AISB Journal

The Interdisciplinary Journal of
Artificial Intelligence and the Simulation of Behaviour

Volume 1 – Number 1 – December 2001

The Journal of the
Society for the Study of Artificial Intelligence
and the Simulation of Behaviour
http://www.aisb.org.uk

Published by
The Society for the Study of Artificial Intelligence and Simulation of Behaviour

http://www.aisb.org.uk/

ISSN 1476-3036

Contents

Preface

It is with great pleasure that I welcome the reader to the first issue of AISBJ, the Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour.

AISBJ has evolved from the former AISB Quarterly, which began life as a newsletter, and then developed into a quarterly referreed journal. Now, finally, AISBJ has formed its own separate species, leaving AISBQ to excel in its original function as a quarterly newsletter for the membership of AISB.

The past few years have been an important period for AISB (whose full name is SSAISB, the Society for the Study of Artificial Intelligence and the Simulation of Behaviour). We have changed from a shrinking, financially challenged, relatively passive grouping into a growing, fiscally controlled and stable organisation for the promotion of AI and Cognitive Science in the UK. We have exceeded predictions annually for the past three years in terms of the success of the AISB Convention, in a format first tried out at Edinburgh in 1999, and we expect to continue this success in April 2002 at Imperial College in London. For the first time, as well, the Society has its own web site, at `www.aisb.org.uk`, and materials and submission methods for AISBJ are published there.

The time is indeed ripe, therefore, for a formal mechanism for the promulgation of the writing of AISB members and others, which reflects the nature of the Society at work. AISB Conventions are multi-disciplinary events, bringing together researchers from all areas of AI and Cogntive Science, and also others from outside those fields. AI, in particular, has tended to fragment in the past decade, for good reason, as it matures as a field, broadening and deepening. While this is clearly the consequence of positive developments, it can also be a drawback, and AISBJ aims, at the referreed journal level, to bridge the potential gaps arising from specialisation, just as the AISB Convention does at the conference level.

This inaugural issue is a fine example, based around agent technology, but bringing together papers developed from three separate specialist symposia in the 2001 Convention at York. I would congratulate Simon Colton, his co-editors, reviewers and all the authors of these papers on producing an excellent first volume of the new Journal, especially in the face of the technical teething problems that always dog such a new enterprise.

Special thanks must go to Medeni Fordham, AISB's office manager, who has been closely and patiently involved with the design and administration of the new Journal, and to Penousal Machado, of the University of Coimbra, in Portugal, who, together with his creative program, NeVar, produced the cover design.

Now it's over to the AISB community. Please submit your excellent work to the new Journal, to help make it our mouthpiece on the world stage.

Geraint A. Wiggins
Chair, AISB

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Editorial: Special Issue on Agent Technology

Eduardo Alonso[⋆], Simon Colton[†], Daniel Kudenko[◇],
Luc Moreau[‡], Michael Schroeder[⋆] and Kostas Stathis[⋆]

[⋆] Department of Computing, City University, London, Northampton Square, London, EC1V OHB, UK. {*eduardo,msch,kostas*}*@soi.city.ac.uk*

[†] Division of Informatics, University of Edinburgh, 80 South Bridge, Edinburgh, EH1 1HN, UK. *simonco@dai.ed.ac.uk*

[◇] Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK. *kudenko@cs.york.ac.uk*

[‡] Department of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK. *L.Moreau@ecs.soton.ac.uk*

1 Introduction

For the last two decades, agent-based technology has been applied to building intelligent systems. The ability of agents to make their own decisions and adapt to and learn from their environment is a very powerful tool for implementing intelligent systems of general competence. In addition, the study of multi-agent systems — collections of specialised agents working in parallel — has helped us to solve complex problems not previously solved using a centralised approach. Moreover, agent-based simulators have been used to better understand cooperation and competition in society. It is therefore not surprising that agent-based systems have been successful in many varied domains, including e-commerce, online learning, medicine, entertainment, human-computer interaction, business management, traffic control and conflict simulation.

Because agent technology is such a key area of Artificial Intelligence, and as Britain is strongly represented in this area, the Society for the Study of Artificial Intelligence and the Simulation of Behaviour decided to make agent technology the theme of their 2001 convention (AISB'01), held in York. Five invited talks in the plenary sessions at AISB'01 were on agent-based topics. Nick Jennings discussed "Automated Haggling: Building Artificial Negotiators", Lyndon Lee described the "Multi-Agent Research at British Telecom", Andrew Jones spoke "On the Concept of Trust", Christoph Benzmüller presented "An Agent Based Approach to Reasoning" and Jim Doran described "Agents and Ecosystem Management: from the Fraser River to Boolean Networks".

Furthermore, the convention included symposia on (a) Software Mobility and Adaptive Behaviour, (b) Information Agents for Electronic Commerce, and (c) Adaptive Agents and Multi-Agent Systems. We invited certain authors from these symposia to submit extended papers to this special issue, so that we could present a broad cross-section of cutting edge agent technology in one volume. In the three sections below, we briefly describe the three subdomains of agent technology represented here, and the specific work from the authors with papers in this volume.

2 Adaptive Agents and Multi-Agent Systems

When designing agent systems, it is impossible to foresee all the potential situations an agent may encounter and specify an agent behaviour optimally in advance. Agents therefore have to learn from and adapt to their environment. This task is even more complex when nature is not the only source of uncertainty, and the agent is situated in an environment that contains other agents with potentially different capabilities, goals, and beliefs. Multi-Agent Learning, i.e., the ability of the agents to learn how to cooperate and compete, becomes crucial in such domains.

Even though Machine Learning (ML) has been studied extensively in the past, ML research has been mostly independent of agent research and only in recent years has it received more attention in connection with Agents and Multi-Agent Systems. This is in some ways surprising, because the ability to learn and adapt is one of (if not the) most important feature of intelligence and autonomy. Nowadays, the integration of ML technology into agent systems has become a major challenge.

Research in Machine Learning for agents and multi-agent systems is still in the beginning stages, and many issues are still unresolved. Amongst these issues are the question of the source and the proper selection of training data (e.g., the credit assignment problem), how to achieve coordinated and specialized behaviour in a team of learning agents, the timeliness of learning problems, and many more (see, for example, (Sen and Weiss, 1999) and (Kazakov and Kudenko, 2001) for overviews). While reinforcement learning has become the predominant learning technique for agents, very recently the focus is shifting towards hybrid solutions that incorporate other ML approaches. We expect to see this trend gaining momentum in the near future.

