# State-of-the-art in Agent-based Services

**Walter Binder (Ecole Polytechnique Fédérale de Lausanne)**
**Junichiro Mori (Ecole Polytechnique Fédérale de Lausanne)**
**David Portabella (Ecole Polytechnique Fédérale de Lausanne)**
**Valentina Tamma (University of Liverpool)**
**Michael Wooldridge (University of Liverpool)**

**Abstract.**
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D2.4.3
This deliverable covers the current state-of-the-art in Agent-based services. It consists of two main parts.
The first part provides an overview of autonomous agents and multi-agents systems, describing their desired properties and micro (agent-level) and macro (society-level) issues that need to be addressed to build such systems.
The second part of the document addresses technical issues of building Agent-based services, describing the FIPA standardization effort, and a list of major publicly available implementations of agent platforms. Finally this delivery presents the Agentcities testbed.
Keyword list: Agent, MAS, Services, Collaboration, ACL, FIPA, Agentcities

| | |
|---|---|
| Document Identifier | KWEB/2004/D2.4.3/v1.0 |
| Project | KWEB EU-IST-2004-507482 |
| Version | v1.0 |
| Date | October 31, 2004 |
| State | final |
| Distribution | public |

## Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

**University of Innsbruck (UIBK) - Coordinator**
Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

**École Polytechnique Fédérale de Lausanne (EPFL)**
Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

**France Telecom (FT)**
4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

**Freie Universität Berlin (FU Berlin)**
Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

**Free University of Bozen-Bolzano (FUB)**
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

**Institut National de Recherche en Informatique et en Automatique (INRIA)**
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

**Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)**
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

**Learning Lab Lower Saxony (L3S)**
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

**National University of Ireland Galway (NUIG)**
National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

**The Open University (OU)**
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

**Universidad Politécnica de Madrid (UPM)**
Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

**University of Karlsruhe (UKARL)**
Institut für Angewandte Informatik und Formale Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

**University of Liverpool (UniLiv)**
Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

**University of Manchester (UoM)**
Room 2.32. Kilburn Building, Department of Computer Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

**University of Sheffield (USFD)**
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

**University of Trento (UniTn)**
Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

**Vrije Universiteit Amsterdam (VUA)**
De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

**Vrije Universiteit Brussel (VUB)**
Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

# Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document:

École Polytechnique Fédérale de Lausanne
University of Liverpool

# Executive Summary

There are lots of interpretations about what Agents are. Rather than entering a debate about the issue, we give a motivation example and afterwards identify the key properties of such systems, split into micro and macro issues.

For Intelligent Autonomous Agents, two main approaches are identified. First, the traditional, symbolic approach as a type of knowledge-based system which deals with what knowledge needs to be represented, how this knowledge is to be represented, what reasoning mechanisms are to be used, etc. The second approach does no symbolic reasoning at all, and agents are designed as a set of task accomplishing behaviours, arranged into a subsumption hierarchy.

In Multi-Agent Systems, the agents are meant to be self interested: they are attempting to get the best deal for themselves, hence we enter into the formal setting of game theory. To obtain the most preferred outcome in such settings agents must engage in strategic reasoning.

In speech act theory, communication is modelled as actions that alter the mental state of the communication participants. This theory has inspired a number of languages that have been developed specifically for agent communication. For agents to communicate in open distributed environments, it is necessary for them to agree on the terminology that they use to describe their domain, i.e., to make an ontological commitment. Moreover, even if agents can successfully communicate, it remains the problem of why and how agents should cooperate, how agents can recognize and resolve conflicts, how agents can negotiate or compromise in situations where they are apparently at loggerheads, etc.

After agent theories have been studied and validated, standardization and implemented tools are needed for the practical commercial and industrial use of heterogeneous and interacting agents and agent-based systems across multiple vendors' platforms. Finally, the Agentcities testbed is analyzed.

Concerning the relationship between Agent-based Services and Semantic Web Services, while the former have been studied in the last 25 years, the latter are still in their infancy. Therefore, they should take profit of the state-of-the-art in the agents domain, specially in the areas of reaching agreements, communication, and collaboration. This topic will be addressed and further elaborated in the following deliverable D2.4.4, titled "Guidelines for the integration of agent-based services and web-based services".

# Contents

**1 Introduction** **1**

1.1 What are Agents? . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Micro and Macro Issues . . . . . . . . . . . . . . . . . . . . . . 4

**2 Intelligent Autonomous Agents** **6**

2.1 Logical/Reasoning Architectures . . . . . . . . . . . . . . . . . . 6

2.2 Reactive Architectures . . . . . . . . . . . . . . . . . . . . . . . 8

2.3 Hybrid Architectures . . . . . . . . . . . . . . . . . . . . . . . . 8

2.3.1 The Procedural Reasoning System (PRS) . . . . . . . . . . . . . 9

2.3.2 The 3T Architecture . . . . . . . . . . . . . . . . . . . . . . 11

**3 Negotiation and Agreement** **15**

3.1 Argumentation . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

3.2 Auctions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.3 Bargaining . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

**4 Communication and Cooperation** **30**

4.1 Communication . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

4.1.1 Speech Acts . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

4.1.2 Agent Communication Languages . . . . . . . . . . . . . . . . . 31

4.1.3 Ontologies . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

4.2 Cooperation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

4.2.1 Task Sharing and Result Sharing . . . . . . . . . . . . . . . . 35

4.2.2 Coordination . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

**5 Standardization** **40**

5.1 FIPA: Open Standards for Software Agents . . . . . . . . . . . . . 40

5.1.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

5.1.2 Structure of FIPA . . . . . . . . . . . . . . . . . . . . . . . . 40

5.1.3 Developments in FIPA . . . . . . . . . . . . . . . . . . . . . . 41

5.2 FIPA Specifications . . . . . . . . . . . . . . . . . . . . . . . . . 42

5.2.1 Specification Structure . . . . . . . . . . . . . . . . . . . . . 42

5.2.2 Current Generation of FIPA Specifications . . . . . . . . . . . 42

5.2.3 Agent Management . . . . . . . . . . . . . . . . . . . . . . . . . 45
5.3 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

**6 Agent Systems 49**
6.1 JADE and LEAP platforms . . . . . . . . . . . . . . . . . . . . . . . . 49
6.2 Publicly available platforms . . . . . . . . . . . . . . . . . . . . . . . 50
6.2.1 Agent Development Kit . . . . . . . . . . . . . . . . . . . . . . 50
6.2.2 April Agent Platform . . . . . . . . . . . . . . . . . . . . . . . 51
6.2.3 Comtec Agent Platform . . . . . . . . . . . . . . . . . . . . . . 51
6.2.4 FIPA-OS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
6.2.5 Grasshopper . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
6.2.6 JACK . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
6.2.7 JADE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
6.2.8 JAS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
6.2.9 LEAP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
6.2.10 ZEUS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56

**7 Agentcities 57**
7.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
7.2 Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
7.3 Methodologies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 58
7.4 Project Results and Achievements . . . . . . . . . . . . . . . . . . . . 61
7.5 Future works . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

**8 Conclusion 66**

# Chapter 1

# Introduction

It is probably quite rare for a software technology to seize the imagination of the computer science community at large. And yet this is precisely what has happened with *autonomous agents and multi-agent systems*, perhaps to a greater degree even than the semantic web itself.

The aim of this deliverable is to survey the state of the art in the area of intelligent agents and multi-agent systems. While we cannot hope to act as an introduction to *all* the issues in a field as rich and diverse as multi-agent systems, the aim is nevertheless to point the reader at the main areas of interest. Note that the article is intended as an *introduction*, not as a specialist, advanced survey.

The article starts — inevitably — by asking the question *what is an agent*? This leads to a brief discussion on the topic of what sort of computer systems are most appropriately conceived and implemented as multi-agent systems. A crude classification scheme is then introduced, whereby the issues relating to the design and implementation of multi-agent systems are divided into *micro* (agent-level) issues and *macro* (society-level) issues. In section 1.2, micro-level issues (essentially, what software structure should an agent have?) are discussed in more detail. Traditional symbolic AI architectures for agents are reviewed, as well as alternative, *reactive* architectures, and finally, various *hybrid* architectures. One particularly well-known agent architecture is discussed in detail: the *Procedural Reasoning System* (PRS) [GL87].

## 1.1 What are Agents?

Like the question *what is intelligence?* in mainstream AI, the question *what is an agent?* is the most frequently-asked question in multi-agent systems research. Everyone working in the field has their own interpretation of the term, and their own ideas about what the important issues are. Rather than enter a debate about the issue, we shall here simply give an example scenario describing computer systems that it seems useful to think of as agents (this example is borrowed from [WJ95]):

> The key air-traffic control systems in the country of Ruritania suddenly fail, due to freak weather conditions. Fortunately, computerized air-traffic control systems in neighboring countries negotiate between themselves to track and deal with all affected flights, and the potentially disastrous situation passes without major incident.

The computer systems — agents — operating in this scenario . . .

- . . . *are situated in a constantly changing* environment;

  Unlike the theorem provers and expert systems of early AI research, agents operate both *in* and *on* some environment, which may be (in this case) the world of air-traffic control, the real world (in the case of a physically embodied robot), the INTERNET (in the case of network agents [Whi94]), or a software environment such as UNIX (in the case of softbots [ELS94]).

- . . . *have only partial, possibly incorrect information about the environment, and are able to make (at best) limited predictions about what the future will hold*;

  The agents in this example are able to perceive their environment through radar and radio contact with pilots; clearly, the information they obtain in this way, (particularly about variables like the weather), is limited and prone to error. Moreover, any predictions the agent makes (e.g., about tomorrow's weather) will be liable to errors.

- . . . *are able to act upon the environment in order to change it, but have at best partial control over the results of their actions*;

  The agents in this scenario do not have complete control over their environment: in particular, they have no control over 'nature' (in the form of, for example, weather). They do have *limited* control over *aspects* of the environment (in particular, they can instruct pilots about what course to fly, altitude, speed, and so on), but they cannot rely upon this control being perfect (pilots can ignore or misunderstand instructions).

- . . . *have possibly conflicting tasks they must perform*;

  More than one re-routed aircraft may wish to land on the same runway at the same time. In circumstances such as this, where the agent cannot achieve all the tasks it has been allocated, it must fix upon some subset of these tasks and *commit* to realizing them (see the discussion on BDI architectures in section 1.2).

- . . . *have available many different possible courses of action*;

  There will typically be many different ways that an agent can achieve its tasks; the agent must select some procedures, that it believes will achieve its selected tasks, and commit to performing them. Moreover, the agents should pick the procedures that are in some sense the *best*.

- . . . *are required to make decisions in a timely fashion*.

  Real agents do not have infinite resources: the world changes in real time, and agents must make the best (most *rational*) decisions possible, given the resources (information, time, computational power) available to them [RSP93].

(These points are borrowed from [RG95a, p313], where they are discussed in more detail.) It should be clear to anyone with more than a passing appreciation of software engineering that the design and implementation of computer systems that can operate under these constraints is *extremely* difficult. Some researchers (e.g., [RG95a]) believe that such computer systems may usefully be considered as *multi-agent* systems, and that the tools and techniques of multi-agent systems research may fruitfully be used to develop them. Clearly, the scenario above is rather extreme (it is hard to think of many more difficult computer systems to build!), but the agent concept is likely to be useful even in domains that do not exhibit such extreme properties.

Let us try to identify the key properties enjoyed by agents such as those in the scenario above [WJ95]:

- *autonomy*: agents operate without the direct intervention of humans or others, and have control over their actions and internal state;
- *social ability*: agents are able to cooperate with humans or other agents in order to achieve their tasks;
- *reactivity*: agents perceive their environment, and respond in a timely fashion to changes that occur in it;
- *pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by *taking the initiative*.

For some researches, an intelligent agent is a system that enjoys these properties. Others argue that different properties should receive greater emphasis. Some other properties discussed in the context of agency are:

- *mobility*: the ability of an agent to move around an electronic network [Whi94];
- *veracity*: the assumption that an agent will not knowingly communicate false information [Gal88];
- *benevolence*: the assumption that agents share common goals, and that every agent will therefore always try to do what is asked of it [RG85];

- *rationality*: the assumption that an agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved — at least insofar as its beliefs permit [RW91];
- *learning*: the assumption that an agent will adapt itself to fit its environment.

For others — particularly those working in AI — the term 'agent' has a stronger meaning. They mean an agent to be a computer system that, in addition to having the properties specified above, is either designed or implemented in terms of concepts more usually applied to humans. One manifestation of this idea, called *agent-oriented programming*, is discussed in section 1.2.

## 1.2 Micro and Macro Issues

If one aims to build a computer system such as that indicated in the example scenario, then there are at least two sorts of issues that one needs to address [Gil95, p145]:

- *Micro-issues*. How does one build an agent that has the kind of properties listed above? This is the area that has become known as *agent architectures*.
- *Macro-issues*. How does one design an agent society that can (co-)operate effectively? This latter area has been the primary focus of research in *Distributed AI (DAI)* [Huh87, BG88, GH89].

These two sets of issues are by no means disjoint: the traditional AI approach to building a system that can operate intelligently in some environment proposes giving that system some symbolic representation of the environment. If an agent is to co-operate effectively with other agents, therefore, one might wish to give that agent a symbolic representation of the cooperative process (see, e.g., Jennings' *cooperation knowledge level* [Jen92], and the co-operation level in the INTERRAP architecture [Mül97]).

The micro/macro distinction has been criticized by a number of researchers. One objection, for example, is that agents can be comprised of a number of other agents, in the same way that any complex system can be decomposed into a number of other subsystems [HRHW$^+$89]. The micro/macro distinction makes little sense if one takes such a view. As the preceding paragraph illustrated, there is certainly a somewhat grey area between micro and macro issues. However, for the purposes of this review, it seems a useful classification scheme.

### Micro (Agent Level) Issues

This section presents a short survey of the area known as *agent architectures*. Researchers working in this area are concerned with the design and construction of agents that enjoy the properties of autonomy, reactivity, pro-activeness, and social ability, as discussed earlier. Following [WJ95], three classes of agent architecture are identified: deliberative, or symbolic architectures, are those designed along the lines proposed by traditional, symbolic AI; reactive architectures are those that eschew central symbolic representations of the agent's environment, and do not rely on symbolic reasoning; and finally, hybrid architectures are those that try to marry the deliberative and reactive approaches.

### Macro (Societal) Issues

So far, this article has considered only the issues associated with intelligent agent design. We have been concerned with *individuals*, but not with *societies*. The design of 'an agent' is a classic AI pursuit: in a sense, this is what the whole of the AI endeavour has been directed towards. However, AI has traditionally not considered the *societal* aspects of agency. In the late 1970s and early 1980s, these aspects of agency began to be studied in a subfield of AI known as Distributed AI (DAI). In this section, our aim is to briefly review work in DAI: we begin by considering the distinction between distributed problem solving and multi-agent systems, and go on to examine the issues of coherence and coordination, communication, cooperation, and negotiation. The best reference to these various issues is the collection of papers edited by Bond and Gasser [BG88].

# Chapter 2

# Intelligent Autonomous Agents

## 2.1 Logical/Reasoning Architectures

The traditional approach to designing agents is to view them as a type of knowledge-based system. Such architectures are often called *deliberate* or *deliberative*. The key questions to be answered when developing an agent using traditional knowledge-based systems techniques are thus what knowledge needs to be represented, how this knowledge is to be represented, what reasoning mechanisms are to be used, and so on.

The symbolic approach to building agents has traditionally been very closely associated with the *AI planning* paradigm [Geo87]. This paradigm traces its origins to Newell and Simon's GPS system [NS61], but is most commonly associated with the STRIPS planning system ([FN71]) and its descendents (such as [Sac74, Sac75, Cha87, Wil88]). A typical STRIPS-style planning agent will have at least the following components:

- a symbolic model of the agent's *environment*, typically represented in some limited subset of first-order predicate logic;
- a symbolic specification of the actions available to the agent, typically represented in terms of PDA (pre-condition, delete, add) lists, which specify both the circumstances under which an action may be performed and the effects of that action;
- a planning algorithm, which takes as input the representation of the environment, a set of action specifications, and a representation of a goal state, and produces as output a plan — essentially, a program — which specifies how the agent can act so as to achieve the goal.

Thus, planning agents decide how to act from *first principles*. That is, in order to satisfy a goal, they first formulate an entirely new plan or program for that goal. We can thus think of planning agent continually executing a cycle of picking a new goal $\varphi_1$, generating a plan $\pi$ for $\varphi_1$, executing $\pi$, picking a new goal $\varphi_2$, and so on. Unfortunately, first-principles planning of the type just described has associated with it a number of difficulties. The most obvious of these is that the processes of finding a goal, generating a plan to achieve it, and executing it are not atomic: they take time — in some cases, a considerable amount of time. Yet there is an assumption implicit in this scheme, that the environment in which the agent is situated does not change so as to invalidate the preconditions of the plan either while the plan is being formed or while it is being executed. Clearly, in any even moderately dynamic domain, this assumption simply will not hold.

There are a number of theoretical results which indicate that first-principles planning is not a viable option for agents that operate in time-constrained environments. For example, Chapman demonstrated that in many circumstances, first-principles planning is *undecidable* [Cha87]. So, building *reactive* agents, that can truly respond to changes in their environment, is not likely to be possible using first-principles planning techniques. Chapman's negative results, (and other intractability results), have caused many researchers to look for alternative paradigms within which to construct agents; such attempts are reviewed below.

Despite the negative results of Chapman and others, planning is still regarded as an important ability for agents, and many attempts have been made to build planners that, for example, interleave planning, plan execution, and monitoring of plans. One example of such a system is Ambros-Ingerson and Steel's IPEM system [AIS88]. Another example is Vere and Bickmore's HOMER system, in which a simulated submarine agent is given English-language instructions about goals to achieve in a changing environment [VB90].

