). "Agent Theories, Architectures, and Languages: A Survey." Lecture Notes in Artificial Intelligence, Springer-Verlag 890.

Zlotkin, G. and J. S. Rosenschein (1989). Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains. The Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan.

Zlotkin, G. and J. S. Rosenschein (1996). "Mechanism Design for Automated Negotiation, and its Application to Task Oriented Domains." Journal of Artificial Intelligence 86(2): 195-244.