The Symposium on Adaptive Agents and Multi-Agent Systems at AISB'01 was a pioneering experience, as no symposium on learning agents had been organized previously in the UK. With the symposium, we intended to increase awareness and interest in adaptive agent research in the European AI community, and encourage further research and collaboration between Machine Learning experts and Agent Systems experts. Last but not least, we intended to give a representative overview of current research in the area of adaptive agents in Europe and worldwide. For this Special Issue, we have chosen three papers from our symposium. Firstly, Frances M. T. Brazier and Niek. J. E. Wijngaards introduce automated servicing as a way of designing adaptable agents, an idea that does not draw on the classical view of machine learning, but sees adaptation as an externally driven process. Secondly, Pedro Rafael Graça and Graça Gaspar propose an agent-architecture to deal with real-time (group communication) problems where it is important both to react to constant changes in the environment and to learn to recognize the generic tendencies in the sequence of those changes. Finally, Chia-Hsuan Yeh and Shu-Heng Chen's paper presents an economic simulation based on an artificial stock market. Their simulation is used to study how economic heterogeneity improves market efficiency.

3 Information Agents for Electronic Commerce

Internet connectivity is creating an information-centric society where millions of people access large amounts of information stored in distinct and possibly heterogeneous data sources, on a daily basis. Frequently, pieces of information from different sources can be combined to create new and often more meaningful information. As a way of automating this process, a research strategy is currently being devoted to the development of information agents (Klusch, 1999), software components that can act on behalf of individual

users to manage intelligently the information one typically finds in distributed networks like the Internet.

However, as the complexity of creating new information increases (finding the right pieces to produce a new one can be quite time consuming), the concept of information is slowly transforming itself from a freely accessible entity to that of a product with a price. This transformation has opened up new markets where information is traded electronically much the same way a physical product is (Cohen and Stathis, 2001). However, the importance of information markets is that physical products are not excluded, in that these too can be represented as pieces of information.

The symposium on Information Agents for Electronic Commerce at AISB'01 presented current research on a wide range of issues, in particular, where activities for trading information in an electronic market could take place between people, organizations or both (Dignum and Sierra, 2001). The papers selected for this special issue address the three particular aspects of service composition, user-agent interaction, and specification of agent behaviour using computational logic and constraints.

In "Towards Agent-Based Service Composition Through Negotiation in Multiple Auctions", Chris Preist, Andrew Byde, Claudio Bartolini and Giacomo Piccinelli go beyond the use of auctions for basic services and investigate how to compose services and in particular how to use auctions to negotiate about such composed services. In "Designing Agents for a Virtual Marketplace", Kaveh Kamyab, Frank Guerin, Petar Goulev and Ebrahim Mamdani design an adaptive virtual sales assistant, which builds on a Belief Desires and Intentions (BDI) architecture and fuzzy rules to engage in interaction with the user. Finally, in "BDI Agents and Constraint Logic", Stuart Chalmers and Peter M. D. Gray declarative specify an electronic commerce application where the internal architecture of the agents is based on the BDI model too, but in this case, using constraints as the main representation and implementation formalism.

4 Software Mobility and Adaptive Agents

Mobile agents have emerged as a paradigm for structuring distributed applications. A mobile agent is a program, which can, at runtime, make autonomous decisions about the locations where it will continue its execution (Lange and Ishima, 1998). Other methods for providing software mobility have been proposed, including evolutionary and self-organising systems, ants, mobile agents, mobile code and active networks.

Software mobility can bring robustness, performance, scalability or expressiveness to systems. It has been used successfully in different application domains, including network management, Internet resource discovery, electronic market places and interaction with mobile users. The topic of mobility is the object of active research, and many varied aspects are under investigation, such as communications between mobile entities, languages to express mobility and their typing, management of resources and security infrastructure for mobile code. The development of a theoretical framework for describing and reasoning about systems with mobility has also received much attention. In particular, the Ambient calculus (Cardelli and Gordon, 1998) introduces the idea of a location delimited by boundaries and operations to move across such boundaries.

For this Special Issue, we have chosen two papers from the symposium on Software Mobility and Adaptive Behaviour. First, Lidia Yamamoto and Guy Leduc present a reflector service to maintain application level-connectivity in the presence of network-level multicast failures; this service relies on mobile code and makes use of migration to adapt its behaviour to the network configuration. Second, Julian Padget presents a specification

of electronic institutions in process algebras extended with locations; he examines some key aspects of a prototypical e-institution using the Seal and type-safe Ambient calculi in order to compare their properties.

Acknowledgements

We wish to thank everyone involved in the organisation of the AISB'01 convention, including the invited speakers, symposium organisers, conference office staff, the technical support and secretarial staff, and the helpdeskers, without whom the conference would have collapsed. In particular, we would like to thank the programme committee members for the three AISB'01 symposia represented here, for their help in attracting, selecting and reviewing papers, not just for the symposia, but also for this volume: Jean-Pierre Briot, Mark D'Inverno, Michael Fisher, Christophe Giraud-Carrier, Lyndon Lee, Michael Luck, Scott Moss, Joerg Mueller, Simon Parsons, Jeremy Pitt, Chris Preist, Omer Rana, Marek Sergot, Carles Sierra, Francesca Toni, Jan Vitek, Gerd Wagner, Ian Wakeman and Mike Wooldridge.

Special thanks to Enric Plaza for giving an invited talk at the symposium on Adaptive Agents and Multi-Agent Systems, and the AgentLinkII network for funding his visit. We wish to join Geraint Wiggins in thanking Medeni Fordham for her invaluable help in the production of the AISBJ. We wish to praise Geraint himself for his inspiration in getting the AISB Journal off the ground. Not only did he suggest the split from the AISBQ and that the first issue be on agents, but, as chair of SSAISB, he has also been involved with every aspect of creating this new journal.

References

Cardelli, L. and Gordon, A. (1998). Mobile Ambients. In Nivat, M., editor, *Foundations of Software Science and Computational Structures, Lecture Notes in Computer Science*, volume 1378, pages 140–155. Springer-Verlag.

Cohen, M. and Stathis, K. (2001). Strategic change stemming from E-commerce: implications of multi-agent systems on the supply chain. *Journal of Strategic Change*, 10:139–149.

Dignum, F. and Sierra, C. (2001). *Agent Mediated Electronic Commerce. The European AgentLink Perspective. Lecture Notes in Computer Science*, volume 1993. Springer-Verlag.

Kazakov, D. and Kudenko, D. (2001). Machine learning and inductive logic programming for multi-agent systems. In Luck, M., Marik, V., Stepankova, O., and Trappl, R., editors, *Multi-Agent Systems and Applications*. Springer-Verlag.

Klusch, M. (1999). *Intelligent Information Agents - Agent-Based Information Discovery and Management on the Internet*. Springer-Verlag.

Lange, D. and Ishima, M. (1998). *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley.

Sen, S. and Weiss, G. (1999). Learning in multi-agent systems. In Weiss, G., editor, *Multi-Agent Systems*. MIT Press.

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Automated Servicing of Agents

Frances M. T. Brazier and Niek J. E. Wijngaards

Intelligent Interactive Distributed Systems Group,
Faculty of Sciences, Vrije Universiteit Amsterdam,
de Boelelaan 1081a; 1081 HV Amsterdam, The Netherlands
frances@cs.vu.nl ; niek@cs.vu.nl

Abstract

Agents need to be able to adapt to changes in their environment. One way to achieve this, is to service agents when needed. A separate servicing facility, an agent factory, is capable of automatically modifying agents. This paper discusses the feasibility of automated servicing.