As well as AI planning systems, a great deal of work has been carried out on deliberative agents within the paradigm known as *agent-oriented programming*. Within AI, it is common practice to characterize the state of an agent in terms of *mentalistic* notions such as *belief*, *desire*, and *intention*. The rationale is that these notions are *abstraction tools* used by us in everyday language to explain and predict the behaviour of complex intelligent systems: people. Just as we can use these notions to explain the behaviour of people, so, the reasoning goes, we can use them to predict, explain, and, crucially, even *program* complex computer systems. This proposal, in its best-known form, was made around 1989 by Yoav Shoham, building on ideas from John McCarthy and others [Sho90, Sho93], and is known as *agent-oriented programming* (AOP). In order to demonstrate the AOP paradigm, Shoham defined a simple experimental language, AGENT0. This language allowed one to specify the behaviour of agents in terms of rules that define how an agent generates *commitments* from the messages it receives and beliefs it holds. The agent continually executes a cycle of receiving messages, updating beliefs, generating commitments, and attempting to discharge current commitments. Building on Shoham's work, many others have developed agent programming environments based on similar ideas. For example: Becky Thomas described an extension to AGENT0 called PLACA [Tho93, Tho95]; Wooldridge and Vandekerckhove developed an AGENT0-like multi-agent testbed called MYWORLD [Woo95]; Poggi described an agent-oriented extension to the CUBL concurrent object language [Pog95]; Weerasooriya *et al* discuss a (hypothetical) agent-oriented programming language called AGENTSPEAK [WRR95];

Fisher has developed a multi-agent programming environment based on executable temporal logic, which enjoys many of the properties of AOP [Fis94, Fis95]; Burkhard discusses some issues in the design of agent-oriented environments [Bur95]; and finally, Lespérance *et al* describe a logic-based multi-agent programming environment called CONGOLOG [LLL$^+$96], which incorporates many ideas from AOP.

## 2.2 Reactive Architectures

In addition to the criticisms outlined above, there are many other objections to symbolic to AI and deliberative agents, some of which have led researchers to seek alternative approaches for building agents. Perhaps the best-known proponent of this 'alternative AI' is Rodney Brooks, who, starting in about 1985, began development of the *subsumption architecture* [Bro86, Bro90, Bro91b, Bro91a]. The basic idea of the subsumption architecture is to design an agent as a set of *task accomplishing behaviours*, arranged into a *subsumption hierarchy*. Each task behaviour is implemented as a simple finite-state machine, which directly maps sensor input to effector output. The layers interact with each other via suppression and inhibition actions. For example, a lower layer in the hierarchy, representing low-level behaviour (such as avoiding obstacles) may suppress higher layers that represent more abstract behaviours (such as exploring or avoiding obstacles). The process of designing an agent becomes one of systematically adding and experimenting with behaviours.

It should be stressed that Brooks' systems do no symbolic reasoning at all. They are, in a sense, extremely simple in computational terms. And yet experiments have shown that agents implemented using Brooks' scheme can achieve near optimal results [Ste90].

A number of other researchers have developed approaches to agent design that borrow from Brooks' work. Some well-known examples are Pattie Maes' agent network architecture [Mae90b, Mae91], and the ABLE and REAL-TIME ABLE (RTA) languages developed at Philips research labs [CW90, Wav92, WG95]. Other approaches to reactive architectures are [Kae86, Fir87, AC87, RK86]; the book edited by Maes contains many relevant papers and references [Mae90a].

## 2.3 Hybrid Architectures

Some researchers strongly believe that the traditional, symbolic approach is the best candidate for the future development of AI; others (such as Brooks) just as strongly assert that symbolic AI is a dead end, and that alternative approaches are required. Still others accept that both arguments have their merits, and suggest that the best direction for future research is to try to marry the two styles of architecture. This has led to work on *hybrid architectures*.

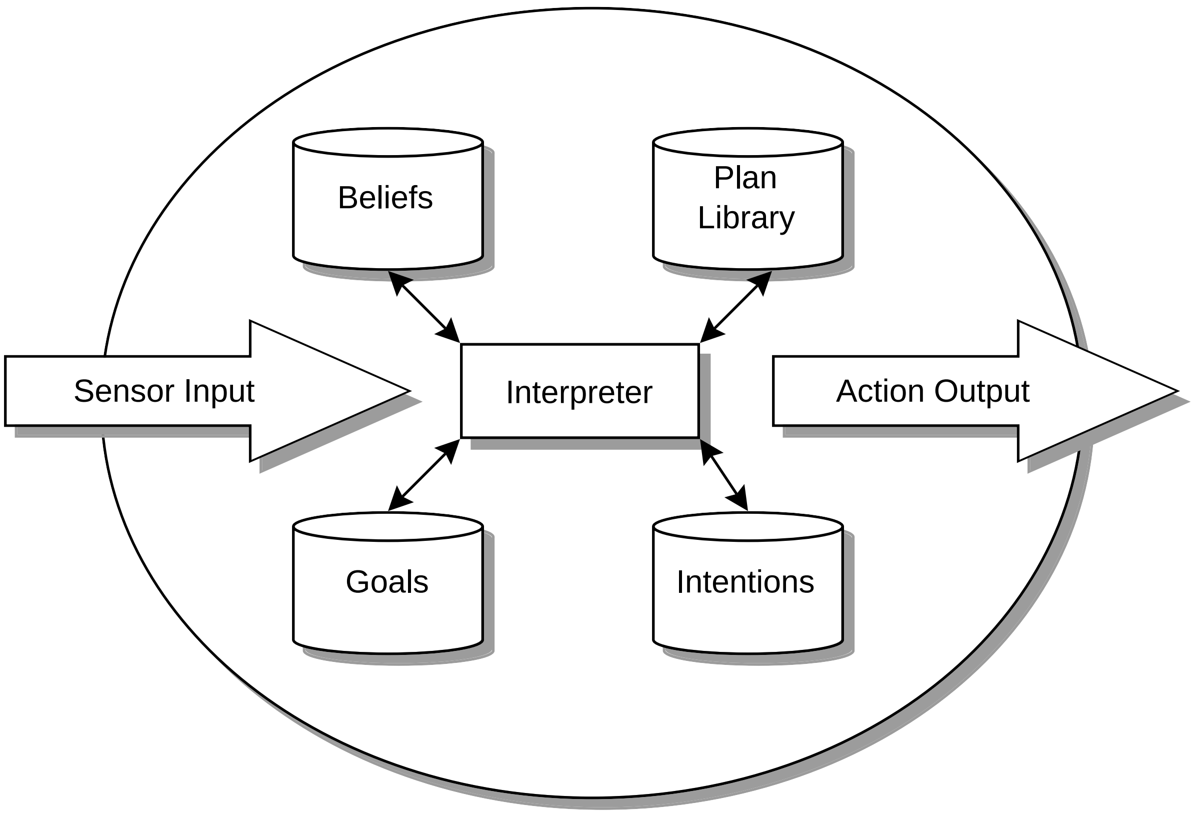

Figure 2.1: The PRS — A BDI Agent Architecture

### 2.3.1 The Procedural Reasoning System (PRS)

Perhaps the best-known agent architecture is the *Procedural Reasoning System (PRS)* developed by Georgeff *et al* [GL87]. The PRS is illustrated in Figure 2.1. The PRS is an example of a currently popular paradigm for agent design known as the *belief-desire-intention* (BDI) approach [BIP88]. As this figure shows, a BDI architecture typically contains four key data structures. An agent's *beliefs* correspond to information the agent has about the world, which may be incomplete or incorrect. Beliefs may be as simple as variables, (in the sense of, e.g., PASCAL programs), but implemented BDI agents typically represent beliefs symbolically (e.g., as PROLOG-like facts [GL87]). An agent's *desires* intuitively correspond to the tasks allocated to it. (Implemented BDI agents require that desires be logically consistent, although *human* desires often fail in this respect.)

An agent's *intentions* represent desires that it has committed to achieving. The intuition is that an agent will not, in general, be able to achieve *all* its desires, even if these desires *are* consistent. Agents must therefore fix upon some subset of available desires and commit resources to achieving them. These chosen desires, which the agent has committed to achieving, are *intentions* [CL90a]. An agent will typically continue to try to achieve an intention until either it believes the intention is satisfied, or else it believes the intention is no longer achievable [CL90a].

The final data structure in a BDI agent is a *plan library*. A plan library is a set of plans (a.k.a. *recipes*) which specify courses of action that may be followed by an agent in order to achieve its intentions. An agent's plan library represents its *procedural knowledge*, or *know-how*. A plan contains two parts: a *body*, or *program*, which defines a course of action; and a *descriptor*, which states both the circumstances under which the plan can be used (i.e., its pre-condition), and what intentions the plan may be used in order to achieve (i.e., its post-condition). PRS agents do no first-principles planning at all.

The *interpreter* in Figure 2.1 is responsible for updating beliefs from observations made of the world, generating new desires (tasks) on the basis of new beliefs, and selecting from the set of currently active desires some subset to act as intentions. Finally, the interpreter must select an action to perform on the basis of the agent's current intentions and procedural knowledge.

In order to give a formal semantics to BDI architectures, a range of *BDI logics* have been developed by Rao and Georgeff [RG91, RG95b]. These logics are extensions to the branching time logic CTL* [EH86], which also contain normal modal connectives for representing beliefs, desires, and intentions. Most work on BDI logics has focussed on possible relationships between the three 'mental states' [RG91], and more recently, on developing proof methods for restricted forms of the logics [RG95b]. In related work, attempts have been made to graft a logic of plans onto the basic BDI framework, in order to represent an agent's procedural knowledge [RGS92, KLR$^+$92].

### TouringMachines

An increasingly popular approach to designing hybrid agents is to use a *layered* architecture. The idea of layered architectures is well-established in traditional systems control theory; in AI, it has been popularized by the work of Brooks (see above) and Kaelbling [Kae86], among others. The basic idea of a layered architecture is that of integrating different agent control subsystems by *layering* them. Figure 2.2 illustrates a good example of a layered agent architecture: Innes Ferguson's TOURINGMACHINES [Fer92b, Fer92a, Fer95]. As this Figure shows, TOURINGMACHINES consists of three *activity producing layers*. That is, each layer continually produces 'suggestions' for what actions the agent should perform. The *reactive layer* provides an immediate response to changes that occur in the environment — it is implemented as a set of situation-action rules, like the behaviours in Brooks' subsumption architecture. These rules map sensor input directly to effector output. The planning layer contains a plan library (much like the PRS — see above), which the agent can use in order to achieve goals. The *modeling* layer represents the various entities in the world (including the agent itself, as well as other agents); the modeling layer predicts conflicts between agents, and generates new goals to be achieved in order to resolve these conflicts, which are posted down to the planning layer, which

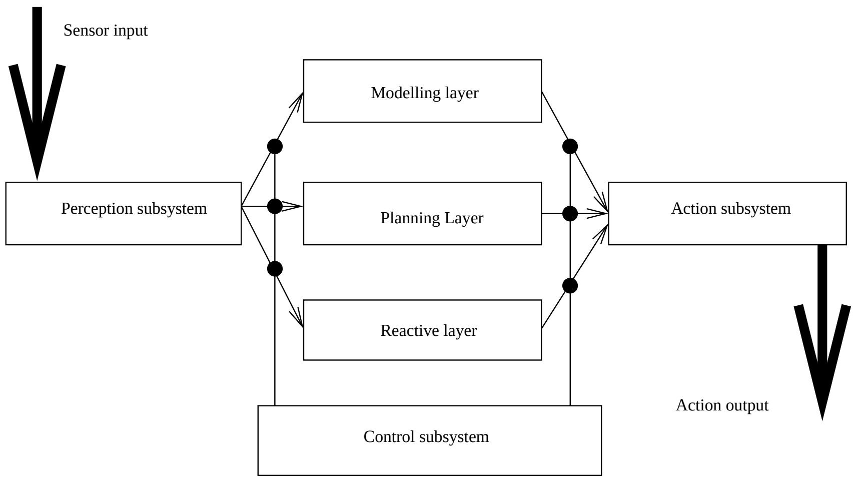

Figure 2.2: TOURINGMACHINES: A Layered Agent Architecture

then determines how to satisfy them. The three layers are embedded within a control subsystem, which is responsible for deciding which of the layers should have control over the agent.

Another example of a layered architecture is INTERRAP [Mül97]. Like TOURINGMACHINES, the INTERRAP architecture has three layers, with the lowest two layers corresponding fairly closely to Ferguson's reactive and planning layers. However, the highest layer in INTERRAP deals with *social* aspects of the system: it not only models other agents, but also explicitly represents social activities (such as cooperative problem solving). The 3T architecture is another layered architecture, that has been used in several real-world robotic agents [BKMS96] — we will describe this architecture in more detail.

### 2.3.2 The 3T Architecture

Like TouringMachines and INTERRAP, the 3T architecture is another example of a three-level architecture for agent control. An overview of the architecture is given in Figure 2.3, while a detailed description is given in Figure 2.4.

The three layers in the 3T architecture are as follows (from least to most abstract):

- *reactive skills*;
- *skill sequencing*; and
- *deliberation and high-level planning*.

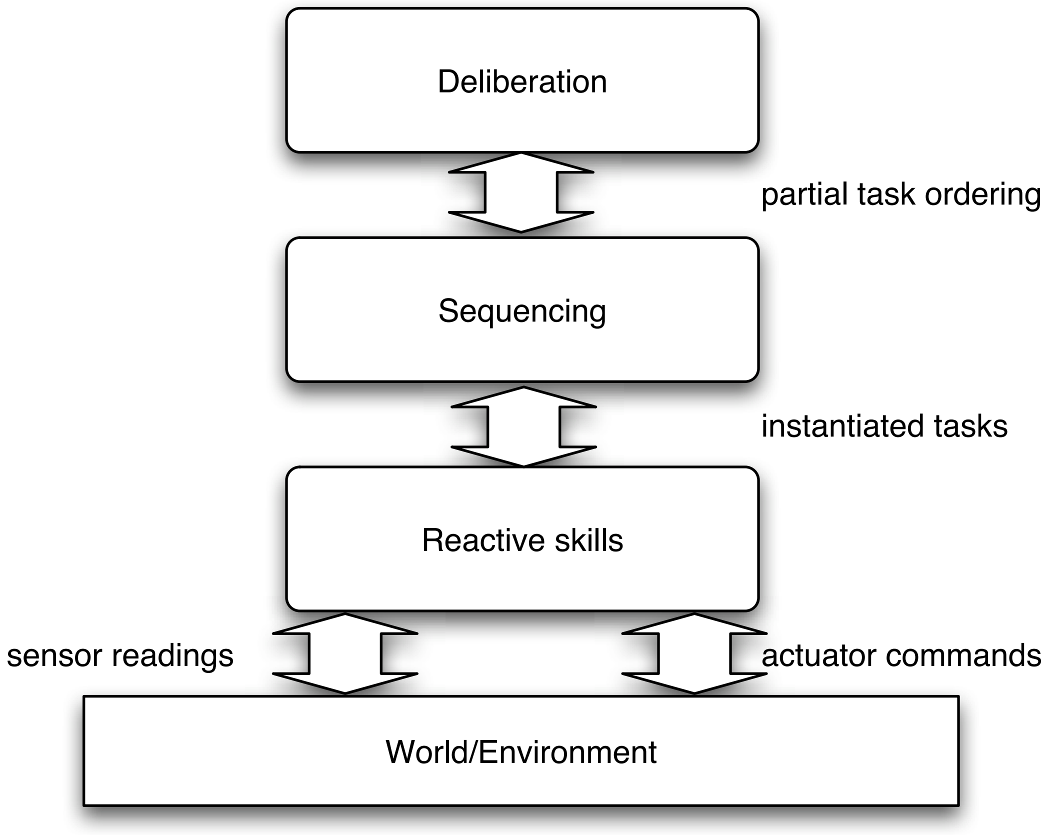

Figure 2.3: The 3T architecture

A *skill* in the 3T architecture is a primitive behaviour that captures a particular kind of ability. For example, a skill might be the ability of a mobile robot to move from one location in a warehouse to another location. We can draw an analogy between skills and (for example), “native methods” in Java. Ultimately, whatever a 3T agent does will involve the execution of particular skills. The skills layer thus contains a set of such skills, and typically, one of these skills will be being executed at any given time. Skills are actually implemented in 3T using Firby’s *reactive action packages* [Fir87].

The *sequencing* layer of 3T is responsible for selecting and instantiating individual skills. Finally, the *planning* layer is responsible for synthesising high-level plans for the agent, including deadlines for the completion of activity and plans for the allocation of resources.

The following example (taken verbatim from [BFG$^+$97]) explains the role of the three layers in more detail:

Imagine a repair robot charging in a docking bay on a space station. At the beginning of a typical day, there will be several routine maintenance tasks to perform on the outside of the station, such as retrieving broken items or inspecting power levels. In addition, a human supervisor assigns the robot a set of inspection and repair tasks at a number of sites around the station. The planner synthesizes all of these goals into a partially ordered plan listing tasks for the robot to perform. These tasks would call for the robot to move from site to site conducting the appropriate repair or inspection at each site. [For example] (1) navigate to the `camera-site-1`, (2) attach to `camera-site-1`, (3) unload a repaired camera, and (4) detach from `camera-site-1`. Each of these corresponds to one or more skills. [. . . ] The [sequencing layer] activates a specific set of skills at the skill level. [. . . ] The activated skills (reactive layer) will move the state of the world in a direction that should cause the desired [state of affairs].