1 Introduction

Agents typically operate in dynamic environments. Agents come and go, objects appear and disappear, and cultures and conventions change. Whenever the environment of an agent changes to the extent that it is unable to cope with (parts of) the environment, the agent needs to adapt. Changes in the social environment of an agent, for example, may require modifications to existing agents. A new agent communication language, or new protocols for auctions, are examples of such changes. An agent may be able to detect gaps in its abilities; but it may not be able to fill these gaps with its own built-in learning mechanisms. Whether the need for servicing is detected by an agent itself, or by another agent (automated or human) is irrelevant to the concept involved: external assistance may be needed to perform the necessary modifications.

This paper discusses the feasibility of a service for automated revision. In Section 2, needs for adaptation are discussed. An automated servicing facility, an agent factory, is described in Section 3. An example of adapting an agent, based on an existing prototype automated servicing service, is provided in Section 4. The feasibility of such a service for automated revision is discussed in Section 5, in which the agent factory is also compared to related approaches. The results presented in this paper are discussed in Section 6.

2 Adaptive Agents

Both static and mobile agents may encounter the need for adaptation. In this section, an example is used to illustrate a few situations in which external adaptation is feasible.

The focus in this example is on an information gathering agent. The information gathering agent is assumed to be mobile. Its task is to find information for a researcher about travel arrangements needed to attend a conference. To this purpose, the agent communicates with three other agents (a personal assistant agent, a travel agent, and a bank agent) and interacts with the World-Wide Web.

Example 1. The personal assistant agent informs the information gathering agent about its preferences with respect to travel agents, and about the researcher's travel preferences. The personal assistant agent has acquired some of this information directly from the researcher, and has acquired some over the course of time from the researcher and from its own experience. The information gathering agent maintains a profile of the personal assistant agent, and adapts this profile on the basis of interaction with the personal assistant agent (e.g., as encountered in negotiation settings (Bui et al., 1996)). Note that in this example personification is not aimed at personalising an agent's representation of a human user (e.g., see (Soltysiak and Crabtree, 1998; Wells and Wolfers, 2000)), but the profile of the personal assistant agent.

Example 2. The information gathering agent consults the World-Wide Web to find dates and a location for the aforementioned conference. The conference page is annotated in an ontology that is unfamiliar to the agent. For example, OIL (Fensel et al., 2000; Horrocks et al., 2001) has been used instead of XML (Bray et al., 2000). One way to approach this problem is to have the information gathering agent acquire understanding of this ontology. Another option is to use an intermediary agent (e.g., brokers/matchmakers (Wong and Sycara, 2000)) to find an agent capable of translating between ontologies, e.g. via SOAP, (the Simple Object Access Protocol (Box et al., 2000)). In this last case, the agent needs to "travel" and collaborate with its new assistant during interaction with the conference site.

The information gathering agent also wishes to discuss possible travel arrangements with the travel agent. The travel agent indicates that it only "speaks" a specific language and protocol. The information gathering agent needs to acquire this agent language and protocol. This is comparable to the acquisition of a new ontology sketched above.

Example 3. During the discussion between the information gathering agent and the travel agent, the issue of credit rating arises. The information gathering agent needs to prove to the travel agent that it is trustworthy and has an acceptable credit rating. In addition to security clearance on its own trustworthiness, the information gathering agent needs additional information from the personal assistant agent. The information gathering agent needs permission to ask a bank for this information and the name and address of a specific bank agent. The bank agent, in turn, requires certificates and a guarantee from the information gathering agent that specific security measures are in place, before it will provide any other information. If the information gathering agent does not have this functionality it may be possible to add this functionality to the agent. (Please note that adding functionality may not be the only measure that needs to be taken in this case).

In each of the situations sketched above, an automated servicing process is to be used. The types of adaptation involved are:

- Personalisation: an agent can be provided with profiles specific to its current co-operation partners.
- Domain and languages: an agent can be adapted to include knowledge about a specific domain to understand a specific agent communication language and protocol.
- Functionality: new functionality or characteristics can be added to (or deleted from) an agent.

3 An Agent Factory

An automated agent servicing facility – an agent factory – is described in this section. The agent factory, in essence, re-designs descriptions of agents. Previous research (Brazier et al., 2000a; Brazier et al., 2000b) focussed on automated redesign of multi-agent systems at a detailed (conceptual) level. The automated servicing service described in this paper is an extension of this work in two ways.

The first distinction with the previous work is that the agent factory as presented in this paper is not primarily focussed on re-designing agents on the basis of first principles on a conceptual level, as described in (Brazier et al., 2000b). The agent factory uses building blocks to construct, and adapt, agents. Building blocks can be templates, i.e. skeletons that describe the architecture of a (larger) part of an agent. Components are building blocks with specific functionality. Templates and components are combined according to pre-defined rules.

The second distinction with previous work is a broadening of the scope of the re-design process. The agent factory modifies not only the conceptual description of an agent, but also its operational code. This necessitates knowledge about the relationship between the conceptual description and detailed (operational) description of templates and components.

On the basis of a need for adaptation, the automated servicing process re-configures templates and components at both levels. Re-configuration (an instance of a re-design process) of an agent first takes place at the conceptual level: templates and components are removed and added until a satisfactory conceptual agent description is acquired. On the basis of the configuration of templates and components in the conceptual description of an agent, a detailed (operational) description of an agent is generated.

To facilitate the automated re-design of agents, a number of assumptions have been made on the descriptions of an agent (Section 3.1). In addition, the agent factory has a library of building blocks, the so-called templates and components (Section 3.2). The configuration task of the agent factory (Section 3.5) is based on knowledge of the characteristics and properties (Section 3.3), and the availability of templates and components (Section 3.4).

3.1 Assumptions on the design of agents

The feasibility of an automated service for revision of agents depends largely on the assumptions imposed on the design of the agents. The most important underlying assumptions for an agent adaptation service used in this paper are as follows.

The first assumption is that agents have a compositional structure. A compositional structure greatly facilitates the possibilities of adding, removing and changing parts of an agent. This principle is used throughout software design, ranging from describing processes (e.g., JSD (Jackson, 1975)), via object-oriented programming (e.g., (Booch, 1991; Pressman, 1997; Wieringa, 1996)) to component-based programming (e.g., (Hopkins, 2000)).

The second assumption is that re-usable parts of agents can be identified: templates (i.e., skeletons) and components (i.e., building blocks). The agent factory can build an agent by correctly configuring templates and components. This assumption relates to design patterns (e.g., (Gamma et al., 1994; Peña-Mora and Vadhavkar, 1996; Riel, 1996)) and libraries of software with specific functionality (e.g., problem-solving models (Schreiber et al., 1999) or generic task models (Brazier et al., 1998)).

The third assumption is that templates and components are described at two levels of abstraction: a conceptual description and a detailed description. This assumption circumvents two problems. On the one hand, it is difficult to determine a conceptual description on the basis of a detailed, operational description of (part of) a system (e.g., (Jackson, 1995)). On the other hand, it is again difficult to determine the operational description of (part of) a system on the basis of a conceptual description (e.g., (Rumbaugh et al., 1999)). In the case of the agent factory, the detailed description is also an operational description.

The fourth assumption is that properties and knowledge of properties are available to describe templates and components. Interfaces provided and required by templates and components need to be described (e.g., as is done in work on describing classes of diagnostic (non-user-interactive) problem-solving methods by (Benjamins, 1995)).

A fifth assumption is that no commitments have been made to specific languages and/or ontologies. The languages used for the descriptions of templates and components on both levels of abstraction are left open, as are the descriptions, and contents, of the properties and knowledge on properties to describe templates and components. The agent factory is explicitly developed to be an open architecture.

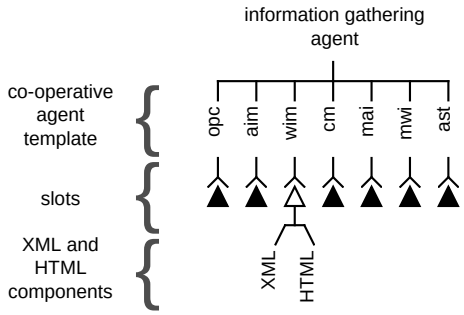


Figure 1: Graphical representation of templates and components and their slots.

3.2 Templates and components

Templates and components are the building blocks with which agents are constructed. Templates are skeletons which describe an architecture of a (larger) part of an agent. A template is usually combined with a number of (other) templates and/or components. A component is a building block with specific functionality.

For each conceptual description, a number of detailed, operational descriptions may be devised. These operational descriptions may differ in the operational language (e.g., C, C++, Java), but also in, for example, the efficiency of the operational code.

Templates and components are configurable. However, templates or components cannot be combined indiscriminately. The open slot concept is used to regulate the ways in which templates and components may be combined. An open slot in a template or component has associated properties that prescribe the properties of the entity to be 'inserted' in addition to the interface of the required building block.

A mapping relation is defined between building blocks containing conceptual descriptions and building blocks containing detailed descriptions. Each conceptual building block may be related to a number of detailed building blocks; the inverse may hold as well.

Templates specify the architecture of an information gathering agent. In figure 1, the information gathering agent is shown to consist of seven processes (as explained in Section 4). Each of these processes has a slot, which is filled by a combination of templates and/or components. The open slot for the world interaction management process (wim), is shown to be filled with two components, which provide specific functionality to interact with web pages annotated in HTML and XML.

An "open-slot preserving" relationship is defined in the mapping relation between building blocks, so that each open slot in a conceptual template or component is related to an open slot in the associated detailed template or component. The open-slot preserving relationship between related conceptual and operational building blocks implies that templates and components are combined in the same configuration at both levels of abstraction. The two-stage revision process facilitates the generation of operational code: on the basis of the configuration of templates and components at a conceptual level, the detailed, operational code is generated in a relatively straightforward manner, as explained in the next section.

3.3 Details of templates and components

The building blocks used by the agent factory, templates and components, have the same structure, as depicted in figure 2. This structure does not make a commitment to specific conceptual or detailed (operational) description languages, but includes types of information that are also included in structures designed to describe design patterns (e.g., (Gamma et al., 1994)).

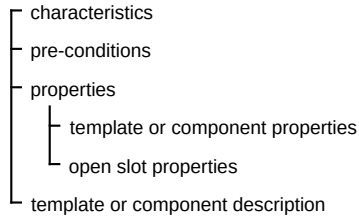


Figure 2: Structure of templates and components used by the agent factory.

The characteristics of a building block describe its name, creation dates, authors, version information, and level of abstraction. This information is not related to the description inside the building block.

The pre-conditions contain assumptions and requirements of the interface of the building block that have to be satisfied by the environment (i.e., an open slot and the template or component containing that open slot) in which this building block is to be placed. For example, a building block which contains a specific sorting algorithm, may require as its input an unordered list of elements, where each element consists of an unknown part and an explicit key. In addition, the pre-conditions describe which languages are used in the description.

The properties of a building block are divided into properties concerning the templates and components, and properties concerning open slots. Examples of properties of a conceptual template containing a skeleton for an agent are: it is autonomous, it is capable of communicating with other agents, it is capable of interacting in the world, it is capable of retaining information on other agents and the world. Properties of an open slot may be, for example, that a specific open slot contains an agent communication language syntax

expressed in XML. Template properties at a detailed (operational) level include properties such as: an agent is a process, the size is so many bytes, and the datastructure is of a specific class. Properties of open slots are, for example, that the first argument in a specific open slot contains the input-information for a specific process, and the second argument contains a pointer to a data structure of a specific class for the results.

3.4 Retrieving building blocks

The agent factory is able to retrieve templates and components on the basis of needs for adaptation. The re-design process inside the agent factory analyses needs for adaptation and transforms these into requirements (on structure, functionality, and behaviour) on agents to be constructed. The agent design is a configuration of templates and components that satisfies these requirements.

Matching requirements on structure, functionality, and behaviour of (parts of) agents to properties of templates and components is not trivial. Requirements may be incomplete, conflicting, or vague. To solve this problem, a matching process is needed which has some understanding of the properties involved.

Properties are related to each other in property networks. This allows generic properties to be, for example, refined into a number of sets of more refined properties. Two assumptions are made: if a more generic property of an agent holds, then at least one set of refined properties holds. If all refined properties of one set hold, then the more generic property also holds.

A number of refinements may exist for a specific property, each of which can be included in a refinement tree. Refinement trees can be combined into property networks. In these networks, it is possible to explore alternative refinements of a property. For example, the property that a specific algorithm is a sorting algorithm can be refined into more specific properties on efficiency, e.g. sorting algorithms in linear time, in $O(^{n}log(n))$ time, etc. Alternatively, the 'sorting algorithm' property may be refined into more specific properties on the number of keys used: one key, one primary key and one secondary key, etc. Yet another alternative is that this property is, in itself, a refined property of a property expressing that an algorithm is a classification algorithm.

The matching process has variable forms of interpretation. One form is that no interpretation is used at all (syntactical or exact matching), so that a required property needs to be explicitly present in a building block. An alternative is to use property refinement: a high level property (e.g., an algorithm which orders a list of elements) for which no building block can be found, can be refined into a more specific property (e.g., an algorithm which orders an array of elements in $O(n^2)$ time), for which a building block can be found. Usually, a building block will exist with a more specific property, which can then fulfil the desired property.

A more elaborate means of query interpretation is by traversing semantic property networks. This usually returns a 'good guess', but not necessarily an optimal answer as a building block with similar properties is returned. The notion of 'similar' can be tuned (e.g., what distance to travel through property networks).

3.5 The process of adaptation

The agent factory is able to adapt an existing agent on the basis of needs for adaptation. The agent factory re-designs agents. The agent factory first obtains an initial set of required properties (the needs for adaptation) and a description of the agent to be adapted.