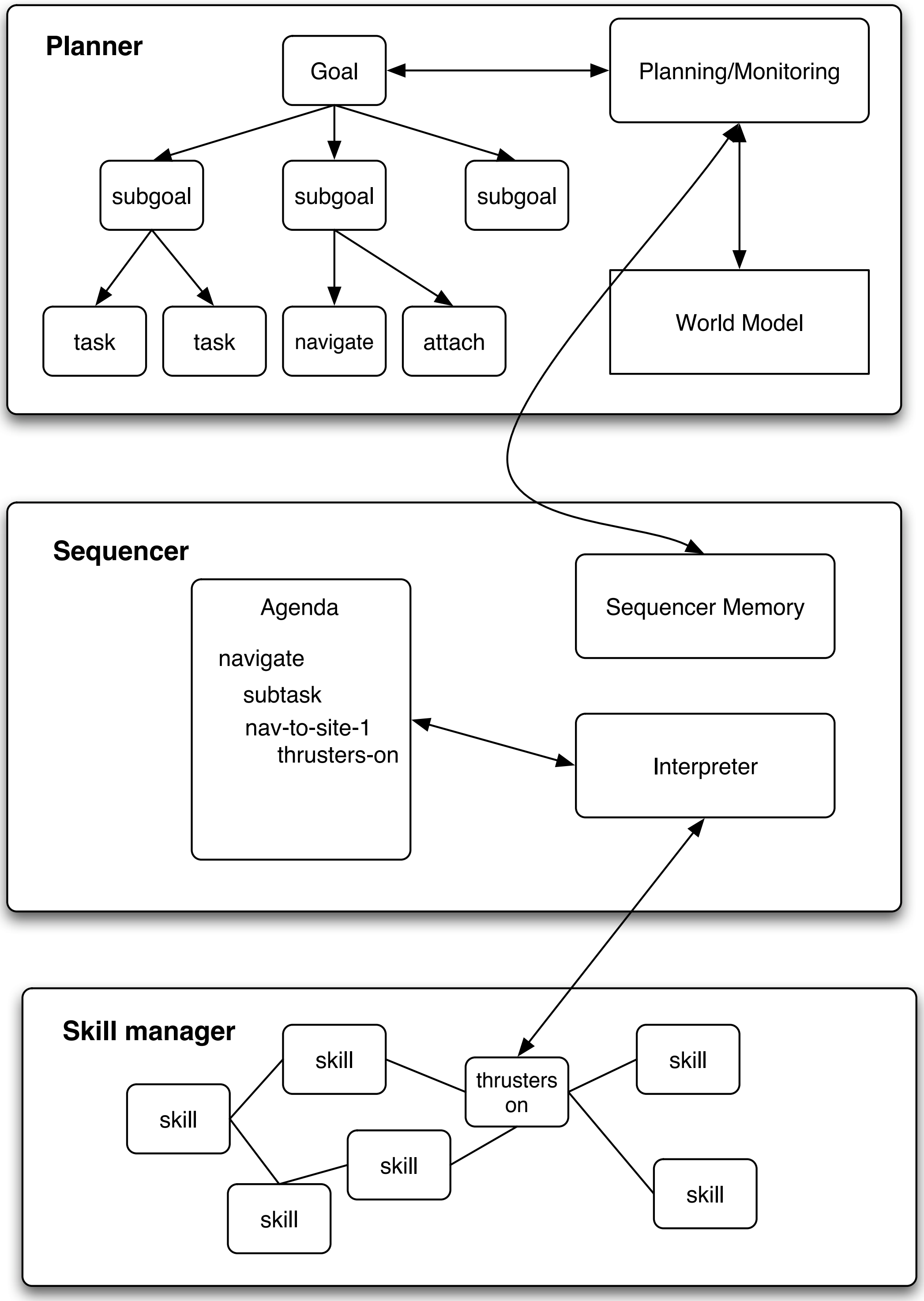

Figure 2.4: The 3T architecture — A Detailed View

# Chapter 3

# Negotiation and Agreement

Automated negotiation is one of the most distinctive research topics in the multiagent systems arena. In the field of automated negotiation, we attempt to develop protocols and algorithms through which software agents can autonomously reach agreements on matters of common interest. It seems to me that the term ”negotiation” is used with reckless abandon throughout computer science. For example, one hears of a modem “negotiating” a baud rate with a host computer, and of course there are many other similar such usages. The term negotiation, used in this sense, is rather trivial – it simply means that the two processes use some pre-agreed procedure to fix on a baud rate, which amounts to not much more than selecting an option from a set of choices. What marks out negotiation as studied in multiagent systems is the idea that the processes (agents) involved are *self interested*: they are attempting to get the best deal for themselves that they can, and what is best for one is not necessarily best for the other. As a consequence, multiagent negotiation settings take on the character of *games*, in the sense that the term is used in the formal setting of game theory. To bring about their most preferred outcome in such settings agents must typically engage in strategic reasoning – that is, considering how other negotiation participants are likely to act, as self-interested, rational entities.

A number of different types of agreement have been discussed in the literature, which we shall survey here (see Figure 3.1):

**Argumentation:** Argumentation originally developed as the study of how people reach agreements, particularly when these agreements relate to beliefs or principles that they hold. Perhaps the paradigm example of argumentation is the formal process of argument that takes place in a court of law, where advocates for the prosecution and defence progressively put forward arguments in an attempt to convince a jury of the guilt or innocence of a defendant. In the multi-agent systems community, models of argument have been developed in an attempt to provide a clear underpinning to this process, and enable software agents to be able to engage in this kind of discourse. Here, we shall survey just one particularly influential model of argument: the *abstract argument* model of Dung [Dun95].

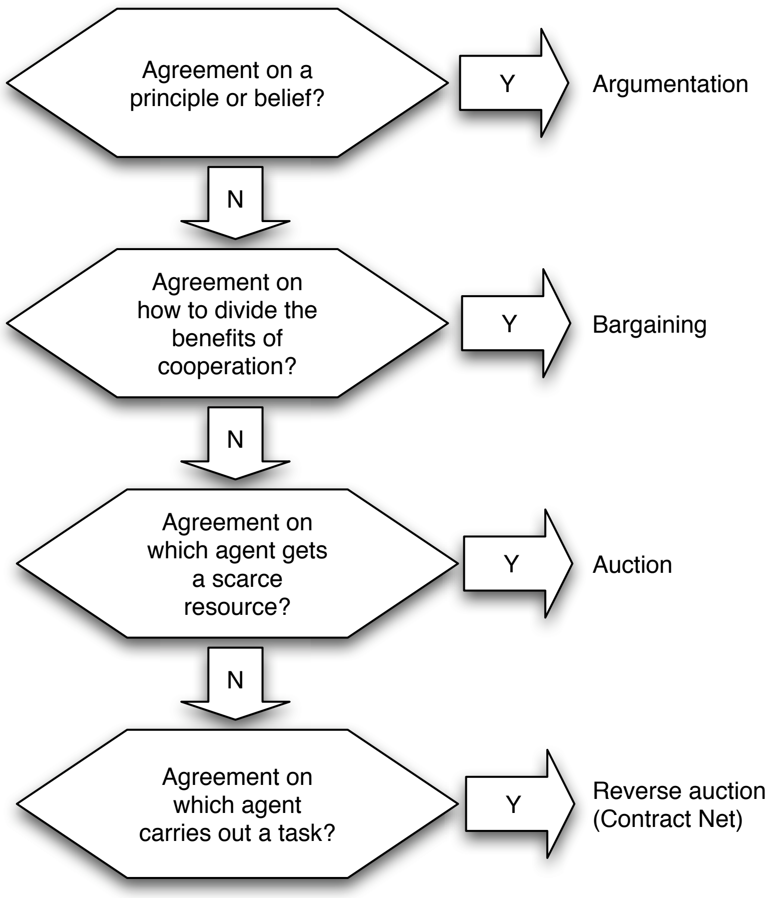
Agreement on a principle or belief?
Y
Argumentation
N
Agreement on how to divide the benefits of cooperation?
Y
Bargaining
N
Agreement on which agent gets a scarce resource?
Y
Auction
N
Agreement on which agent carries out a task?
Y
Reverse auction (Contract Net)

Figure 3.1: Types of agreement

**Bargaining:** Bargaining (also known as one-to-one negotiation) is the process by which two agents attempt to reach agreement on how to divide the benefits of their cooperation. It typically takes place in a series of negotiation steps, where each side puts forward a proposal. We shall survey one particularly influential type of bargaining: the *alternating offers* protocol of Rubinstein [OR90, Kra01].

**Auctions:** Auctions are a mechanism by which it is possible to efficiently allocate some (scarce) resource to some agent. Auctions have received a great deal of attention in the multi-agent systems literature, because the advent of the Internet has made it possible to run auctions with a very low overhead cost. We will briefly survey the main types of auctions for multi-agent systems.

**Reverse auctions:** A reverse auction takes place when, instead of a number of bidders attempting to gain a scarce resource, the owner of that resource directly contacts the bidders in an attempt to solicit bidders. Reverse auctions, in the form of the *Contract Net Protocol*, are the dominant mechanism for allocating tasks in multi-agent systems. As the contract net is discussed elsewhere in this report, we will not discuss it in this section.

## 3.1 Argumentation

Argumentation is the process of one agent attempting to convince another of the acceptability of some position or belief. Several different models of argumentation have been described in the literature – see [PV01] for an overview. The model of argument systems that we work with throughout the remainder of this survey was introduced by Dung [Dun95]. Dung's aim was to introduce an *abstract* model of argumentation; that is, a model of argument systems which abstracts away from what arguments actually *are* or how the notion of "attack" between arguments should be interpreted. Arguments in Dung's model are atomic elements, and the notion of attack is simply represented by a binary relation over these arguments, denoting the structure of attacks in this system. By abstracting away from questions about the content of arguments and the nature of attack, Dung's model makes it possible to focus on the status of arguments – whether they are acceptable or not.

Formally, a *Dung-style argumentation framework* (DAF) is a pair $\mathcal{D} = (X, A)$, where $X = \{\alpha, \ldots\}$ is a (fixed, finite, non-empty) set of *arguments*, and $A \subseteq X \times X$ is the *attack* relation of $\mathcal{D}$, with the intended interpretation that if $(\alpha, \alpha') \in A$ then $\alpha$ attacks $\alpha'$. Let $in(\alpha)$ and $out(\alpha)$ denote the sets of arguments that attack $\alpha$ and that $\alpha$ attacks, respectively:

$$\begin{array}{rcl} in(\alpha) & = & \{\alpha' \in X \mid (\alpha', \alpha) \in A\} \\ out(\alpha) & = & \{\alpha' \in X \mid (\alpha, \alpha') \in A\} \end{array}$$

At this point, let us consider a small example.

> Consider the simple DAF $\mathcal{D}_1 = (\{p, q, r, s\}, \{(r, q), (s, q), (q, p)\})$; this argument system is illustrated in Figure 3.2 (vertices correspond to arguments and edges correspond to attacks). In this system, the arguments $r$ and $s$ have no attackers, while $q$ is attacked by both $r$ and $s$, and $p$ is attacked by $q$. Thus, for example, $in(q) = \{r, s\}$ while $out(q) = \{p\}$.

Given these basic definitions, we can define various notions of when an argument is *acceptable* [Dun95]. The basic idea is that an acceptable argument is one that a rational agent must accept. Unfortunately, there is no universally accepted definition of acceptability – there are a number of different notions, each with their own advantages and disadvantages. Here we shall spell out the key ones.

$$\begin{tikzcd}
r \arrow[dr] & & \\
 & q \arrow[r] & p \\
s \arrow[ur] & &
\end{tikzcd}$$

Figure 3.2: The abstract argument system $\mathcal{D}_1$.

## Preferred Extensions

The first notion of acceptability that we define is that of a *preferred extension*. First, given a DAF $\mathcal{D} = (X, A)$ an argument $\alpha \in X$, and a set $S \subseteq X$, we say that $x$ is *acceptable* with respect to $S$ if any attacker of $\alpha$ is itself attacked by some member of $S$. We write $acc(\alpha, S)$ to denote the fact that $\alpha$ is acceptable with respect to $S$:

$$acc(\alpha, S) \text{ iff } \forall \alpha' \in X : (\alpha', \alpha) \in A \Rightarrow \exists \alpha'' \in S : (\alpha'', \alpha') \in S.$$

A set of arguments $S$ is *conflict free* if no member of this set attacks any other member of this set; we write $cf(S)$ to indicate that $S$ is conflict free.

$$cf(S) \text{ iff } \forall \alpha, \alpha' : (\alpha, \alpha') \notin A \ \& \ (\alpha', \alpha) \notin A.$$

A set of arguments $S$ is *admissible* if it is conflict free, and every member of $S$ is acceptable with respect to $S$; we write $adm(S)$ to indicate that $S$ is admissible:

$$adm(S) \text{ iff } cf(S) \ \& \ \forall \alpha \in S : acc(\alpha, S).$$

Finally, $S$ is a *preferred extension* of $\mathcal{D}$ is $S$ is a maximal admissible set with respect to set inclusion, i.e., if $S$ is admissible, and every superset of $S$ is inadmissible. We write $pe(S)$ to indicate that $S$ is a preferred extension:

$$pe(S) \text{ iff } adm(S) \ \& \ \forall S' \supset S : \neg adm(S).$$

Recall the DAF $\mathcal{D}_1$, given earlier. In this system, the $\{p, r, s\}$ is conflict free, since no member of this set attacks any other member. However, any set containing $q$ and at least one other member is not conflict free, since both $r$ and $s$ attack $q$ and $q$ attacks $p$. The set $\{p, r, s\}$ is also acceptable with respect to itself: although $p$ is attacked by $q$, the argument $q$ is attacked by both $r$ and $s$. Thus $\{p, r, s\}$ is admissible, and since it is not possible to add any argument from $\mathcal{D}_1$ to this set without making it inadmissible, then $\{p, r, s\}$ is a preferred extension of $\mathcal{D}_1$ (In fact, it is the only preferred extension of $\mathcal{D}_1$.)

The concept of a preferred extension provides perhaps the most common and important definition of when an argument is “reasonable”. The Dung proved that every DAF has at least one preferred extension, although this may be the empty set [Dun95].

We say that $\alpha$ is *sceptically accepted* if it is a member of every preferred extension, and *credulously accepted* if it is a member of at least one preferred extension. Since there will alwatys be at least one preferred extension, it follows that any argument that is sceptically accepted is also credulously accepted. We use $sc(\alpha)$ and $cr(\alpha)$ to indicate that $\alpha$ is sceptically and credulously accepted, respectively:

$$\begin{array}{lll} sc(\alpha) & \text{iff} & \forall S \subseteq A : pe(S) \Rightarrow \alpha \in S \\ cr(\alpha) & \text{iff} & \exists S \subseteq A : pe(S) \;\&\; \alpha \in S \end{array}$$

Returning again to $\mathcal{D}_1$, the arguments $p$, $r$, and $s$ are all sceptically accepted (and hence credulously accepted). The argument $q$ is neither sceptically nor credulously accepted.

The idea of a preferred extension thus captures the concept of a mutually supportive set of arguments, which together present a coherent interpretation to the world. However, the fact that there may be a single preferred extension may not be useful in practice. Moreover, the fact that there may be *multiple* preferred extensions means that there may be multiple possible putatively acceptable sets of arguments according to this definition – in which case, which one should we choose as “the” interpretation?

**Grounded Extensions**

An alternative notion of an acceptable set of arguments, also defined by Dung, is that of the *grounded* extension. The idea is as follows. Given an abstract argument system, (such as $\mathcal{D}_1$, above) we start by looking for arguments that are *guaranteed* to be acceptable: those that could not possibly be unacceptable under any sensible definition of acceptability. The most obvious candidates for such acceptable arguments are those that have *no attackers whatsoever*: for surely, the status of arguments that have no attackers whatsoever must be beyond question. Having identified such arguments, we can then proceed to eliminate those that are attacked by these: since these arguments are attacked by arguments that are surely in, then these arguments must surely be out. We can then iterate this process, deleting arguments that are attacked by arguments that are surely in, until we find no changes occurring to the argument graph that remains. We refer to the graph that remains after this as the *grounded extension* of the argument system. The algorithm to compute the grounded extension of a Dung argument system is given in Figure 3.3.

If we apply the algorithm in Figure 3.3 to the argument system $\mathcal{D}_1$, then we obtain the following sequence of values for the variable $new$:

```
function $ge(X, A)$ returns a subset of $X$
begin
    $old := \emptyset$
    $new := \{\alpha \mid acc(\alpha, old)\}$
    while $old \neq new$ do
        $old := new$
        $new := \{\alpha \mid acc(\alpha, old)\}$
    end-while
    return $new$
end function $ge$
```

Figure 3.3: Computing a grounded extension.

$$\begin{array}{lclclcl} old & = & \emptyset & new & = & \{r, s\} \\ old & = & \{r, s\} & new & = & \{r, s, p\} \\ old & = & \{r, s, p\} & new & = & \{r, s, p\} \end{array}$$

at which point, with $old = new$, the algorithm returns $\{r, s, p\}$ as the grounded extension.

The main advantage of the notion of grounded extension is that there will always be such an extension, and this extension is guaranteed to be unique. However, if there are no arguments that are free of attackers, then the grounded extension will be empty.

Notice that some argument systems appear to be paradoxical. For example, consider the argument system $\mathcal{D}_2 = (\{p, q, r\}, \{(p, q), (q, r), (r, p)\})$. This argument system contains an odd length cycle. Clearly, the grounded extension is empty, and it has a single empty preferred extension. But, nevertheless, surely it should have an interpretation? For example, one could say that each of the singleton sets of arguments $p$, $q$, and $r$ are reasonable in this case.

Argument systems seem to be an interesting way of modelling dialogue between agents that are attempting to convince one-another of certain states of affairs. Implementations of argument systems have been developed, e.g., by Sycara [Syc90] and Kraus et al [KSE98].

## 3.2 Auctions

Auctions used to be comparatively rare in everyday life; every now and then, one would hear of astronomical sums paid at auction for a painting by Monet or Van Gogh, but other than this, they did not enter the lives of the majority. The Internet and Web fundamentally changed this. The Web made it possible for auctions with a large, international audience to be carried out at very low cost. This in turn made it possible for goods to be put up for auction which hitherto would have been too uneconomical. Large businesses have sprung up around the idea of online auctions, with eBay being perhaps the best-known example [EBA01].

One of the reasons why online auctions have become so popular is that auctions are extremely simple interaction scenarios. This means that it is easy to automate auctions; this makes them a good first choice for consideration as a way for agents to reach agreements. Despite their simplicity, auctions present both a rich collection of problems for researchers, and a powerful tool that automated agents can use for allocating goods, tasks, and resources.

Abstractly, an auction takes place between an agent known as the *auctioneer* and a collection of agents known as the *bidders*. The goal of the auction is for the auctioneer to allocate the *good* to one of the bidders. In most settings – and certainly most traditional auction settings – the auctioneer desires to maximize the price at which the good is allocated, while bidders desire to minimize price. The auctioneer will attempt to achieve his desire through the design of an appropriate auction mechanism – the rules of encounter – while bidders attempt to achieve their desires by using a strategy that will conform to the rules of encounter, but that will also deliver an optimal result.