The initial set of required properties is analysed and manipulated (e.g., interpreted, conflicts are resolved, etc.) to form a set of refined required properties that are still related to the initial set, yet are more specific. This may already involve checking the library of templates and components for the presence of templates and components with specific properties (it makes no sense to require, for example, a sorting algorithm in O(1/n) time if there are no such building blocks in the library).

On the basis of such a more specific set of required properties, the conceptual description of the agent is adapted. Building blocks are inserted, moved, and/or deleted, until the required properties are satisfied if possible. Additional adaptation of the set of required properties may be necessary (if, for example, the required properties prove to be conflicting). A new set of required properties may be constructed, based on both the previous set of required properties and evaluations of the success or failure in constructing a satisfactory conceptual description.

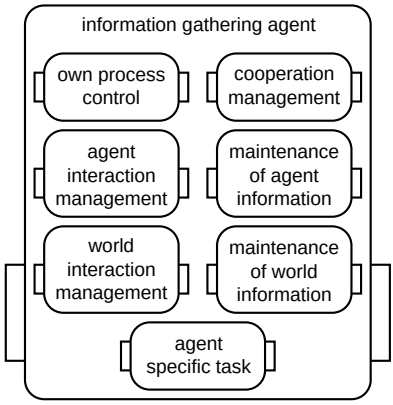


Figure 3: The seven processes inside the information gathering agent. Each process, including the agent itself, has an interface. Between processes, information transfer is defined (not shown).

At some point in this cycle, the conceptual description of an agent is analysed to check whether it satisfies a specific set of required properties (based on the initial set of required properties). If this point has been reached, the agent factory focusses on adapting the detailed, operational description of the agent. If not, the agent factory may adapt the set of required properties.

The operational description of an agent is based on the configuration of templates and components in the conceptual description of the agent. If problems occur in combining operational descriptions from templates and components, either other templates and components are used (with the same conceptual description and properties, but different operational description and properties) or a different conceptual description of the agent is needed. The process described above is then repeated with additional requirements (i.e., required properties).

4 Automated Servicing of an Information Gathering Agent

In this paper, an example is given of an agent that requires servicing. The adaptation of an information gathering agent in this example is based on an existing prototype automated

servicing service. The conceptual descriptions of the templates and components are specified in the DESIRE knowledge-level modelling language (Brazier et al., 1998) and the operational descriptions are in Java.

The information gathering agent used in this example is based on a template containing a generic co-operative agent model (Brazier et al., 1996). Figure 3 illustrates the seven processes distinguished in this generic model. This architecture models an agent that:

- reasons about its own processes (component Own Process Control, or opc),
- communicates with other agents (component Agent Interaction Management, or aim),
- maintains information about other agents (component Maintenance of Agent Information, or mai),
- interacts with the external world (component World Interaction Management, or wim),
- maintains information about the external world (component Maintenance of World Information, or mwi),
- participates in project co-ordination (component Co-operation Management, or cm) and
- the agent's specific tasks (component Agent Specific Tasks, or ast).

This model of a co-operative agent includes components for management of its own processes, interaction with other agents including co-operation, interaction with the external (material) world, and an agent's more specific tasks. In this model, a co-operative agent receives messages from other agents, and observations in the external world (its input). It sends messages to other agents and directs its own observations and actions in the external world (its output).

In this section, two examples are given of adaptation of the information gathering agent. In the first example, the information gathering agent is adapted to include functionality for understanding a new language (Section 4.1). In the second example, the information gathering agent is adapted to include new functionality for (more) secure communications and co-operations (Section 4.2).

4.1 Shallow adaptation

Figure 4 depicts the information gathering agent and shows that both the agent interaction management process (aim) and the co-operation management process (cm) are open slots. The open slot of the agent interaction management process is filled by two components providing functionality for understanding a communication language with the personal assistant agent and an information provider agent (e.g., which provides access to the World-Wide Web). The open slot of the co-operation management process is filled by two components providing functionality for understanding how to co-operate (protocols) with the personal assistant agent and an information provider agent.

One of the needs for adaptation identified in Section 2 was that the information gathering agent needed to interact with the travel agent. The travel agent was able to indicate that a specific language and protocol was to be employed. One way for the information gathering agent to approach this problem is to use the agent factory to have itself

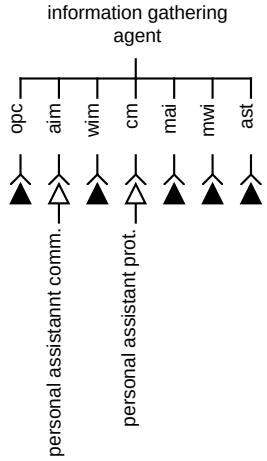


Figure 4: Partial description of the information gathering agent. The open slots for the agent interaction management and co-operation management processes are filled with two components each: agent communication languages and protocols.

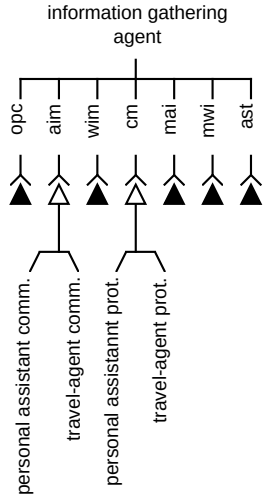


Figure 5: Partial description of the information gathering agent. The open slots for the agent interaction management and co-operation management processes are filled with two components each: agent communication languages and protocols.

changed, such that it can understand the languages and protocols needed for interaction with the travel agent.

In the first case, the agent factory searches its libraries of templates and components and is able to find components that support the functionality required. In addition, these components contain descriptions at both levels of abstraction, and each description needs to be linked to the existing description of the information gathering agent. This results in a description of the information gathering agent, as depicted in figure 5.

The information gathering agent is adapted to include functionality on a language and protocol for interaction with the travel agent.

First, the new components are inserted into the conceptual description of the agent. Once this has been achieved successfully, the operational parts of the components are inserted into the operational description of the agent.

4.2 Deep adaptation

In another example in Section 2, one of the needs for adaptation arises from communication with a bank agent. This agent requires that the information gathering agent uses specific security functionality. Again, the information gathering agent uses the agent factory to have itself adapted.

The agent factory now has two goals in adapting the information gathering agent. Specific security functionality needs to be added, and functionality for understanding a language and protocol shared with the bank agent. The latter case has been described in the previous subsection.

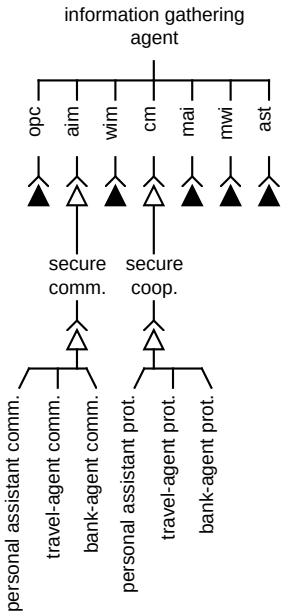


Figure 6: The information gathering agent is adapted to include functionality on secure communication and co-operation, and functionality on understanding a language and protocol for interaction with the bank agent.

Adapting the information gathering agent to include specific security functionality is translated by the agent factory to the need to adapt the agent to include functionality for secure communication and secure co-operation (as in both processes security-related awareness is needed). Two conceptual templates have been retrieved from the library available to the agent factory: a template for secure communication and a template for secure co-operation. Both templates can be used together, as can be derived from their characteristics, and both templates can be embedded in the current configuration of templates and components. Detailed templates are available for these conceptual templates, which can also interface with detailed templates and components in the current detailed configuration of the information gathering agent.

As shown in figure 6, the information gathering agent is modified in a non-trivial manner: the two new templates are inserted into two of the open slots of the top-most template, and the original fillings of these open slots are inserted into the open slots of the new templates.

5 Feasibility

The feasibility of an agent factory hinges on a number of aspects. These aspects are briefly described in Section 5.1. A comparison of the agent factory to other approaches in constructing agents in described in Section 5.2.

5.1 Crucial aspects

A number of aspects are crucial to the feasibility of an agent factory. These aspects are mainly related to building blocks; the templates and components. Inserting templates or components into an open slot of a template or component involves understanding:

- the properties associated with the interface required by an open slot, which prescribe properties of the interfaces of entities to be inserted,
- how properties relate to each other,
- how a description of the template or component to-be-inserted can be connected to an open slot (this may involve a mapping of interfaces expressed in two different conceptual description languages),
- how multiple components can be inserted into one open slot (cf. stacking blocks), especially when different description languages are employed.

Experience with the current prototype has increased confidence in the feasibility of the agent factory. This prototype is capable of automatically configuring relatively simple information retrieval agents from a limited set of building blocks. An agent can be constructed and/or adapted, on the basis of a description of required functionality. This prototype uses a framework for describing conceptual descriptions based on DESIRE (Brazier et al., 1998) (simplified) and operational descriptions based on the programming language Java. The performance of the current prototype is limited in both functionality and resource usage. Current and future research focusses more on improving the functionality of the agent factory than reducing its resource usage.

More research is needed (and is being conducted) to, for example, develop ontologies for building blocks, extend the library with building blocks for other types of agents, and assess genericity and specificity of (descriptions of) building blocks. The use of

additional frameworks (such as UML) and languages (such as C) is also being pursued. Current research includes development of a language to describe blueprints (including the configuration of building blocks).

An application area in which the agent factory can play an important role is *generative migration* (Brazier et al., 2002). In most of today's agent systems migration of an agent requires homogeneity in the programming language and/or agent platform in which an agent has been designed. The agent factory supports generative migration: agents can migrate between non-identical platforms and need not be written in the same language. Instead of migrating the 'code' (including data and state) of an agent, a blueprint of the agent is transferred. An agent factory generates new code on the basis of this blueprint. A prototype is currently being developed as a service of agent-oriented middleware: AgentScape.

5.2 Comparison to other approaches

The agent factory is in some ways comparable to component-based development, agent construction kits, software reusability, case-based reasoning, configuration design, and IBROW.

The agent factory's approach to combining templates and components seems similar to the approach taken in *component-based development* of software (Hopkins, 2000; Sparling, 2000). One distinction is that our approach includes annotations of templates and components at two levels of abstraction (conceptual and operational). In component-based development, interfaces are described for components (which are independent of an operational language); this corresponds to the descriptions of interfaces of templates and components and interfaces needed by open slots in templates and components. From our perspective, component-based development provides a useful means to describe operational descriptions of the building blocks used by the agent factory.

Currently, a relatively large number of tools and/or frameworks exists for the (usually semi-automatic) *creation of agents* (not automated adaptation). Examples include e.g. AgentBuilder (Reticular, 1999), D'agents/AgentTCL (Gray et al., 1997), ZEUS (Nwana et al., 1999), NOMADS (Suri et al., 2000), Sensible Agents (Barber et al., 2001), and Tryllian (Tryllian, 2001). All of these approaches commit to a specific operational description of agents, and in some cases also commit to a specific conceptual description of their agents. The agent factory does not make such commitments, making the agent factory more general purpose (with all the common advantages and disadvantages).

The agent factory currently pragmatically circumvents a number of issues related to *software reusability* (e.g., (Biggerstaff and Perlis, 1997)). A major problem is annotating reusable pieces of software such that they can be retrieved at a later time (by other people) and reused with a minimal number of changes. In the agent factory, the latter is endeavoured as well. The former is currently solved in a pragmatic way: templates and components are annotated, and, when needed, a mapping is provided to other annotations. This, however, is not a scalable solution, and, as such, is one of our current foci of research. An important decision concerning standardisation is that the agent factory does not aim to adhere to one specific standard, but a number of standards.

In *case-based reasoning* approaches (e.g., (Kolodner, 1993; Watson and Marir, 1994)), libraries of cases are consulted to find a case which matches a problem, upon which retrieved cases are adapted. This approach differs from the agent factory in that cases are modified internally, instead of combined with other cases. Techniques for retrieving cases from case libraries are, of course, relevant to retrieving templates and components from libraries.

The approaches taken by *design-as-configuration* (e.g., as described in (Stefik, 1995), CommonKads (Schreiber et al., 1999), and elevator configuration (Schreiber and Birmingham, 1996)) focus on constructing a satisfactory configuration of elements on the basis of a given set of requirements (also named: constraints). In most of these approaches, no explicit manipulation of requirements is present, nor is a multi-levelled description of the elements taken into account. Models and theories on configuration-based design are relevant to the agent factory, in particular to the processes involved in combining conceptual and operational descriptions.

The approach taken is similar, to some extent, to approaches such as IBROW (Motta et al., 1999). In IBROW, semi-automatic configuration is supported of intelligent problem solvers. Their building blocks are 'reusable components', which are not statically configured, but dynamically 'linked' together by modelling each building block as a CORBA object. The CORBA-object provides a wrapper for the actual implementation of a reusable component. A Unified Problem-solving method development language UPML (Fensel et al., 2001) has been proposed for the conceptual modelling of their building blocks. The agent factory differs in a number of aspects, which include: multiple conceptual and detailed languages, no pre-defined wrappers for detailed building blocks, agents consist of one process, and the process of reconfiguration is an automated (re-)design process.

6 Discussion

An automated servicing process for agent adaptation is described in this paper. This servicing process is the task of an agent factory. Agents are constructed from templates and components. Adapting an agent entails adapting the configuration of templates and components.

Five assumptions underly our approach: (1) agents have a compositional structure, (2) re-usable parts of agents can be identified, (3) two levels of descriptions are used: conceptual and operational, (4) properties and knowledge of properties and interfaces of re-usable parts of agents are available, and (5) no commitments are made to specific languages and/or ontologies.