There are several factors that can affect both the protocol and the strategy that agents use. The most important of these is whether the good for auction has a *private* or a *public/common* value. Consider an auction for a one dollar bill. How much is this dollar bill worth to you? Assuming it is a 'typical' dollar bill, then it should be worth exactly $1; if you paid $2 for it, you would be $1 worse off than you were. The same goes for anyone else involved in this auction. A typical dollar bill thus has a *common value*: it is worth exactly the same to all bidders in the auction. However, suppose you were a big fan of the Beatles, and the dollar bill happened to be the last dollar bill that John Lennon spent. Then it may well be that, for sentimental reasons, this dollar bill was worth considerably more to you – you might be willing to pay $100 for it. To a fan of the Rolling Stones, with no interest in or liking for the Beatles, however, the bill might not have the same value. Someone with no interest in the Beatles whatsoever might value the one dollar bill at exactly $1. In this case, the good for auction – the dollar bill – is said to have a *private value*: each agent values it differently.

A third type of valuation is *correlated value*: in such a setting, an agent's valuation of the good depends partly on private factors, and partly on other agent's valuation of it. An example might be where an agent was bidding for a painting that it liked, but wanted to keep open the option of later selling the painting. In this case, the amount you would be willing to pay would depend partly on how much you liked it, but also partly on how much you believed other agents might be willing to pay for it if you put it up for auction later.

Let us turn now to consider some of the dimensions along which auction protocols may vary. The first is that of *winner determination*: who gets the good that the bidders are bidding for. In the auctions with which we are most familiar, the answer to this question is probably self-evident: the agent that bids the most is allocated the good. Such protocols are known as *first-price* auctions. This is not the only possibility, however. A second possibility is to allocate the good to the agent that bid the highest, but this agent pays only the amount of the *second* highest bid. Such auctions are known as *second-price* auctions.

At first sight, it may seem bizarre that there are any settings in which a second-price auction is desirable, as this implies that the auctioneer does not get as much for the good as it could do. However, we shall see below that there are indeed some settings in which a second-price auction is desirable.

The second dimension along which auction protocols can vary is whether or not the bids made by the agents are known to each other. If every agent can see what every other agent is bidding (the terminology is that the bids are *common knowledge*), then the auction is said to be *open cry*. If the agents are not able to determine the bids made by other agents, then the auction is said to be a *sealed-bid* auction.

A third dimension is the mechanism by which bidding proceeds. The simplest possibility is to have a single round of bidding, after which the auctioneer allocates the good to the winner. Such auctions are known as *one shot*. The second possibility is that the price starts low (often at a *reservation price*) and successive bids are for increasingly large amounts. Such auctions are known as *ascending*. The alternative – *descending* – is for the auctioneer to start off with a high value, and to decrease the price in successive rounds.

The main auction types that we may single out from this space of possible auctions are as follows:

**English auctions:** The most well known of all auction types, an English auction is a first price, open cry, ascending auction: the auctioneer starts off by offering the good at a *reserve price*, and bidders make progressively higher bids. Bidders can withdraw at any point in the auction, and the good is allocated to the agent that made the final bid, when there is only one agent left in the auction; the amount they pay is the price of the final bid.

**Dutch auction:** This is a first price, open cry, descending auction. The auctioneer starts off by offering the good at some unrealistically high price, and the price gradually comes down by some small amount until an agent makes a bid – at which point the good is allocated for the amount of this bid.

**First-price sealed bid auctions:** Here, agents make a single offer for the good, which is not seen by any other agent. The good is allocated to the agent that makes the highest bid, and this auction pays the amount they bid.

**Vickrey auctions:** These are *second price sealed bid auctions*: each agent makes a single bid, and the good is allocated to the agent that make the highest bid, but the amount they pay is the amount of the *second highest bid*. Although this seems paradoxical,

> it can easily be proved that the optimal strategy for an agent in such a case is to bid the amount of their private valuation: to bid *exactly* what they think the good is worth, and no more.

The study of auctions is one of the most active ongoing areas of research in the multi-agent systems arena.

## 3.3 Bargaining

Bargaining is the process of two agents attempting to reach an agreement on matters of common interest. Notice that bargaining is generally a *cooperative* endeavour, in the sense that cooperation to reach an outcome must be in the best interests of those participating (otherwise they would not engage in negotiation). So, we can think of negotiation as attempting to reach agreement on the issue of how to *divide the benefits* of negotiation. For example, when negotiating with a second-hand car salesman about the price of a car, you are essentially trying to decide how to distribute the "surplus" benefit gained by cooperation amongst the participants: the car salesman wants as much surplus as possible (i.e., he wants the price to be as high as possible) and you also want as much surplus as possible (i.e., you want the price as low as possible). We can make this idea concrete in the following example of *task sharing* [RZ94].

> Imagine that you have three children, each of whom needs to be delivered to a different school each morning. Your neighbour has four children, and also needs to take them to school. Delivery of each child can be modelled as an indivisible task. You and your neighbour can discuss the situation, and come to an agreement that it is better for both of you (for example, by carrying the other's child to a shared destination, saving him the trip). There is no concern about being able to achieve your task by yourself. The worst that can happen is that you and your neighbour will not come to an agreement about setting up a car pool, in which case you are no worse off than if you were alone. You can only benefit (or do no worse) from your neighbour's tasks.
>
> Assume, though, that one of my children and one of my neighbours' children both go to the same school (that is, the cost of carrying out these two deliveries, or two tasks, is the same as the cost of carrying out one of them). It obviously makes sense for both children to be taken together, and only my neighbour or I will need to make the trip to carry out both tasks.
>
> What kinds of agreement might we reach? We might decide that I will take the children on even days each month, and my neighbour will take them on odd days; perhaps, if there are other children involved, we might have my neighbour always take those two specific children, while I am responsible for the rest of the children. [RZ94, p. 29]

To formalize this kind of situation, Rosenschein and Zlotkin defined the notion of a *task-oriented domain* (TOD). A task-oriented domain is a triple

$$\langle T, Ag, c \rangle,$$

where

- $T$ is the (finite) set of all possible tasks;
- $Ag = \{1, \ldots, n\}$ is the (finite) set of negotiation participant agents;
- $c : \wp T \rightarrow R^+$ is a function which defines the *cost* of executing each subset of tasks: the cost of executing any set of tasks is a positive real number.

The cost function must satisfy two constraints. First, it must be *monotonic*. Intuitively, this means that adding tasks never decreases the cost. Formally, this constraint is defined as follows:

If $T_1, T_2 \subseteq T$ are sets of tasks such that $T_1 \subseteq T_2$, then $c(T_1) \leq c(T_2)$.

The second constraint is that the cost of doing nothing is zero, i.e. $c(\emptyset) = 0$.

An *encounter* within a task-oriented domain $\langle T, Ag, c \rangle$ occurs when the agents $Ag$ are assigned tasks to perform from the set $T$. Intuitively, when an encounter occurs, there is potential for the agents to reach a deal by reallocating the tasks amongst themselves; as we saw in the informal car pool example above, by reallocating the tasks, the agents can potentially do better than if they simply performed their tasks themselves. Formally, an encounter in a TOD $\langle T, Ag, c \rangle$ is a collection of tasks

$$\langle T_1, \ldots, T_n \rangle,$$

where, for all $i$, we have that $i \in Ag$ and $T_i \subseteq T$. Notice that a TOD together with an encounter in this TOD is a type of *task environment*, of the kind we saw in Chapter 2. It defines both the characteristics of the environment in which the agent must operate, together with a task (or rather, set of tasks), which the agent must carry out in the environment.

Hereafter, we will restrict our attention to one-to-one negotiation scenarios, as discussed above: we will assume the two agents in question are $\{1, 2\}$. Now, given an encounter $\langle T_1, T_2 \rangle$, a *deal* will be very similar to an encounter: it will be an allocation of the tasks $T_1 \cup T_2$ to the agents 1 and 2. Formally, a pure deal is a pair $\langle D_1, D_2 \rangle$ where $D_1 \cup D_2 = T_1 \cup T_2$. The semantics of a deal $\langle D_1, D_2 \rangle$ is that agent 1 is committed to performing tasks $D_1$ and agent 2 is committed to performing tasks $D_2$.

The *cost* to agent $i$ of a deal $\delta = \langle D_1, D_2 \rangle$ is defined to be $c(D_i)$, and will be denoted $cost_i(\delta)$. The *utility* of a deal $\delta$ to an agent $i$ is the difference between the cost of agent $i$

doing the tasks $T_i$ that it was originally assigned in the encounter, and the cost $cost_i(\delta)$ of the tasks it is assigned in $\delta$:

$$utility_i(\delta) = c(T_i) - cost_i(\delta).$$

Thus the utility of a deal represents how much the agent has to gain from the deal; if the utility is negative, then the agent is *worse off* than if it simply performed the tasks it was originally allocated in the encounter.

What happens if the agents *fail* to reach agreement? In this case, they must perform the tasks $\langle T_1, T_2 \rangle$ that they were originally allocated. This is the intuition behind the terminology that the *conflict deal*, denoted $\Theta$, is the deal $\langle T_1, T_2 \rangle$ consisting of the tasks originally allocated.

The notion of *dominance*, as discussed in the preceding chapter, can be easily extended to deals. A deal $\delta_1$ is said to dominate deal $\delta_2$ (written $\delta_1 \succ \delta_2$) if and only if the following hold.

1. Deal $\delta_1$ is at least as good for every agent as $\delta_2$:

$$\forall i \in \{1,2\}, utility_i(\delta_1) \geq utility_i(\delta_2).$$

2. Deal $\delta_1$ is better for some agent than $\delta_2$:

$$\exists i \in \{1,2\}, utility_i(\delta_1) > utility_i(\delta_2).$$

If deal $\delta_1$ dominates another deal $\delta_2$, then it should be clear to all participants that $\delta_1$ is better than $\delta_2$. That is, all 'reasonable' participants would prefer $\delta_1$ to $\delta_2$. Deal $\delta_1$ is said to weakly dominate $\delta_2$ (written $\delta_1 \succeq \delta_2$) if at least the first condition holds.

A deal that is not dominated by any other deal is said to be *pareto optimal*. Formally, a deal $\delta$ is pareto optimal if there is no deal $\delta'$ such that $\delta' \succ \delta$. If a deal is pareto optimal, then there is no alternative deal that will improve the lot of one agent except at some cost to another agent (who presumably would not be happy about it!). If a deal is not pareto optimal, however, then the agents could improve the lot of at least one agent, without making anyone else worse off.

A deal $\delta$ is said to be *individual rational* if it weakly dominates the conflict deal. If a deal is *not* individual rational, then at least one agent can do better by simply performing the tasks it was originally allocated – hence it will prefer the conflict deal. Formally, deal $\delta$ is individual rational if and only if $\delta \succeq \Theta$.

We are now in a position to define the space of possible proposals that agents can make. The *negotiation set* consists of the set of deals that are (i) individual rational, and (ii) pareto optimal. The intuition behind the first constraint is that there is no purpose in proposing a deal that is less preferable to some agent than the conflict deal (as this agent would prefer conflict); the intuition behind the second condition is that there is no

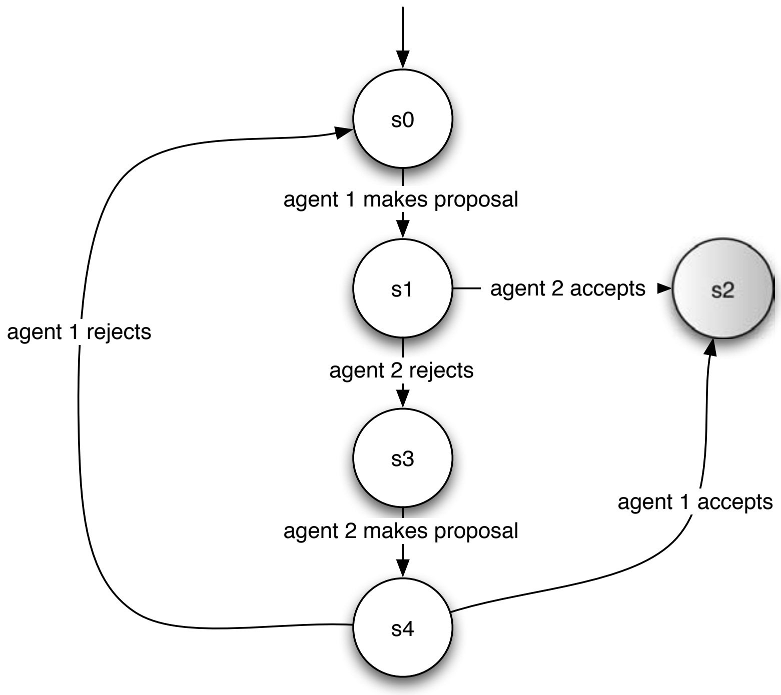

Figure 3.4: The alternating offers protocol (state $s_0$ is the start state, state $s_2$ is the end state).

point in making a proposal if an alternative proposal could make some agent better off at nobody's expense.

So far, we have said nothing about how agents might actually *engage* in negotiation in such scenarios. We will introduce a particular model: the *alternating offers* model of Rubinstein [OR90, Kra01]. We assume just two agents: 1 and 2. Negotiation takes place in a sequence of rounds, which we will assume are indexed by the natural numbers. Agent 1 begins, at round 0, by making a proposal $x^0$ (from the negotiation set), which agent 2 can either accept ($A$) or reject ($R$). If the proposal is accepted, then the deal $x^0$ is implemented. Otherwise, if agent 2 rejected the proposal, then negotiation moves to another round, where agent 2 makes a proposal and agent 1 chooses to either accept or reject it. The overall protocol is illustrated by the state transition diagram in Figure 3.4, where state $s_0$ is the start state and state $s_2$ is a terminal (sink) state.

There is of course nothing to stop negotiation using the alternating offers protocol going on for ever: according to the protocol, they can just keep on $R$ing and $R$ing and $R$ing... If the agents *never* reach agreement, (i.e., a history of the proposal simply consists of a sequence of deals and associated $R$s), then we define the outcome of negotiation to be the *conflict deal*, $\Theta$.

The following basic assumptions are made [OR90, p.33–34][1]:

**Disagreement is the worst outcome:** Both agents prefer any outcome at least as much as disagreement.

**Agents seek to maximise utility:** Agents really do prefer to get larger utility values.

**Time is valuable:** For any outcome $x$ and times $t_1$ and $t_2$, both agents would prefer outcome $x$ at time $t_1$ over outcome $x$ at time $t_2$ if $t_2 > t_1$. Thus, given a choice between agreement $x$ being made now and $x$ being made later, we would always prefer $x$ to be made now.

This basic model captures perhaps the most pertinent aspects of negotiation, but as it stands, it admits some arguably strange and undesirable behaviours. Consider the following situation, for example.

> The two agents are "dividing a pie". That is, there is some resource whose value is "1", which can be divided into two parts, such that (i) the value of the two parts must each be between 0 and 1; and (ii) the values of the parts must sum to 1. Now, suppose that agent 1 uses the following strategy for negotiation: always propose that agent 1 gets the whole pie, and reject any other offer. Now, suppose that agent 1 plays this strategy: what is agent 2's best response? If agent 2 *rejects* the proposal, then the agents will never reach agreement, and so by definition the conflict deal $\Theta$ is enacted. By definition, this would be worse for agent 2 than the deal where agent 1 gets the whole pie, and so agent 2's best response is to *accept* the proposal that agent 1 makes. Moreover, again by assumption, if agent 2 is to accept agent 1's proposal, then it is better to do this *on the first round*, (because it will lose out in any delay). So, agent 2's best strategy in response to agent 1 is to accept agent 1's proposal on the first round. Since getting the whole pie on the first round is obviously agent 1's best outcome, it cannot improve on it at all, and so these two negotiation strategies are in fact in Nash equilibrium with one another [OR90, p.42]!

In fact, it is not hard to see that the alternating offers bargaining scheme admits an *infinite* set of Nash equilibrium outcomes: for any possible deal $x$ in the negotiation set, there is a Nash equilibrium pair of negotiation strategies such that the outcome will be agreement on the deal $x$ in the first time period.

Notice that for agent 2 to understand the situation, it must have access to agent 1's strategy. In human terms, this would mean something like agent 2 being convinced that agent 1 was going to use this strategy. In terms of software agents, however, we can place

[1] Some other technical assumptions are made, but these are the "central" ones.

a different interpretation on this: namely, that agent 2 *has access to agent 1's program*. This is not unrealistic: think of agent 1 is a mobile agent, or Java applet, for example. In such cases, agent 1 can *benefit* by making its code available: it can in effect say "look, here is my strategy" – you have no choice other than to accept my proposal.

If the analysis of alternating offers is deepened somewhat, we can recover a slightly more interesting result – a "fundamental theorem" of alternating offers. Given a modest set of additional assumptions, it is possible to prove that there is a *unique* "subgame perfect equilibrium" in which agent 1 (making the first proposal) makes the proposal that agent 2 would make in the second time period, and this deal is accepted. This actually results in the agent 1 – the "first mover" getting a *bigger* slice of the pie. This may seem paradoxical, but the point is that agent 2 has *no incentive to delay* agreement: if it rejects the proposal made by agent 1 in the first time period, then *both* agents will suffer by having the value of their slice of the pie depreciate because of the delay.

Notice that if the preferences over time are *different*, then the *more patient* agent will benefit in negotiating, because it has increased power. It can "hold out" for longer; it's *threats* will be more credible. For example, suppose that each agent $i$'s time preferences are given by a *discount factor*, $\delta_i \in (0,1)$, so that the utility to agent $i$ of a deal $x$ at time $t$ is $\delta_i^t x$. A *larger* value for $\delta_i$ thus implies *more* patience; a *smaller* value means *less* patience. In this case, the agreement that will be reached in the first time step involves agent 1 getting

$$\frac{1-\delta_2}{1-\delta_1\delta_2}$$

of the pie, and agent 2 getting

$$\frac{\delta_2(1-\delta_1)}{1-\delta_1\delta_2}$$

of the pie. As $\delta_1$ approaches the value 1 (i.e., as agent 1 becomes patient) than its slice of the pie becomes larger and larger.

Despite its simplicity, the alternating offers model is one of the most popular and most influential models of bargaining in multi-agent systems. For example, [Kra01] studies its use in the *data allocation problem*, where one is given a set of *servers* and a set of *datasets*, with utility functions for each server. The servers are each allocated a set of data sets, and provide answers to queries posted by a local client base; answering a query may require only locally held datasets, or datasets held by other queries, in which case the datasets must be purchased. Negotiation in this domain relates to the way in which allocation of datasets to servers: suppose a server has queries relating to a particular dataset, then it may benefit from being allocated this dataset and not having to purchase it from other servers. But, if queries relating to the dataset are popular (think "Britney Spears"), then other servers might also prefer this allocation. Kraus gives a detailed analysis of the different kinds of utility functions that servers might have in such scenarios, and the different ways in which such utility functions can depend upon time. She then gives a brief theoretical analysis of the negotiation problem, and demonstrates that treating the allocation problem as a centralised optimisation problem is not feasible, as the problem is computationally rather complex (it is NP-complete).

Kraus [Kra01] also reviews the use of negotiation in the *resource allocation problem*, where the situation is as follows. The agents are negotiating over the use of some resource that, although it may be renewed infinitely often (in the sense that once one agent has finished using it, another agent can begin), cannot be used by more than one agent at the same time. A deal in such a scenario thus involves producing a schedule, describing who will have access to the resource when. The story is made more interesting by the fact that the agents may have preferences about when they have access to the resource. The structure of the analysis of this scenario is broadly the same as that for data allocation: Kraus gives a detailed discussion of the various properties that utility functions in such scenarios might have, an analytical study of negotiation in such scenarios, and finally, detailed simulation results. She then goes on to study richer *multi-attribute* resource allocation in much the same way.

# Chapter 4

# Communication and Cooperation

By considering the consistency with other chapters, the abstract of this chapter should be here as follows.

In the multiagent systems, communications are indispensable for agents to cooperate each other. In this chapter, the foundations of agent communication are explained. It is also explained how agents can cooperate by using the communications. Most of this chapter is organized and described based on Woodridge's book [Woo01].

## 4.1 Communication

### 4.1.1 Speech Acts

Speech act theory treats communication as action. In the speech act theory, communications are modelled as actions that alter the mental state of communication participants. It is predicated on the assumption that speech actions are performed by agents just like other actions, in the furtherance of their intentions.

The theory of speech acts is generally recognized to have begun with the work of the philosopher John Austin[Aus62]. He noted that a certain class of natural language utterances had the characteristics of actions, in the sense that they change the state of the world in a way analogous to physical actions.

Austin identified a number of performative verbs, which correspond to various different types of speech acts. Examples of such performative are *request*, *inform*, and *promise*. In addition, he distinguished three different aspects of speech acts: *locutionary act* (act of making an utterance), *illocutionary act* (act performed in saying something), *perlocution* (effect of the act)

Austin's work was extended by John Searle[Sea69]. Searle identified several properties that must hold for a speech act performed between a hearer and a speaker to succeed.

- Normal I/O conditions: The conditions state that HEARER is able to hear the request, the act was performed in normal circumstances, etc.

- Preparatory conditions: The conditions state that what must be true of the world in order that SPEAKER correctly choose the speech act. In this case, HEARER must be able to perform ACTION, and SPEAKER must believe that HEARER is able to perform ACTION. Also, it must not be obvious that HEARER will do ACTION anyway.

- Sincerity conditions: The conditions distinguish sincere performances of the request; insincere performance of the act might occur if SPEAKER did not really want ACTION to be performed.

In the late 1960s and early 1970s, a number of researchers in AI began to build systems that could plan how to autonomously achieve goals. Clearly, if such a system is required to interact with humans or other autonomous agents, then plans must include speech actions. This introduced the question of how the properties of speech acts could be represented such that planning systems could reason about them. Cohen and Perrault[CP79] gave an account of the semantics of speech acts by using techniques developed in AI planning research. The aim of their work was to develop a theory speech acts by modelling them in a planning system as operators defined in terms of speakers's and hears'beliefs and goals. Thus, speech acts are treated in the same way as physical actions. The formalism chosen by Cohen and Perrault was the STRIPS notation, in which the properties of an action are characterized via preconditions and postconditions[FN71]. Cohen and Perrault demonstrated how the preconditions and postconditions of speech acts such as *request* could be represented in a multimodal logic containing operators for describing the *beliefs*, *abilities*, and *wants* of the participants in the speech act.

While the plan-based theory of speech acts was a major step forward, it was recognized that a theory of speech acts should be rooted in a more general theory of rational action. This observation led Cohen and Levesque to develop a theory in which speech acts were modelled as actions performed by rational agents in the furtherance of their intentions[CL90b]. The foundation on which they built this model of rational action was their theory of intention[CL90a].

### 4.1.2 Agent Communication Languages

Speech act theories have directly informed and influenced a number of languages that have been developed specifically for agent communication. In the early 1990s, the US-based DARPA-funded Knowledge Sharing Effort (KSE) was formed, with the remit of developing protocols for the exchange of represented knowledge between autonomous information systems.

The KSE generated two main deliverables as follows.

- The *Knowledge Interaction Format* (KIF). KIF is a language explicitly intended to allow the representation of knowledge about some particular 'domain of discourse'. It was intended primarily to form the content parts of KQML messages.

- The *Knowledge Query and Manipulation Language* (KQML). KQML defines an 'envelope' format for messages, using which an agent can explicitly state the intended illocutionary force of a message. KQML is not concerned with the content part of messages[Pat92][MLF96].

### KIF

KIF[GF92] was originally developed with the intent of being a common language for expressing properties of a particular domain. It was not intended to be a language in which messages themselves would be expressed, but rather it was envisaged that the KIF would be used to express message content. KIF is closely based on first-order logic. Thus, for example, by using KIF, it is possible for agents to express properties of things of domain, relationships between things in a domain, and general properties of a domain. In order to express these things, KIF assumes a basic, fixed logical apparatus, which contains the usual connectives that one finds in first-order logic: the binary Boolean connectives *and, or, not*, and so on, and the universal and existential quantifiers *forall* and *exists*. In addition, KIF provides a basic vocabulary of objects such as numbers, characters, and strings. Some standard functions and relations for these objects are also provided. A LISP-like notation is also provided for handling lists of objects. Using this basic apparatus, it is possible to define new objects, and the functional and other relationships between these objects.

### KQML

KQML is a message-based language for agent communication. Thus KQML defines a common format for messages. A KQML message may crudely be thought of as an object: each message has a *performative*, and a number of *parameters*.

The take-up of KQML by the multiagent systems community was significant, and several KQML-based implementations were developed and distributed. Despite this success, KQML was subsequently criticized on a number of grounds as follows.

- The basic KQML performative set was rather fluid. It was never tightly constrained, and so different implementations of KQML were developed that could not interoperate.

- Transport mechanisms for KQML messages (ways of getting a message from agent A to agent B) were never precisely defined, again making it hard for different KQML-talking agents to interoperate.

- The semantics of KQML were never rigorously defined, in such a way that it was possible to tell whether two agents claiming to be talking KQML were in fact using the language 'properly'. The 'meaning' of KQML performatives was only defined using informal, English language descriptions, open to different interpreatations.

- The language was missing an entire class of performatives (commisives) by which one agent makes a commitment to another. As Cohen and Levesque point out, it is difficult to see how many multiagent scenarios could be implemented without commisives, which appear to be important if agents are to coordinate their actions with one another.

- The performative set for KQML was overly large and, it could be argued, rather *ad hoc*.

These criticisms led to the development of a new, but rather closely related language by the FIPA consortium.

**FIPA ACL**

In 1995, the Foundation for Intelligent Physical Agents (FIPA) began its work on developing standards for agent systems. The centerpiece of this initiative was the development of an ACL[FIP99]. This ACL is superficially similar to KQML: it defines an 'outer' language for messages, it defines 20 performatives (such as *inform* and *request*) for defining the intended interpretation of messages, and it does not mandate any specific language for message content. In addition, the concrete syntax for FIPA ACL messages closely resembles that of KQML: the structure of message is the same, and the message attribute fields are also very similar. The most important difference between the two languages is in the collection of performatives they provide.

Given that one of the most frequent and damning criticisms of KQML was the lack of an adequate semantics, it is perhaps not surprising that the developers of the FIPA ACL felt it important to give a comprehensive formal semantics to their language. The approach adopted drew heavily on Cohen and Levesque's theory of speech acts as rational action[CL90b], but in particular on Sadek's enhancements to this work[BS97]. The semantics were given with respect to a formal language called SL. This language allows one to represent *beliefs*, *desires*, and *uncertain beliefs* of agents, as well as the actions that agent perform. The semantics of the FIPA ACL map each ACL message to a formula of SL, which defines a constraint that the sender of the message must satisfy if it is to be considered as conforming to the FIPA ACL standard. FIPA refers to this constraint as the *feasibility* condition. The semantics also map each message to an SL-formula that defines the *rational effect* of the action (the purpose of the message: what an agent will be attempting to achieve in sending the message). However, in a society of autonomous agents, the rational effect of a message cannot be guaranteed. Hence conformance does not require the recipient of a message to represent the rational effect part of the ACL semantics (only the feasibility condition).

Several FIPA implementations have been developed, of which the Java-based Jade system is probably the best known[PR00].

### 4.1.3 Ontologies

If two agents are to communicate about some domain, then it is necessary for them to agree on the terminology that they use to describe this domain. Therein, the issue of ontologies arises. An ontology is a specification of a set of terms as follows: An ontology is a formal definition of a body of knowledge. The most common type of ontology used in building agents involves a structural component. Essentially, a taxonomy of class and subclass relations coupled with definitions of the relationships between these things.

Practical ontology languages are being adopted. For example, the W3C recently recommended the OWL [Hef04] language and RDF for building web ontologies. These language specifications were developed over several years both within and outside of the organization, and OWL is rapidly replacing its predecessor DAML+OIL [CvHH$^+$01] with the blessing of the DAML [dam] Office in the Department of Defense, which funded much of its early development. Commensurate W3C standardization activities are now underway to expand the development framework for building and using web ontologies with web services [owl03, RLK04], deductive rules, and optimized query languages.

Numerous commercial and open-source software tools are available for building and deploying ontologies, and for integrating inference systems with web and database infrastructures. Increasingly, these tools directly support the emerging web ontology standards, as well as related, standard-language efforts like Simple Common Logic (SCL as an offshoot of KIF) and ISO EXPRESS.

Reference to taxonomies and ontologies by vendors of mainstream enterprise-application-integration (EAI) solutions are becoming commonplace. Popularly tagged as semantic integration, vendors like Verity, Modulant, Unicorn, Semagix, and many more are offering platforms to interchange information among mutually heterogeneous resources including legacy databases, semi-structured repositories, industry-standard directories, vocabularies like ebXML, and streams of unstructured content as text and media. Ontologies, for example, are being used to guide the extraction of semantic content from collections of plain-text documents describing medical research, consumer products, and business topics.

## 4.2 Cooperation

Given languages that agents might use to communicate with one another, the important thing is how agents can be designed so that they can work together effectively. The term 'cooperation' is frequently used in the concurrent systems literature, to describe systems that must interact with one another in order to carry out their assigned tasks. Working together involves several different kinds of activities such as the sharing both of tasks and of information

Historically, most work on cooperative distributed problem solving (CDPS) has made the *benevolence* assumption: that the agents in a system implicitly share a common goal, and thus that there is no potential for conflict between them. This assumption implies that agents can be designed so as to help out whenever needed, even if it means that one or more agents must suffer in order to do so.

In contrast to this, the more general area of multiagent systems has focused on the issues associated with societies of *self-interested* agents. Thus agents in a multiagent system cannot be assumed to share a common goals, as they will often be designed by different indivisuals or organization in order to represent their interests. One agent's interests may therefore conflict with those of others, just as in human societies. Despite the potential for conflict of interest, the agents in a multiagent system will ultimately need to cooperate in order to achieve their goals. Multiagent system research is therefore concerned with the wider problems of designing societies of autonomous agents, such as why and how agents cooperate; how agents can recognize and resolve conflicts; how agents can negotiate or compromise in situations where they are apparently at loggerheads; and so on.

The main issues to be addressed in cooperative distributed problem solving include the following.

- How can a problem be divided into smaller tasks for distribution among agents?
- How can a problem solution be effectively synthesized from sub-problem results?
- How can the overall problem-solving activities of the agents be optimized so as to produce a solution that maximizes the coherence metric?
- What techniques can be used to coordinate the activity of the agents, so avoiding destructive interaction, and maximizing effectiveness?

### 4.2.1 Task Sharing and Result Sharing

How do a group of agents work together to solve problems? Smith and Davis[SD80] suggested that the CDPS process can canonically be viewed as a three-stage activity as follows.

- Problems decomposition. In this stage, the overall problem to be solved is decomposed into smaller sub-problems. The decomposition will typically be hierarchical, so that sub-problems are then further decomposed into smaller sub-problems, and so on, until the sub-problems are of an appropriate granularity to be solved by individual agents.

- Sub-problem solution. In this stage, the sub-problems identified during problem decomposition are individually solved. This stage typically involves sharing of information between agents: one agent can help another out if it has information that may be useful to the other

- Solution synthesis. In this stage, solutions to individual sub-problems are integrated into an overall solution. As in problem decomposition, this stage may be hierarchical, with partial solutions assembled at different levels of abstraction

Given this general framework for CDPS, there are two specific cooperative problems-solving activities that are likely to be present: *task sharing* and *result sharing*.

### Task sharing

Task sharing takes place when a problem is decomposed to smaller sub-problem and allocated to different agents. The key problems to be solved in a task-sharing system is that of how tasks are to be allocated to individual agents. In cases where the agents are really autonomous, then task allocation will involve agents reaching agreements with others by using the techniques such as negotiation.

The Contract Net (CNET) protocol is a high-level protocol for achieving efficient cooperation through task sharing in networks of communicating problem solvers[SD80]. The basic metaphor used in the CNET is contracting as follows.

- A node that generates task advertises existence of that task to other nodes in the net with a *task announcement*, then acts as the *manager* of that task for its duration. In the absence of any information about the specific capabilities of the other nodes in the net, the manager is forced to issue a general broadcast to all other nodes.

- Nodes in the net listen to the task announcements and evaluate them. When a task to which a node is suited is found, it submits a *bid*. A bid indicates the capabilities of the bidder that are relevant to the execution of the announced task.

- A manager may receive several bids in response to a single task announcement. Based on the information in the bids, manager selects the most appropriate nodes to execute the task. The selection is communicated to the successful bidders through an award message.

- These selected nodes assume responsibility for execution of the task, and each is called a *contractor* for that task.

The Contract Net has become the most implemented and best-studied framework for distributed problem solving.

### Result Sharing

Result sharing involves agents sharing information relevant to their sub-problems. This information may be shared *proactively* (one agent sends another agent some information because it believes the other will be interested in it), or *reactively* (an agent sends another information in response to a request that was previously sent).

In result sharing, problem solving proceed by agents cooperatively exchanging information as a solution is developed. Typically, these results will progress from being the solution to small problems, which are progressively refined into larger, more abstract solutions. Durfee[Dur99] suggests that problem solvers can improve group performance in result sharing in the following ways.

- Confidence: independently derived solutions can be cross-checked, highlighting possible errors, and increasing confidence in the overall solution.

- Completeness: agents can share their local views to achieve a better overall global view.

- Precision: agents can share results to ensure that the precision of the overall solution is increased.

- Timeliness: even if one agent could solve a problem on its own, by sharing a solution, the result could be derived more quickly.

One of the major problems that arises in cooperative activity is that of inconsistencies between different agents in the system. Agents may have inconsistencies with respect to both their *beliefs* and their *goals*, *intentions*. Inconsistencies between goals generally arise because agents are assumed to be autonomous, and thus not share common objectives. Inconsistencies between the beliefs that agents have can arise from several sources. First, the viewpoint that agents have will typically be limited. No agent will ever be able to obtain a complete picture of their environment. Also, the sensors that agents have may be faulty, or the information sources that the agents has access to may in turn be faulty.

In a system of moderate size, inconsistencies are inevitable: the question is how to deal with them. Durfee *et al.*[DLC89] suggest a number of possible approaches to the problem as follows.

- Do not allow it to occur or at least ignore it. This is essentially the approach of the Contract Net: task sharing is always driven by a manager agent, who has the only view of the problem that matters.
- Resolve inconsistencies through negotiation. While this may be desirable in theory, the communication and computational overheads incurred suggest that it will rarely be possible in practice
- Build systems that degrade gracefully in the presence of inconsistency.

### 4.2.2 Coordination

The defining problem in cooperative working is that of *coordination*. The coordination problem is that of *managing inter-dependencies between the activities of agents*: some coordination mechanism is essential if the activities that agents can engage in can interact in any way. von Martial suggested a typology for coordination relationships[vM90]. He suggested that, broadly, relationships between activities could be either positive or negative.

Positive relationships are all those relationships between two plans from which some benefit can be derived, for one or both of the agents plans, by combining them. Such relationships may be *requested* (I explicitly ask you for help with my activities) or *non-requested* (it so happens that by working together we can achieve a solution that is better for at least one of us, without making the other any worse off).

Coordination in multiagent systems is assumed to happen at run time, that is, the agents themselves must be capable of recognizing these relationships and where necessary, managing them as part of their activities. Some approaches have been developed for dynamically coordinating activities.

#### Partial global planning

The main principle of partial global planning[DL87] is that cooperating agents exchange information in order to reach common conclusions about the problem solving process. Planning is partial because the system does not (indeed cannot) generate a plan for the entire problem. It is *global* because agents form non-local plans by exchanging local plans and cooperating to achieve a non-local view of problem solving.

Partial global planning involves three iterated stages.

- Each agent decides what its own goals are, and generates short-term plans in order to achieve them.
- Agents exchange information to determine where plans and goals interact.
- Agents alter local plans in order to better coordinate their own activities.

### Joing intentions

The second approach to coordination is the use of *human teamwork models*. Therein, intentions play a critical role in coordination: they provide both the stability and predictability that is necessary for social interaction, and the flexibility and reactivity that is necessary to cope with a changing environment. When humans work together as a team, mental states that are closely related to intentions appear to play a similarily important role. It is also important to be able to distinguish coordinated action that is not cooperative from coordinated cooperative actions. Being part of a team implies some sort of responsibility towards the other members of the team.

### Mutual modelling

Another approach to coordination, closely related to the models of human teamwork is that of coordination by *mutual modelling*. The idea is to put ourselves in the place of the other: to build a model of other agents - their beliefs, intentions, and the like - and to coordinate our activities around the predictions that this model makes. This approach to coordinate was first explicitly articulated in Genesereth *et al.*[GGR83] , where the approach was dubbed 'cooperation without communication'. The models that were proposed were essentially the game-theoretic models. The idea was that if you assume that both you and the other agents with which you interact share a common view of the scenario (in game-theory terms, you all know the payoff matrix), then you can do a game-theoretic analysis to determine what is the rational thing for each player to do.