The main advantage of an agent factory as an automated servicing process is that an agent can easily obtain new functionality, without obliging the agent itself to have its own adaptation mechanism. During their lifetime, agents acquire new skills and knowledge.

The agent factory is still being researched; the current research focusses on:

- building a library of templates and components,
- designing description languages for properties of interfaces of, and knowledge about the use of, templates and components,
- learning from experiences with different conceptual and operational description languages,
- designing and implementing more extensive prototypes of the agent factory,
- investigating security and trust in using an agent factory.

Acknowledgements

The authors wish to thank the graduate students Hidde Boonstra and David Mobach for their explorative work on the application of an agent factory for an information retrieving agent. This work was supported by NLnet Foundation, http://www.nlnet.nl/.

References

Barber, K., McKay, R., MacMahon, M., Martin, C., Lam, D., Goel, A., Han, D., and Kim, J. (2001). Sensible agents: An implemented multi-agent system and testbed. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, pages 92–99.

Benjamins, V. (1995). Problem-solving methods for diagnosis and their role in knowledge acquisition. *International Journal of Expert Systems: Research & Applications*, 8(2):93–120.

Biggerstaff, T. and Perlis, A., editors (1997). *Software Reusability: Concepts and models*, volume 1. New York, ACM Press.

Booch, G. (1991). *Object oriented design with applications*. Redwood City, Benjamins Cummins Publishing Company.

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). Simple object access protocol (soap) 1.1. Technical report, W3C. http://www.w3.org/TR/SOAP/.

Bray, T., Paoli, J., Sperberg-McQueen, C., and Maler, E. (2000). Extensible markup language (xml) 1.0 2nd ed. Technical Report 20001006, W3C. http://www.w3.org/TR/2000/REC-xml-20001006.

Brazier, F., Jonker, C., and Treur, J. (1998). Principles of compositional multi-agent system development. In Cuena, J., editor, *Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98*, pages 347–360.

Brazier, F., Jonker, C., and Treur, J. (2000a). Compositional design and reuse of a generic agent model. *Applied Artificial Intelligence Journal*, 14:491–538.

Brazier, F., Jonker, C., Treur, J., and Wijngaards, N. (2000b). Deliberate evolution in multi-agent systems. In Gero, J., editor, *Proceedings of the Sixth International Conference on AI in Design, AID'2000*, pages 633–650. Kluwer Academic Publishers.

Brazier, F., Jonker, J., and Treur, J. (1996). Modelling project coordination in a multi-agent framework. In *Proc. Fifth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE'96*, pages 148–155. Los Alamitos, IEEE Computer Society Press.

Brazier, F., Overeinder, B., van Steen, M., and Wijngaards, N. (2002). Agent factory: Generative migration of mobile agents in heterogeneous environments. In *Proceedings of the AIMS workshop at SAC-2002*. to appear.

Bui, H., Kieronska, D., and Venkatesh, S. (1996). Learning other agents' preferences in multiagent negotiation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-96)*, pages 114–119.

Fensel, D., Horrocks, I., van Harmelen, F., Decker, S., Erdmann, M., and Klein, M. (2000). Oil in a nutshell. In Dieng, R., editor, *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modelling, and Management (EKAW'00)*, volume 1937 of *Lecture Notes in Artificial Intelligence*, pages 1–16. Springer-Verlag.

Fensel, D., Motta, E., Benjamins, V., Crubezy, M., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., van Harmelen, F., Musen, M., Plaza, E., Schreiber, A., Studer, R., and Wielinga, B. (2001). The unified problem-solving method development language UPML. *Knowledge and Information Systems*. to appear.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of reusable object-oriented software*. Addison Wesley Longman, Reading, Massachusetts.

Gray, R., Kotz, D., Cybenko, G., and Rus, D. (1997). *Agent Tcl*, chapter 4, pages 58–95. Manning Publishing. W. Cockayne and M. Zyda, editor.

Hopkins, J. (2000). Component primer. *Communications of the ACM*, 43(10):27–30.

Horrocks, I., van Harmelen, F., Patel-Schneider, P., Berners-Lee, T., Brickley, D., Connoly, D., Dean, M., Decker, S., Fensel, D., Hayes, P., Heflin, J., Hendler, J., Lassila, O., McGuinness, D., and Stein, L. (2001). Daml+oil. Technical report, DAML. http://www.daml.org/2001/03/daml+oil-index.html.

Jackson, M. (1975). *Principles of Program Design*. Academic Press.

Jackson, M. (1995). *Software Requirements and Specifications*. Addison-Wesley, Wokingham, England.

Kolodner, J. (1993). *Case-Based Reasoning*. Morgan Kauffman, San Mateo, California.

Motta, E., Fensel, D., Gaspari, M., and Benjamins, V. (1999). Specifications of knowledge component reuse. In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE-99)*, pages 17–19, Kaiserslautern, Germany.

Nwana, H., Ndumu, D., Lyndon, L., and Collis, J. (1999). Zeus: A toolkit and approach for building distributed multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Autonomous Agents'99)*, pages 360–361.

Peña-Mora, F. and Vadhavkar, S. (1996). Design rationale and design patterns in reusable software design. In Gero, J. and Sudweeks, F., editors, *Artificial Intelligence in Design (AID'96)*, pages 251–268, Dordrecht, The Netherlands. Kluwer Academic Publishers.

Pressman, R. (1997). *Software Engineering: A practitioner's approach*. Computer Science. McGraw-Hill, fourth edition.

Reticular (1999). *AgentBuilder: An integrated toolkit for constructing intelligent software agents*. Reticular Systems Inc, white paper edition. http://www.agentbuilder.com.

Riel, A. (1996). *Object-Oriented Design Heuristics*. Addison Wesley Publishing Company, Reading Massechusetts.

Rumbaugh, J., Jacobson, I., and Booch, G. (1999). *The unified modeling language reference manual*. Addison Wesley, Reading, Massachusetts.

Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W. V., and Wielinga, B. (1999). *Knowledge Engineering and Management, the CommonKADS Methodology*. MIT press.

Schreiber, G. and Birmingham, W. (1996). Special issue on sisyphus-vt. *International Journal of Human-Computer Studies (IJHCS)*, 44. editors.

Soltysiak, S. and Crabtree, B. (1998). Knowing me, knowing you: Practical issues in the personalisation of agent technology. In *Proceedings of the third international conference on the practical applications of intelligent agents and multi-agent technology (PAAM98)*, pages 467–484, London.

Sparling, M. (2000). Lessons learned through six years of component-based development. *Communications of the ACM*, 43(10):47–53.

Stefik, M. (1995). *Introduction to Knowledge Systems*. Morgan Kaufmann Publishers Inc., San Francisco, California.

Suri, N., Bradshaw, J., Breedy, M., Groth, P., Hill, G., Jeffers, R., Mitrovich, T., Pouliot, B., and Smith, D. (2000). Nomads: Toward a strong and safe mobile agent system. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 163–164. ACM Press.