### Norms and social laws

In our everyday lives, we use a range of techniques for coordinating activities. One of the most important is the use of *norms* and *social laws*[Lew69]. A norm is simply an established, expected pattern of behaviour (e.g. forming a queue when waiting for a bus, and allowing those who arrived first to enter the bus first); the term social law carries essentially the same meaning, but it is usually implied that social laws carry with them some authority. Although the norm is not enforced in any way, it provides a template that can be used by all those around to regulate their own behaviour.

# Chapter 5

# Standardization

In the multiagent systems, standardization is indispensable for agents to cooperate each other. In this chapter, an overview of the FIPA standardization effort is given, providing an introduction to the current set of FIPA specifications. Most of this chapter is organized and described based on FIPA document 00024 [BDW01].

## 5.1 FIPA: Open Standards for Software Agents

### 5.1.1 Overview

The Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 as a non-profit organisation with the remit of producing software standards for heterogeneous and interacting agents and agent-based systems across multiple vendors' platforms. This is expressed more formally in FIPA's official mission statement:

*The promotion of technologies and interoperability specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings.*

The emphasis here is therefore on the practical commercial and industrial uses of agent systems. The aim is to bring together the latest advances in agent research with industry best practice in software, networks and business systems.

FIPA undertakes its work at meetings that are held four times a year and conducts its standardisation process in a collaborative and open manner as specifications are publicly accessible during their life time and participation to meetings is free of charge.

### 5.1.2 Structure of FIPA

FIPA is organised and structured according to two groups:

#### Administrative Groups

- The FIPA *Board of Directors* (BoD) is responsible for managing and conducting the business of the FIPA organisation.
- The FIPA *Architecture Board* (FAB) is the authority within FIPA that is responsible for ensuring the consistency, accuracy and suitability of FIPAs technical work.
- The FIPA secretariat, in charge of administration, logistics, membership and information dissemination of FIPA.
- The Image committee is building some communication channels for FIPA and presentations of the standard inside and outside the Agent Community.

#### Technical Groups

FIPA's core standardisation activities are centred around the creation and maintenance of specifications.

- Technical Committees (TCs) produce technical work and write the FIPA specifications. The life cycle of a TC starts with a work plan submitted to the FAB. If approved, the FAB proposes to create a TC, the BoD takes the decision and the TC is created with the mission to fulfil the work plan.
- it Working Groups (WGs) are designed to carry out other aspects of FIPA's work, which are not necessarily defined by technology; they may have an application focus or be responsible for coordinating implementation activities. The lifecycle of a WG is similar to the one of a TC.
- *Special Interest Groups* (SIGs) undertake auxiliary work which is of interest to sections of FIPA membership, such as liasing with other standards bodies and dealing with emerging technologies which might be suitable for standardisation.

### 5.1.3 Developments in FIPA

The year 2000 - 2001 there was intense activity within FIPA and great changes in the technological landscape which forms the backdrop for its activities. This resulted in both technical advances and changes to FIPAs structure and specification process. The main procedural changes can be summarised as follows:

- A new, more open specification process based on a permanently open call for work plans from both members and non-members (see http://www.fipa.org/about/fab.html).

- A multi-track standardisation process decoupling standards to enable work in different areas to progress at different speeds.
- A multi-step document process reflecting the different levels of maturity of the specifications through successive grades of maturity: *Preliminary*, *Experimental*, *Standard* through *Deprecated* and *Obsolete*.

There was also significant progress in technical work:

- Development of a global Framework to link together the totality of FIPA specifications into a single Abstract Architecture.
- Integration of Web technologies into the standards including: HTTP message transport, XML encodings for FIPA ACL messages and an RDF content language.
- Modularisation of all specifications to create a plug and play architecture supporting interchangeable *component* specifications for areas such as content languages, performatives, protocols, message transport and language syntax complemented by *profiles* to link together sets of components.

## 5.2 FIPA Specifications

Since January 2000, FIPA adopted a new procedure for classifying, organising and releasing specifications to ensure coherence, completeness and consistency of its work as well as its relevance to industrial and commercial interests. This section provides an overview of the new specification structure and the current set of FIPA specifications.

### 5.2.1 Specification Structure

FIPA specifications are divided into five categories: Applications, Abstract Architecture, Agent Communication, Agent Management and Agent Message Transport. Each area of specifications has one or more specification documents assigned to it and involved one or more technical committees or working groups.

### 5.2.2 Current Generation of FIPA Specifications

#### Abstract Architecture

The purpose of the FIPA Abstract Architecture (see [FIPA00001]) is to foster interoperability and reusability, this leads to the identification of architectural abstractions linked by their relationships. It makes a distinction between those elements which can easily be defined in an abstract manner, such as agent message transport, FIPA ACL, directory services and content languages, and between those elements that cannot, such as agent management and agent mobility. These are considered difficult to represent abstractly since they occur too close to the concrete realisation (implementation) of an agent system and very little commonality can be derived from analysing them. Yet, these issues will have to be addresses by developers and the abstract architecture will provide a number of instantiation guidelines in the future for specific groupings of implementation technologies.

The first concrete realization of the abstract architecture will be the Java Agent Service project which is being developed as part of the Java Community Process.

### Agent Message Transport

The FIPA Agent Message Transport Specifications deal with the delivery and representation of messages across different network transport protocols, including wireline and wireless environments.

At the message transport level, a message consists of a message envelope and a message body. The envelope contains specific transport requirements and information that is used by the Message Transport Service (MTS) on each agent platform to route and handle messages. The message body is the payload and is usually expressed in FIPA ACL but is

The agent message transport reference model provides facilities for (see*5.1*):

- General support for an MTS within an agent platform (see [FIPA00067]).
- Guidelines for using specific Message Transport Protocols (MTPs), such as IIOP (see [FIPA00075]), HTTP (see [FIPA00084]) and WAP (see [FIPA00076]).
- Message envelope representations that are suitable for each MTP, such as an XML encoding for HTTP (see [FIPA00085]) and a bit-efficient encoding for WAP (see [FIPA00088]).
- FIPA ACL representations, such as a string encoding (see [FIPA00070]), an XML encoding (see [FIPA00071]) and a bit-efficient encoding (see [FIPA00069]).

The MTS on each agent platform can support any number of message transport protocols and will normally translate between a FIPA-supported MTP that is used for interoperable communication between heterogeneous agent platforms (such as XML over HTTP) and an MTP that is used internally to the agent platform (such as Java objects over the Java Messaging Service).

Consequently, the components of the MTS are designed to be modular and extensible to handle different message transport protocols, message envelope and FIPA ACL representations in the future.

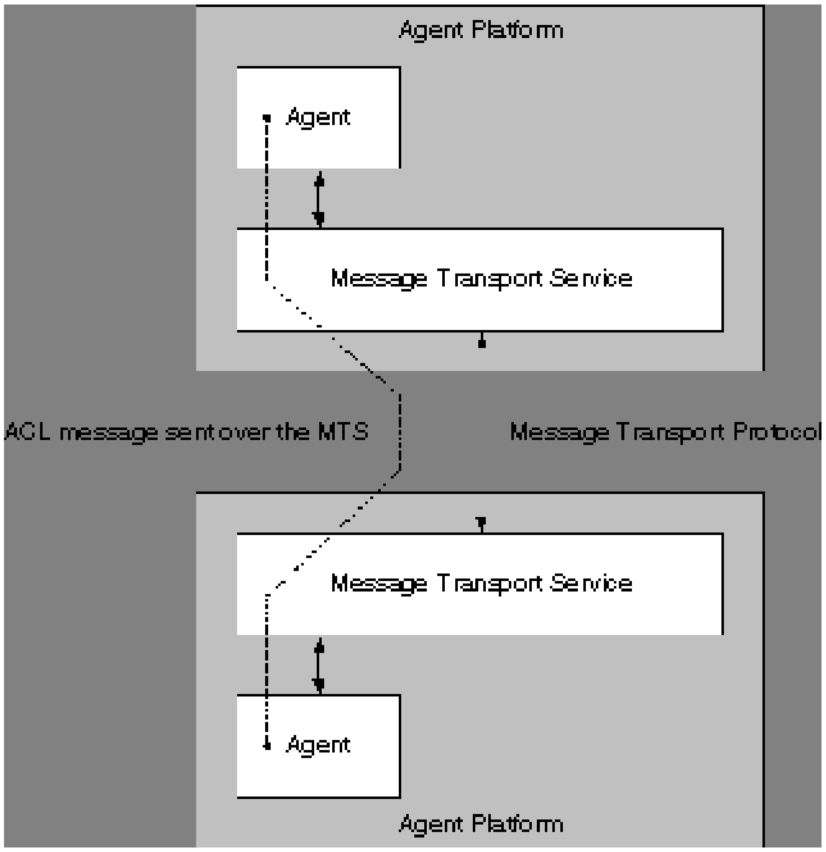
Agent Platform
Agent
Message Transport Service
ACL message sent over the MTS
Message Transport Protocol
Message Transport Service
Agent
Agent Platform

Figure 5.1: Agent Message Transport Reference Model

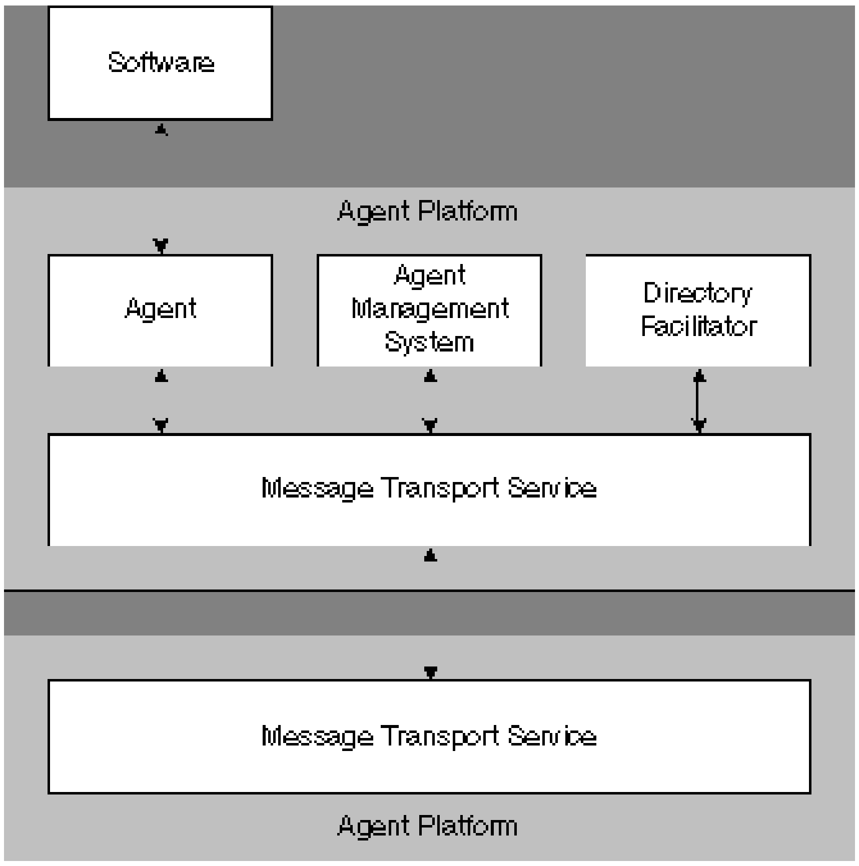

Figure 5.2: A
gent Management Reference Model

### 5.2.3 Agent Management

The FIPA Agent Management Specification (see [FIPA00023]) provides the framework within which FIPA agents exist and operate. It establishes the logical reference model for the creation, registration, location, communication, migration and retirement of agents.

The reference model (see*5.2*) describes the primitives and ontologies necessary to support the following services in an agent platform:

- White pages, such as agent location, naming and control access services, which are provided by the Agent Management System (AMS). Agent names are represented by a flexible and extensible structure called an agent identifier, which can support social names, transport addresses, name resolution services, amongst other things.
- Yellow pages, such as service location and registration services, which are provided by the Directory Facilitator (DF).
- Agent message transport services as described previously in section 3.2.2.

In conjunction with the FIPA Agent Message Transport Specifications, the FIPA Agent Management Specification also provides support for intermittently connected devices, such as laptop computers and personal digital assistants through message buffering, redirection and proxying.

### Agent Communication

Developers of multi-agent systems require specialised communication techniques in order to structure the interactions in their agent systems. Ad hoc techniques are usually not sufficiently well designed or documented to be consistently extensible and implementable by others, or generally applicable to a wide set of agent problems. The FIPA specifications for agent communication address these issues. The core of these specifications was largely completed in FIPA 97, but this specification set has required continual maintenance and development since then. The specifications of the communication language, along with libraries of predefined communicative act types, interaction protocols and content languages were developed:

- FIPA ACL Communicative Act Specifications is the library of all the 22 FIPA communicative acts and their requirements (see [FIPA00037])
- FIPA ACL Message Structure Specification describes the grammatical structure of the FIPA ACL (see [FIPA00061])
- FIPA Interaction Protocol Library Specification is the library of FIPA interaction protocols and requirements for new interaction protocols (see [FIPA00025]). Currently existing FIPA interactions protocols are:
  - FIPA Request Interaction Protocol Specification (see [FIPA00026])
  - FIPA Query Interaction Protocol Specification (see [FIPA00027])
  - FIPA Request When Interaction Protocol Specification (see [FIPA00028])
  - FIPA Contract Net Interaction Protocol Specification (see [FIPA00029])
  - FIPA Iterated Contract Net Interaction Protocol Specification (see [FIPA00030])
  - FIPA English Auction Interaction Protocol Specification (see [FIPA00031])
  - FIPA Dutch Auction Interaction Protocol Specification (see [FIPA00032])
  - FIPA Brokering Interaction Protocol Specification (see [FIPA00033])
  - FIPA Recruiting Interaction Protocol Specification (see [FIPA00034])
  - FIPA Subscribe Interaction Protocol Specification (see [FIPA00035])
  - FIPA Propose Interaction Protocol Specification(see [FIPA00036])

- FIPA Content Language Library is a generic description of the requirements for a FIPA content languages. (see [FIPA00007]). The following content languages have been specified by FIPA:
  - FIPA SL Content Language Specification (see [FIPA00008])
  - FIPA CCL Content Language Specification (see [FIPA00009])
  - FIPA KIF Content Language Specification (see [FIPA00010])
  - FIPA RDF Content Language Specification(see [FIPA00011])

**AgentApplications**

FIPA has developed specifications of four agent-based applications that contain service and ontology descriptions and case scenarios:

- *Personal Travel Assistance:* individualised, automated access to travel services (see [FIPA00080]).
- *Audio-Visual Entertainment and Broadcasting:* negotiating, filtering, and retrieving audio-visual information, in particular for digital broadcasting networks (see [FIPA00081]).
- *Network Management and Provisioning:* automated provisioning ofdynamic Virtual Private Network services where a user wants to set up a multi-media connection with several other users (see [FIPA00082]).
- *Personal Assistant:* management of a user's personal meeting schedule, in particular in determining time and place arrangements for meetings with several participants (see [FIPA00083]).

Additionally, the *Agent Software Integration* specification (see [FIPA00079]) contains guidelines for integrating legacy software, that is, software that does not communicate using FIPA ACL.

## 5.3 Conclusions

FIPA has devoted its first years to specify the basics elements of an agent-based world, defining the Communication language, the management and the connection to existing software. This initial specification has been implemented by several teams and showed the way for further improvement towards a more modular agent environment capable of evolution and integration of new technologies, e.g. using several Message Transport Layers, and moving towards higher levels of abstraction. Four of these new implementations are already accessible in open source, and new FIPA implementations enable now agents to run on small wireless devices such as PDA. As such FIPA is making an important contribution to the practical and commercial viability of agent systems by providing a good basis to develop future agent-based applications.

FIPA is now looking ahead to some of the major challenges that are already beginning to arise in the new networked world, and works on specifications for:

- Domains, policies, agreements and contracts, security, dependencies between large numbers of agents, and agent usage of ontology.
- Key applications such as product manufacturing and design and peer-to-peer systems.
- Large-scale deployment of agent systems in open environments.

# Chapter 6

# Agent Systems

Once standardization would be achieved, implemented tools are needed for the practical commercial and industrial use of heterogeneous and interacting agents and agent-based systems. As described in [vEHKB02], many agent platforms are available, both commercial and academic. All agent platforms try to provide support for mobile agent technology, i.e. to support migration, naming, location and communication services. These systems differ widely in architecture and implementation, thereby impeding interoperability and rapid deployment of mobile agent technology in the marketplace.

To promote interoperability, some aspects of mobile agent technology need to be standardised.

## 6.1 JADE and LEAP platforms

Because it is well-developed, we describe the JADE/LEAP platform in more detail.

JADE 3 is an agent platform developed under a grant from the European Commission (IST-1999- 10211). It is software framework implemented in the Java programming language. Agents are defined within the scope of a platform. The agent platform can be distributed across machines (which do not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. A platform is composed of a number of agent containers, which is an agent s home environment in a platform, there is one and only one main container and any number of normal containers. The main container can also host normal agents, but in any case, it contains a number of platform agents that perform special functions on the platform such as the Agent Management System (AMS), the Agent Communication Channel (ACC) and the Directory Facilitator (DF), which is a white and yellow page directory where agents can request each other s address and where they can check whether agents exists anywhere that perform specific functions. In addition, JADE implements support for FIPA s Agent

Communication Language (ACL).

LEAP is the result of a different EU project of a consortium consisting of Motorola, ADAC, BT, Broadcom Eireann Research, Telecom Italia Lab, University of Parma and Siemens, which has been integrated with JADE. LEAP (Bergenti & Poggi, 2001) stands for Leightweight and Extensible Agent Platform, it has a small footprint that makes it possible to run an agent container on any Java enabled (mobile) device. In this way, an agent platform can be extended to include such mobile devices, which means the advantages of agentbased technology can be extended to those devices through wireless communication.

## 6.2 Publicly available platforms

Following is a list of major publicly available implementations of agent platforms which conform to the FIPA Specifications published in the FIPA web site.

### 6.2.1 Agent Development Kit

| | |
|---|---|
| Name | Agent Development Kit |
| Authors | Tryllian BV |
| Description | Tryllian introduces the latest release of the Agent Development Kit (ADK), a mobile component-based development platform that allows you to build reliable and scalable industrial strength applications.<br>The ADK features dynamic tasking, JXTA-based P2P architecture with XML message-based communication that supports FIPA and SOAP, JNDI directory services, using a reliable, lightweight runtime environment based on Java. These allow Java Developers to easily build, deploy and manage secure, large-scale distributed solutions that operate regardless of location, environment or protocol, enabling an adaptive, dynamic response to changes. |
| License | Commercial license. Free research license available for selected projects. |
| Requirements | Environment, minimal configuration. |
| Contact | http://www.tryllian.com/ |

### 6.2.2 April Agent Platform

| | |
|---|---|
| Name | April Agent Platform |
| Authors | Jonathan Dale (jonathan.dale@fla.fujitsu.com) and Johnny Knottenbelt (jak97@doc.ic.ac.uk) |
| Description | The April Agent Platform (AAP) is a FIPA-compliant agent platform that is designed to be a lightweight and powerful solution for developing agent-based systems. It is written using the April programming language and the InterAgent Communication System (IMC), and provides many features to accelerate the development and deployment of agents and agent platforms. |
| License | GPL |
| Requirements | The AAP requires the April programming language and the ICM to be installed, and runs either on Linux, Unix or Windows. |
| Contact | http://sf.us.agentcities.net/aap/ |

### 6.2.3 Comtec Agent Platform

| | |
|---|---|
| Name | Comtec Agent Platform |
| Authors | Information-Technology Promotion Agency, Japan and Communication Technologies |
| Description | Comtec Agent Platform is an open-source, free implementation of FIPA agent communication, agent management, agent message transport and some of the applications. Unique to the Comtec Platform is the implementation of FIPA Ontology Service and Agent/Software Integration, which require SL2 as the content language. |
| License | GPL |
| Requirements | JDK 1.2 or higher. |
| Contact | http://ias.comtec.co.jp/ap/ |

### 6.2.4 FIPA-OS

| | |
|---|---|
| Name | FIPA-OS |
| Authors | Emorphia and contributors |
| Description | FIPA-OS was the first Open Source implementation of the FIPA standard and has now recorded thousands of downloads. Dedicated developers from around the world have contributed to numerous bug fixes and upgrades, leading to over 10 formal new releases. FIPA-OS now supports most of the FIPA experimental specifications currently under development. With the new in depth developers guides, it is an ideal starting point for any agent developer wishing to benefit from FIPA technology. FIPA-OS 2 is a component-based toolkit implemented in 100% pure Java. One of the most significant contributions received is a small-footprint version of FIPA-OS (FIPA-OS), aimed at PDAs and smart mobile phones, which has been developed by the University of Helsinki as part of the IST project Crumpet. |
| License | The license is EPL, details of which can be found at http://www.emorphia.com/EPL/ |
| Requirements | Java virtual machine |
| Contact | http://fipa-os.sourceforge.net/ |

### 6.2.5 Grasshopper

| | |
|---|---|
| Name | Grasshopper |
| Description | Grasshopper is an open 100% Java-based mobile intelligent agent platform, which is compliant to both available international agent standards, namely the OMG MASIF and FIPA specifications. Grasshopper includes two optional open source extensions providing the OMG MASIF and FIPA standad interfaces for agent/platform interoperability. |
| Requirements | Java virtual machine |
| Contact | http://www.grasshopper.de/ |

### 6.2.6 JACK

| | |
|---|---|
| Name | JACK Intelligent Agents |
| Description | JACK is an environment for building, running and integrating commercial-grade multi-agent systems using a component-based approach. JACK is based upon the company's Research and Development work on software agent technologies. The JACK Agent Language is a programming language that extends Java with agent-oriented concepts, such as: agents, capabilities, events, plans, agent knowledge bases (Databases) and resource and concurrency management. |
| Contact | http://www.agent-software.com/ |

### 6.2.7 JADE

| | |
|---|---|
| Name | JADE |
| Authors | TILAB (fromerly CSELT) |
| Description | JADE simplifies the development of multi-agent applications, which comply with the latest FIPA 2000 specifications. While appearing as a single entity to the outside world, a JADE agent platform can be distributed over several hosts. Agents can also migrate or clone themselves to other hosts of the platform, regardless of the OS. The life cycle of agents can be remotely controlled via a GUI, which also allows debugging tools to be started. The communication architecture tries to offer (agent transparent) flexible and efficient messaging by choosing, on an as needed basis, the best of the FIPA-compliant Message Transport Protocols (MTP) that are activated at platform run time. JADE is implemented in version 1.2 of JAVA and has no further dependency on third-party software. |
| License | LGPL |
| Requirements | Java Virtual Machine (1.2 minimum) |
| Contact | http://jade.cselt.it/ |

### 6.2.8 JAS

| | |
|---|---|
| Name | Java Agent Services API (JAS) |
| Authors | Fujitsu, Sun, IBM, HP, Spawar, InterX, Institute of Human and Machine Cogtnition, Comtec, Verizon. |
| Description | The Java Agent Services (JAS) project defines an industry standard specification and API for the deployment of agent platform-service infrastructures. It is an implementation of the FIPA Abstract Architecture within the Java Community Process [http://www.jcp.org] initiative and is intended to form the basis for creating commercial grade applications based on FIPA specifications. Specifically, the project consists of a Java API (in the javax.agent namespace) for deploying open platform architectures that support the plug-in of third-party platform service technology. The API provides interfaces for message creation, message encoding, message transport, directory and naming. This design is intended to ensure that a JAS based system deployment remain transparent to shifts in underlying technology without causing interruption to service delivery and therefore the business process. The project also delivers a Reference Implementation of the API, including sample services for RMI, LDAP and HTTP. The forthcoming Technology Compatibility Kit will ensure compliance of all JAS based implementations. |
| License | Open Specification License v0.4 [http://www.java-agent.org/Documents/OSLFLAvo.4.htm]<br>Common Public License v0.5 [http://www.opensource.org/licenses/cpl.html] |
| Requirements | Java Virtual Machine (1.1 minimum) |
| Contact | http://www.java-agent.org/ |

### 6.2.9 LEAP

| | |
|---|---|
| Name | LEAP |
| Description | LEAP (Lightweight Extensible Agent Platform (IST-1999-10211)) is a development and run-time environment for Intelligent Agents, is the precursor of the second generation of FIPA compliant platforms. It represents a major technical challenge - it aims to become the first integrated agent development environment capable of generating agent applications in the ZEUS environment and executing them on run-time environments derived from JADE, implemented over a large family of devices (computers, PDA and mobile phones) and communication mechanisms (TCP/IP, WAP). In this way LEAP benefits from the advanced design-time features of Zeus and the lightweight and extensible properties of JADE. |
| Requirements | Java Virtual Machine |
| Contact | http://leap.crm-paris.com/ |

### 6.2.10 ZEUS

| Name | ZEUS |
|---|---|
| Description | ZEUS is an Open Source agent system entirely implemented in Java, developed by BT Labs and can be considered a toolkit for constructing collaborative multi-agent applications. Zeus provides support for generic agent functionality and has sophisticated support for the planning and scheduling of an agent's actins. Moreover, Zeus provides facilities for supporting agent communications using FIPA ACL as the message transport and TCP/IP sockets as the delivery mechanism. Zeus also provides facilities for building agents in a visual environment and support for redirecting agent behavior. The Zeus approach to planning and scheduling involves representing goals and actions using descriptions that include the resources they require and the pre-conditions they need to be met in order to function. This allows goals to be represented using a chain of actions that have to b fulfilled before the goal can be met. This action chain is built up using a process of backwards chaining. |
| Requirements | As ZEUS uses the latest Swing GUI components it will run on any platform that has had a JDK1.2 (aka JDK2) virtual machine installed. Each host machine should also be capable of TCP/IP communication, but there is no need for any middleware services to be installed. So far ZEUS has been successfully tested on Windows 95/98/NT4 and Solaris platforms. |
| Contact | http://www.labs.bt.com/project/agents.htm |

# Chapter 7

# Agentcities

In this chapter, an evaluation of the topics discussed in this deliverable is described by means of the Agentcities project. Considering the results and achievements of the project, the remaining problems and future works are also discussed.

## 7.1 Introduction

Aiming at the development of an open, worldwide interoperable agent network, Agentcites projects[age] includes two projects: Agentcites.RTD[rtd] and Agentcities.NET[net].

Agentcities.RTD was a European Commission funded 5th Framework IST Research project which ran from July 2001 to June 2003. The project deployed a 24/7 testbed backbone of 14 live agent platforms representing: 12 European cities (Barcelona, Berlin, Chambery, Dublin, Ipswich, Lausanne, Lisbon, London (2 platforms), Paris, Parma, Saarbruecken, Turin), 1 Japanese city (Sendai), 1 American city (San Francisco). All agent platforms in the backbone and the prototype agent services running on them has been publicly accessible. The Agentcities.NET project is a taken-up measure project engaging in a series of concerted actions to make the Agentcities network accessible to the research and business community.

## 7.2 Objectives

The Agentcities project[DWB02][rep03] aimed to develop a foundation for the vision of an ambient proactive environment where heterogeneous, autonomous and increasingly intelligent systems representing businesses, services and individuals are able to interact with each other and enable flexible and dynamic composition of services.

More formally, the objectives were as follows[rep03].

1. **Open Network Architecture**: Achieve advances in technology / understanding of open systems supporting many heterogeneous, autonomous interacting entities to produce sustainable network architecture for on-line open systems.

2. **Dynamic Value Creation**: Achieve greater understanding of the type of mechanisms, methodologies and techniques required to achieve seamless, effective service composition in an open dynamic environment.

3. **Service Level Interoperability**: Perform validation and refinement of agent communication technologies (such as semantic models, ontology models, content expression techniques and interaction protocols) for use in open dynamic environments.

These project objectives were translated into a primary set of operational goals[rep03]:

1. To implement a realistic, decentralised, open distributed system enabling high-level peer-to-peer interoperability between agents on multiple platforms across Europe and the World.

2. To develop a rich trading environment through agent-based business services, enabling business transactions between agents in the system to support the dynamic composition of services in a JIT (just-in-time) manner.

3. To create agent-based applications by deploying a large number of agents offering diverse services which can turn platforms into Agentcities by offering leisure, entertainment and community services for European and World cities.

4. To ensure that the network, platforms and services are open to and accessible by project external users - thereby serving as a continuously running demonstrator and resource for others.

## 7.3 Methodologies

The nodes in the Agentcities Network are agent platforms that are running on one or more machines and are hosted by an organization or individual. Agents running on a particular Agentcity are able to connect to other publicly available cites and can communicate directly with their agents. Applications involving agents on multiple different cities can be created through the flexible use of inter-agent communication models and semantic frameworks, shared ontologies, content languages and interaction protocols that support it. This model consists of the following levels:

- **Network level**: Platforms in the Agentcities Network interoperate and exchange basic communications at the communication and infrastructure level.

- **Service composition level**: The services provide an open test bed where anybody can observe services at work, run services, and/or add new services into this network. This level should be able to host business components including their service and behaviour descriptions.
- **Semantic interoperability level**: In the longer term, Agentcities become a test bed for system-system communication in an open environment that is capable to dynamically host business components without requiring human intervention to set perform service discovery and invocation.

To realize the above models, the Agentcities Network might be implemented in the multiagent systems. The multiagent systems are distributed systems, with each agent able to act autonomously and able to reason and make decisions about the environment in which it is embedded. The distributed and autonomous nature of mulitagent systems potentially supports flexibility and robustness in system operation and organization, but it is precisely these features that make it difficult to design, build, test, integrate and even use computer systems composed of communicating agents. FIPA created one of the first steps to developing a standard environment to lead the way to autonomous entities that can semantically interoperate. Agentcities.RTD project used these initial developments and results, such as, specifications, agent platforms, and initial interoperable results as the first step towards creating a large scale test-bed of a network of agent platforms.

In the Agentcities Network, the coordination feature of the multiagent systems may be distributed over four abstractions:

- **Ontologies**: A detailed approach of designing and building collaborative ontologies is used. These ontologies and their concepts are shared between agents to enable delivery of coherent services.
- **Actions**: An agent providing a service or a task provides access to these services and tasks through a set of actions (e.g. make-cinema-booking). These sets of actions can also be modelled as an ontology, and a published as part of the service description.
- **Service description**: Each service is described using the FIPA service description and registered in a FIPA-DF so that agents can find the service they need for a particular task.
- **Communication language**: All of these abstractions are captured in a structured way in the FIPA-ACL, this allows the interoperability of services and the exchange of information to occur. This also enables incremental testing through each service provider publishing sample messages both as a receiver and sender.

The actual reasoning about these aspects of a communication, which is communicated from an agent, needs to be internalised so the reasoning behaviour of an agent can deal with the intended meaning of the content. The receiving agent must take into account the protocol and its own belief model and current behaviour status of a set of current interactions.

The fundamentally difference between an agent approach and other engineering approaches is that when a message is dispatched from an agent, it is not object or method dependent but is a rich context of information that can be read (not necessarily understood) by another computational entity. A developer at the ACL level focuses on understanding a computational message protocol context. So part of the trade-off is whether development is more effective at the detailed API level or at interpreting the protocol context. We can assist in developing tools such as parsers and dialogue managers for the protocol context.

There are tools and parsers available to allow engineers to develop distributed software service systems through an agent approach. Developers can treat the communication language as a software interface. The interactions, dialogues or protocols are a set of calls to a particular program. Often an internal model of an agent interprets the input from a message, using both its current beliefs and the message to create a context in which to determine what to do next. Agent developers, like with most software development, will have a knowledge of both the expected input (communications received) and the expected output (communications sent). Protocols, communicative acts and content definitions are driven by the application requirements. Certain degree of separation can be achieved (e.g. through the use of FIPA protocols to obtain a level of architectural openness).

These multiagent architectures on the Agentcities Network are implemented based on following principles.

- Autonomous and communication principals of agent technology for developing and demonstrating how to create and benefit from semantic interoperability.
- FIPA-based platforms and specifications: Using state of the art distributed software engineering principals and deployment, for loosely coupled autonomous software entities and semantic communication theories for enabling high-level semantic interoperability.
- Fundamental bases of the design was the use of “ontological” engineering approach to the utilization of domain concepts, knowledge, and content.
- Use of current developments and advances in the Semantic Web and W3C, such as, the standards developing for defining ontologies in an open Web-based manner, taking into account where possible the issues that impact “open semantic” based environments.
- State of the art research and development in enabling dynamic composition of services define through autonomous agent entities.
- Development tools and advancement of tools to assist organizations to join the network with platforms, test the network, in particular web interface for viewing the

activity of the agent platform network and testing the basic interoperable position of a particular platform through use of the test suite.

- Experimentation of network via deployment of set of services and tested work flow cases.
- Openness of results, following an open source policy where possible to enable broader set of developer and users to participate in research test-bed. All results and developments published for use where possible.

## 7.4 Project Results and Achievements

The Agentcities project produced the following major results:

- A global scale, live and open test environment for development and deployment of service composition technologies.
- Technical frameworks, methods and software illustrating how Agent, Semantic Web and Web Services technologies can be combined to support the construction of dynamic application across open networks of automated software components.
- A large-scale demonstration application combining over 25 different service types and nearly 200 individual service components deployed throughout Europe, the USA and Japan.

More precisely, the achievements of the project are described as follows.

**Agentcities Network**

Since the launch of the project, the network has grown explosively to over 150 registered platforms and the network involves well over 100 organisations worldwide contributing to the construction of the testbed, developing applications and gathering experiences from it.

The network and all services are completely open to enable third parties to access services, add their own and extend the functionality and richness of the environment. This not only makes the result a public resource but also forces Agentcities to address fundamental issues related to open system and provides a vital feedback channel.

The Agentcities testbed network is now being actively used by a wide range of organisations and this usage is supported by a range of network services which enable: service advertisement/discovery, identity management, communication as well as basic management. These systems will now be maintained under the auspices of a new non-profit organisation to enable collaboration between project users. Specifically they will also receive support from the following new projects: Agentcities.ES (Spain), Agentcities.UK (UK), @lis technology net (Europe and Latin America) which will keep the servers and systems available. This will keep the testbed usable for continued research and development.

The current services are of a prototype nature but well tested. With new project activity the services will be regularly upgraded to keep pace with new needs. All technical documents, recommendations and interfaces associated with the network services will be free and in the public domain.

### Agent Platforms and Standards

In respect to the agent platforms, the deployment of the network and combination of technologies lead to:

- The development of new FIPA compliant Agent platforms and the adaptation of existing non-compliant platforms - specifically at the beginning of the project there were 3-4 platforms which had been shown to interoperate effectively (in a FIPA ebake-offf interoperability test) and by the end of the project nearly 20 compliant toolkits are available. Much of this growth in interest is attributable to the value of connecting to the Agentcities network.

- The deployment of FIPA platforms has increased very significantly over the period of the project - with over 150 platforms deployed at one point or another during the project. While FIPA platforms had previously been deployed in individual organisations and projects there is little doubt that Agentcities strongly encouraged this take up.

- Agentcities's adoption of DAML-OIL/OWL[CvHH+01][Hef04] and subsequent documentation in Agentcities technical recommendations undoubtedly raised awareness of the technology among Agent researchers and encourage use of it in conjunction with FIPA Agent languages.

One of the first objectives of the Agentcities project was to ensure that different implementations of the FIPA standard could interoperate. In order to achieve this, a Test Suite was created. It is a tool, available on the web, that allows to test the main interoperability features of FIPA platforms. The implementations of the FIPA standard used in the project were consequently improved to allow smooth interoperation. In turn, the feedback of this experience allowed the FIPA standard to be refined. Finally, considering the success of the deployment of the Agentcities network, several platform developers added FIPA-compliance in order to connect to Agentcities This resulted in the following outcome:

- A Test Suite that allows to test the main interoperability features of any platforms. The Test Suite remains available on the web for public use.
- A wider choice of FIPA-compliant platforms, with different types of license, allowing each to choose the best-suited platform and enhancing competition between the platforms.
- An overall better experience in FIPA platforms interoperation. This makes deployment of applications in the network more realistic

The work carried out in Agentcities has also impacted standards development in several key ways:

- Generating direct feedback and input to standards organisations such as the FIPA and the W3C.
- Encouraging the take up and re-use of standards within the research community to create an installed base of users.
- Working towards the integration of standards from different organisations to produced a unified view of how standards can be combined to serve open, dynamic service environments.

For the first point, Agentcities contributed to the FIPA standards organisation with informal inputs (to a number of Technical Committees such as Security), informal feedback from a number of Agentcities.RTD partners, formal inputs to the X2S experimental to standards conversion process and finally the submission of a formal input document documenting Agentcities use of FIPA technology and recommendations for further standardisation. The major Agentcities.RTD output documents have also been made available as resources for future FIPA standardisation. The primary formal contribution made by Agentcities.RTD to W3C standardisation was through the submission of a use case based on the project scenario that was included in the documentation leading to the W3C standardisation of the OWL ontology language. Further informal inputs have been passed on through partners involved in both Agentcities and W3C Semantic Web activities.

The second and third points are in strongly encouraging integration of FIPA (FIPA languages, FIPA Management and FIPA Message Transport specifications) and W3C (DAML-OIL and OWL) standards. The project decided to use DAML+OIL as the ontology representation framework given the current and widely disseminated trend towards DAML+OIL. This decision has shown to be well justified since W3C has chosen DAML+OIL as the basis for their Ontology Representation Standard OWL.

### Service Composition

Agentcities developed a framework for service composition in an open environment that was instantiated using the FIPA standards and a number of open-source and proprietary software toolkits into a functioning use case. The implementation was then deployed on the Agentcities testbed environment and trialled in the live environment. Service composition is the process of taking a number of independently produced services, provided by a number of different providers and organising them into a workflow so that they run to perform one group task. For example a service that summarises text, a service that provides news and a service that sends text messages may be combined together to provide a news feed to a users mobile phone.

The specific services built by the Agentcities project are constructed from the abstract service model and can be categorised as followed:

- Component Services. These are core service providers. Agentcities has built component services for Restaurants, Cinemas, Theatres and Hotels.
- Information Brokers. These are finders that provide aggregated searching across multiple Component Services. Agentcities has built finder services to match the component services (Restaurant, Cinema, Theatre and Hotel finders).
- Organising Applications. These are composite service provides which bind a number of component services and information brokers into a combined service offering. Agentcities has built the Evening Organiser application which can operate over component services and information brokers for Restaurants, Cinemas, Theatres and Hotels.

The potential of this framework is to provide a common starting point for the description of service composition solutions. This will provide (at the lowest level) a documentation standard for developers. At the higher level this will provide a mechanism for understanding and dimensioning interoperability challenges in a proposed system.

The relevance of the result is in the realisation of service-orientated architectures for software development and deployment which will enable software outsourcing and a more flexible form of ASP (application service provision). This will enable organisations to control and optimise spending on software use and provide a new business model for software developers.

### Creation of a Community

The Agentcities project has been to create an active, worldwide user community of researchers (academic and industry) applying testbed technologies to their own application areas. The number of organisations involved is approximately 100 with groups all over Europe as well as in Australia, China, Japan, New Zealand and the USA. Activities by many of the teams are supported by new research projects such as Agentcities.ES (Spain), Agentcities.UK (UK), WATAC (Finland), an Australia linkage grant (Australia) and the Europeaid @lis technology network project (Europe and Latin America).

The community is in the process of forming a non-profit body to guide the further development of the Agentcities testbed and enable collaboration between these different projects. The non-profit body and associated resources help users share information about technical problems and act as a forum for reaching consensus on emerging issues.

## 7.5 Future works

Agentcities developed the first global network environment to allow the deployment of agents, enable communication between them on the basis of standard protocols, agent languages and ontologies and enable them to dynamically discover on another through directories and support interactions between them to establish coordination relationships. The network enables high-level semantic interoperability between systems and builds knowledge and understanding of dynamic open environments for eventual transition to usage in reliable commercial grade systems.

Although the result has shown the maturity agent technology as a software paradigm to create a test-bed of interoperable software systems, many works should be done to achieve the fully interoperable systems in an open dynamic environment. By considering the different level of communications between agents, the interoperability between different ontologies is a remaining problem and must be supported in the future. And the languages in the content level must be interoperable.

In addition to the interoperability, the service composition must be worked further to be the fully automatic. The coordination technology such as the global planning will be one approach. The other related issues to solve in the open network are service discovery (meaningful description of building block services) and stability (interdependence between remote building block services). As the Agentcities network grows and becomes a more heterogeneous environment, challenging these issues for the full interoperability and automatic service composition will be important.

# Chapter 8

# Conclusion

This deliverable has revised the current state-of-the-art in agent systems (Sections 1-4), as well as standardization and validation efforts (Section 5-7).

In a first part it has provided an overview of autonomous agents and multi-agents systems, making a clear distinction between micro (agent-level) and macro (society-level) issues.

Afterwards, it has described the FIPA standardization effort (Section 5), a list of major publicly available implementations of agent platforms (Section 6) as well as the Agentcities validation framework (Section 7).

Concerning the relationship between Agent-based Services and Semantic Web Services, while the former have been studied in the last 25 years, the latter are still in their infancy. Therefore, they should take profit of the state-of-the-art in the agents domain, specially in the areas of reaching agreements, communication, and collaboration. This topic will be addressed and further elaborated in the following deliverable D2.4.4, titled "Guidelines for the integration of agent-based services and web-based services".

# Bibliography

[AC87] P. Agre and D. Chapman. PENGI: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 268–272, Seattle, WA, 1987.

[age] Agentcities web. See `http://www.agentcities.org/`.

[AIS88] J. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, pages 83–88, St. Paul, MN, 1988.

[Aus62] J. L. Austin. *How To Do Things With Words*. Oxford University Press, 1962.

[BDW01] B. Burg, J. Dale, and S. Willmott. Open standards and open source for agent-based systems, 2001.

[BFG$^+$97] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(3-4):237–256, 1997.

[BG88] A. H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[BIP88] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

[BKMS96] R. P. Bonasso, D. Kortenkamp, D. P. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 187–202. Springer-Verlag: Berlin, Germany, 1996.

[Bro86] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[Bro90] R. A. Brooks. Elephants don't play chess. In P. Maes, editor, *Designing Autonomous Agents*, pages 3–15. The MIT Press: Cambridge, MA, 1990.

[Bro91a] R. A. Brooks. Intelligence without reason. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, Sydney, Australia, 1991.

[Bro91b] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

[BS97] P. Bretier and D. Sadek. A rational agent as the kernel of a cooperative spoken dialogue system:implementing a logical theory of interaction. In *Intelligent Agents III*, pages 189–204, 1997.

[Bur95] H.-D. Burkhard. Agent-oriented programming for open systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 291–306. Springer-Verlag: Berlin, Germany, January 1995.

[Cha87] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[CL90a] P. R. Cohen and H. J. Levesque. Intention is choice with commitiment. *Artificial Intelligence*, 42:213–261, 1990.

[CL90b] P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In *Intention in Communication*, pages 221–256, 1990.

[CP79] P. R. Cohen and C. R. Perrault. Elements of a plan based theory of speech acts. *Cognitive Science*, 3:177–212, 1979.

[CvHH$^+$01] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Daml+oil reference description, w3c note, 2001. See `http://www.w3.org/TR/daml+oil-reference`.

[CW90] D. Connah and P. Wavish. An experiment in cooperation. In Y. Demazeau and J.-P. Müller, editors, *Decentralized AI — Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, pages 197–214. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.

[dam] The darpa agent markup language. See `http://www.daml.org`.

[DL87] E. H. Durfee and V. R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *Proceeding of the 10th International Joing Conference on Artificial Intelligence*, pages 875–883, 1987.

[DLC89] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Cooperative distributed problem solving. In *Handbook of Artificial Intelligence*, pages 83–147, 1989.

[Dun95] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and $n$-person games. *Artificial Intelligence*, 77:321–357, 1995.

[Dur99] E. H. Durfee. Distributed problem solving and planning. In *Multiagent Systems*, pages 121–164, 1999.

[DWB02] J. Dale, S. Willmot, and B. Burg. Agentcities: Challenges and deployment of next-generation service environments. In *Proceedings of the Pacific Rim Intelligent Multi-Agent Systems Conference*, 2002.

[EBA01] EBAY. The eBay online marketplace, 2001. See `http://www.ebay.com/`.

[EH86] E. A. Emerson and J. Y. Halpern. 'Sometimes' and 'not never' revisited: on branching time versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[ELS94] O. Etzioni, N. Lesh, and R. Segal. Building softbots for UNIX. In O. Etzioni, editor, *Software Agents — Papers from the 1994 Spring Symposium (Technical Report SS–94–03)*, pages 9–16. AAAI Press, March 1994.

[Fer92a] I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, Clare Hall, University of Cambridge, UK, November 1992. (Also available as Technical Report No. 273, University of Cambridge Computer Laboratory).

[Fer92b] I. A. Ferguson. Towards an architecture for adaptive, rational, mobile agents. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 249–262. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.

[Fer95] I. A. Ferguson. Integrated control and coordinated behaviour: A case for agent models. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 203–218. Springer-Verlag: Berlin, Germany, January 1995.

[FIP99] FIPA. Specification part 2 — Agent communication language, 1999. The text refers to the specification dated 16 April 1999.

[Fir87] J. A. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87)*, pages 202–206, Milan, Italy, 1987.

[Fis94] M. Fisher. A survey of Concurrent METATEM — the language and its applications. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic — Proceedings of the First International Conference (LNAI Volume 827)*, pages 480–505. Springer-Verlag: Berlin, Germany, July 1994.

[Fis95] M. Fisher. Representing and executing agent-based systems. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 307–323. Springer-Verlag: Berlin, Germany, January 1995.

[FN71] R. E. Fikes and N. Nilsson. Strips: a new approach to the application of theorem proving to prblem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Gal88] J. R. Galliers. *A Theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Multi-Agent Conflict*. PhD thesis, Open University, UK, 1988.

[Geo87] M. P. Georgeff. Planning. *Annual Review of Computer Science*, 2:359–400, 1987.

[GF92] M. R. Genesereth and R. E. Fikes. Knowledge interchange format, version 3.0 reference manual. *Technical report, Computer Science Department, Stanford University*, 92-1, 1992.

[GGR83] M. R. Genesereth, M. Ginsberg, and J. S. Rosenschein. Cooperation without communication. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 125–129, 1983.

[GH89] L. Gasser and M. Huhns, editors. *Distributed Artificial Intelligence Volume II*. Pitman/Morgan Kaufman, 1989.

[Gil95] N. Gilbert. Emergence in social simulation. In N. Gilbert and R. Conte, editors, *Artificial Societies: The Computer Simulation of Social Life*, pages 144–156. UCL Press: London, 1995.

[GL87] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682, Seattle, WA, 1987.

[Hef04] J. Heflin. Owl web ontology language. use cases and requirements, w3c recommendation, 2004. See `http://www.w3.org/TR/webont-req/`.

[HRHW$^+$89] B. Hayes-Roth, M. Hewett, R. Washington, R. Hewett, and A. Seiver. Distributing intelligence within an individual. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*, pages 385–412. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.

[Huh87] M. Huhns, editor. *Distributed Artificial Intelligence*. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1987.

[Jen92] N. R. Jennings. Towards a cooperation knowledge level for collaborative problem solving. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 224–228, Vienna, Austria, 1992.

[Kae86] L. P. Kaelbling. An architecture for intelligent reactive systems. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning About Actions & Plans — Proceedings of the 1986 Workshop*, pages 395–410. Morgan Kaufmann Publishers: San Mateo, CA, 1986.

[KLR$^+$92] D. Kinny, M. Ljungberg, A. S. Rao, E. Sonenberg, G. Tidhar, and E. Werner. Planned team activity. In C. Castelfranchi and E. Werner, editors, *Artificial Social Systems — Selected Papers from the Fourth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW-92 (LNAI Volume 830)*, pages 226–256. Springer-Verlag: Berlin, Germany, 1992.

[Kra01] S. Kraus. *Strategic Negotiation in Multiagent Environments*. The MIT Press: Cambridge, MA, 2001.

[KSE98] S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation: a logical model and implementation. *Artificial Intelligence*, 104:1–69, 1998.

[Lew69] D. Lewis. *Convention - a Philosophical Study*. Harvard University Press, 1969.

[LLL$^+$96] Y. Lésperance, H. J. Levesque, F. Lin, D. Marcu, R. Reiter, and R. B. Scherl. Foundations of a logical approach to agent programming. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *Intelligent Agents II (LNAI Volume 1037)*, pages 331–346. Springer-Verlag: Berlin, Germany, 1996.

[Mae90a] P. Maes, editor. *Designing Autonomous Agents*. The MIT Press: Cambridge, MA, 1990.

[Mae90b] P. Maes. Situated agents can have goals. In P. Maes, editor, *Designing Autonomous Agents*, pages 49–70. The MIT Press: Cambridge, MA, 1990.

[Mae91] P. Maes. The agent network architecture (ANA). *SIGART Bulletin*, 2(4):115–120, 1991.

[MLF96] J. Mayfielod, Y. Labrou, and T. Finin. Evaluation kqml as an agent communication language. In *Intelligent Agents II*, pages 347–360, 1996.

[Mül97] J. Müller. A cooperation model for autonomous agents. In J. P. Müller, M. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III (LNAI Volume 1193)*, pages 245–260. Springer-Verlag: Berlin, Germany, 1997.

[net] Agentcities european project (agentcities.net). See `http://www.agentcities.org/EUNET/`.

[NS61] A. Newell and H. A. Simon. GPS: A program that simulates human thought. In *Lernende Automaten*. R. Oldenbourg, KG, 1961.

[OR90] M. J. Osborne and A. Rubinstein. *Bargaining and Markets*. The Academic Press: London, England, 1990.

[owl03] Owl-s: Semantic markup for web services, version 1.0, 2003. See `http://www.daml.org/services/owl-s/1.0/owl-s.pdf`.

[Pat92] R. S. Patil. The darpa knowledge sharing effort: progress report. In *Proceeding if Knowledge Representation and Reasoning*, pages 777–788, 1992.

[Pog95] A. Poggi. DAISY: An object-oriented system for distributed artificial intelligence. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 341–354. Springer-Verlag: Berlin, Germany, January 1995.

[PR00] A. Poggi and G. Rimassa. Adding extensible synchronization capabilities to the agent model of a fipa-compliant agent platform. In *Proceeding of the First International Workshop AOSE*, pages 307–322, 2000.

[PV01] H. Prakken and G. Vreeswijk. Logics for defeasible argumentation. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic (second edition)*. Kluwer Academic Publishers: Dordrecht, The Netherlands, 2001.

[rep03] Deliverable d1.4 final report agentcities.rtd, 2003.

[RG85] J. S. Rosenschein and M. R. Genesereth. Deals among rational agents. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, pages 91–99, Los Angeles, CA, 1985.

[RG91] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.

[RG95a] A. S. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319, San Francisco, CA, June 1995.

[RG95b] A. S. Rao and M. P. Georgeff. Formal models and decision procedures for multi-agent systems. Technical Note 61, Australian AI Institute, Level 6, 171 La Trobe Street, Melbourne, Australia, June 1995.

[RGS92] A. S. Rao, M. P. Georgeff, and E. A. Sonenberg. Social plans: A preliminary report. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 57–76. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.

[RK86] S. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In J. Y. Halpern, editor, *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge*, pages 83–98. Morgan Kaufmann Publishers: San Mateo, CA, 1986.

[RLK04] D. Roman, H. Lausen, and U. Keller. Web service modeling ontology - standard (wsmo - standard), version 0.2, 2004. See `http://www.wsmo.org/2004/d2/v02/`.

[RSP93] S. J. Russell, D. Subramanian, and R. Parr. Provably bounded optimal agents. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 338–344, Chambéry, France, 1993.

[rtd] Agentcities european project (agentcities.rtd). See `http://www.agentcities.org/EURTD/`.

[RW91] S. J. Russell and E. Wefald. *Do the Right Thing — Studies in Limited Rationality*. The MIT Press: Cambridge, MA, 1991.

[RZ94] J. S. Rosenschein and G. Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*. The MIT Press: Cambridge, MA, 1994.

[Sac74] E. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[Sac75] E. Sacerdoti. The non-linear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, pages 206–214, Stanford, CA, 1975.

[SD80] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. volume 11, 1980.

[Sea69] J. R. Searle. *Speech Acts: an Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[Sho90] Y. Shoham. Agent-oriented programming. Technical Report STAN–CS–1335–90, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.

[Sho93] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[Ste90] L. Steels. Cooperation between distributed agents through self organization. In Y. Demazeau and J.-P. Müller, editors, *Decentralized AI — Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-89)*, pages 175–196. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990.

[Syc90] K. P. Sycara. Persuasive argumentation in negotiation. *Theory and Decision*, 28:203–242, 1990.

[Tho93] S. R. Thomas. *PLACA, an Agent Oriented Programming Language*. PhD thesis, Computer Science Department, Stanford University, Stanford, CA 94305, August 1993. (Available as technical report STAN–CS–93–1487).

[Tho95] S. R. Thomas. The PLACA agent programming language. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 355–369. Springer-Verlag: Berlin, Germany, January 1995.

[VB90] S. Vere and T. Bickmore. A basic agent. *Computational Intelligence*, 6:41–60, 1990.

[vEHKB02] R. van Eijk, J. Hamers, T. Klos, and M. S. Bargh. Agent technology for designing personalized mobile service brokerage. 2002.

[vM90] F. von Martial. Interactions among autonomous planning agents. In *Proceeding of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 105–120, 1990.

[Wav92] P. Wavish. Exploiting emergent behaviour in multi-agent systems. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 297–310. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.

[WG95] P. Wavish and M. Graham. Roles, skills, and behaviour: a situated action approach to organising systems of interacting agents. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures,* *and Languages (LNAI Volume 890)*, pages 371–385. Springer-Verlag: Berlin, Germany, January 1995.

[Whi94] J. E. White. Telescript technology: The foundation for the electronic marketplace. White paper, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040, 1994.

[Wil88] D. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[WJ95] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[Woo95] M. Wooldridge. This is MyWorld: The logic of an agent-oriented testbed for DAI. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 160–178. Springer-Verlag: Berlin, Germany, January 1995.

[Woo01] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiely, 2001.

[WRR95] D. Weerasooriya, A. Rao, and K. Ramamohanarao. Design of a concurrent agent-oriented language. In M. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 386–402. Springer-Verlag: Berlin, Germany, January 1995.