Tryllian (2001). Agent development kit. Technical report, Tryllian. http://www.tryllian.com/sub_documentation/whitepapers/english/Technical white paper ADK v1.0.pdf.

Watson, I. and Marir, F. (1994). Case-based reasoning: a review. *The Knowledge Engineering Review*, 9(4):327–354.

Wells, N. and Wolfers, J. (2000). Finance with a personalized touch. *Communications of the ACM, Special Issue on Personalization*, 43(8):31–34.

Wieringa, R. (1996). *Requirements Engineering: Frameworks for Understanding*. Wiley and Sons.

Wong, H.-C. and Sycara, K. (2000). A taxonomy of middle-agents for the internet. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS'2000)*, pages 465–466.

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

BDI Agents and Constraint Logic

Stuart Chalmers and Peter M. D. Gray

Department of Computing Science, University of Aberdeen
King's College, Aberdeen, AB24 3UE, United Kingdom
schalmer@csd.abdn.ac.uk ; pgray@csd.abdn.ac.uk

Abstract

Using Constraint Logic Programming and a data model approach, we provide an agent with a flexible way to plan and direct its actions and to manipulate and represent its knowledge. We use constraint solving techniques developed in the KRAFT project to provide the agent with a sophisticated reasoning and deliberation process. We explore the declarative use of constraints within a BDI Agent framework to represent knowledge as complex quantified constraints and apply these techniques to a courier scenario where cooperating agents communicate, delegate and exchange desires and information using Generalised Partial Global Planning mechanisms to solve a given set of tasks.

1 Introduction

Multi-agent systems are emerging as an important software model for next generation computing problems that deal with distributed tasks in dynamic, heterogeneous environments. The ability to cooperate and share knowledge is essential in an environment where agents are executing, scheduling and exchanging tasks, whilst simultaneously judging the impact that non-local events and new information have on their decision making process.

An agent operating in such an environment must be able to exchange and delegate tasks in order to satisfy deadlines and resource constraints. To this end, they must be able to reason rationally about their current capabilities and status, as well as those of other agents. Beliefs, Desires, Intentions (BDI) agents (Rao and Georgeff, 1995) use three intentional attitudes to model an agent's internal state and provide it with such a reasoning process.

Of particular interest is the performance of such agents in highly dynamic environments where preset, deterministic plans may become redundant by the time of execution. BDI agents provide a way of specifying plans at such a high level as to leave the specific implementation to the agent at run-time. This is very different from Object Oriented Programming systems where messages to objects invoke procedural methods which lead to foreseen changes of state.

The way in which the agent collates and uses the information available to it will affect its choice of plan. What is needed is a way of making all possible information available to the agent, so that it has enough autonomy to make the most appropriate decision given its surroundings, its current beliefs, and its current status.

1.1 Representing Information using Constraints

Traditionally, constraints have been used as a representation of restrictions relating to a set of data values. More recently constraints have been looked at as a representation for knowledge in a distributed agent-based environment (Eaton et al., 1998; Gray et al., 1999b). Our research is concerned with using the declarative nature of constraints within a *BDI* Agent framework which expresses an agent's *Desires* as complex quantified constraints. Thus the *Intentions* will become plans to achieve them, depending on current *Beliefs*.

Constraints are used to model a description of part of a *desired* goal state which the agent is free to achieve in any number of different ways. In doing so, the agent can take account of other constraints it has undertaken to satisfy by combining all the information into a CLP (Constraint Logic Programming) problem. This representation means that the agent is not just responding to a sequence of messages, one at a time as in Object Oriented programming; instead it is able to *plan* its behaviour by taking into account both the message and other constraints. Further, if the desired state is impossible to achieve (over constrained) because of too many different desires, it has to relax some constraints, by delegating or exchanging tasks with other agents, which is an important aspect of agent behaviour in multi-agent systems. The constraints thus become mobile (Gray et al., 1999a).

The constraints can refer to the configuration in space of a number of objects, constrained by various relationships and inequalities described in predicate logic. This has been successfully used in the KRAFT project (Preece et al., 1999), but there the agents were 'information seeking', extracting the constraint information from databases and returning them to a central planner. Here the agents are themselves capable of planning and interacting with other agents and are using temporal, rather than spatial constraints.

The great virtue of constraints is that they remove unnecessary procedurality from specifications. Thus it is much easier to combine constraint information from different sources when it is expressed as First Order Logic clauses than when it appears as independently constructed pieces of code. We can transform constraints to use a local ontology or data model as a common basis (Hui and Gray, 2000). In this form, we can easily treat it as a CLP and thus provide a decision making process which can take into account many disparate factors, which is vital when providing autonomy in a highly dynamic environment.

1.2 Structure of the Paper

We start by looking at an E-commerce scenario which requires agent autonomy, using couriers. Sections 3 and 4 show how these agents work as BDI agents, and how they can reason using a CLP solver. Section 5 discusses the agent coordination model and how constraints are exchanged, and tasks delegated. The final sections discuss implementation details, evaluation and closely related work before concluding.

2 The Design of an E-commerce Scenario

2.1 Test Environment

We have designed a group of cooperating agents using a constraint based architecture which operate as couriers in a map-like environment (Fig. 1).

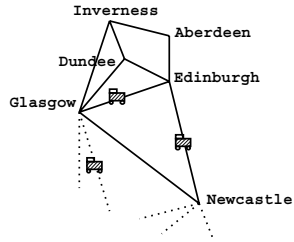


Figure 1: *Courier Agents in a Test Environment*

Each courier is given a set of tasks to complete by a specific deadline and they must each individually try to construct and execute a schedule which will allow them to complete their given tasks by their respective deadlines. The agents have to take into account the job of collecting items from customers and delivering them before a set time at a specified location. They must be able to take on and reorganise jobs, and be able to take into account road closures, flight delays, traffic jams, etc. Couriers must be able to cooperate with each other if deadlines cannot be met, or if they are unable to get all parcels delivered due to the time or distance involved. In these situations, they must be able to delegate and redistribute the tasks amongst other couriers who are more capable of completing the task given the constraints imposed. We wish to show that by using a constraint based deliberation process for our agents, and by representing each agent's desires as constraints, we can provide a flexible (though not always optimal) way of giving the agent enough autonomy to deal with unexpected situations and with incomplete information in such an environment, whilst maintaining desired goals.

2.2 The Courier Scenario

A courier agent starting at 7am in Glasgow has 3 parcels to deliver, each by a specific deadline:

- `parcel1` : Dundee $\rightarrow$ Glasgow, Deadline: 5pm, Payoff: 10
- `parcel2` : Edinburgh $\rightarrow$ Inverness, Deadline: 12pm, Payoff: 7
- `parcel3` : Glasgow $\rightarrow$ London, Deadline: 11pm, payoff: 8

Each parcel has an associated payoff, which is the amount the agent will get upon delivery of each parcel by its specific deadline. Using a simple scheduling algorithm, the agent decides that it can, based on the initial information it knows, deliver these 3 parcels to their destinations by the given times. It builds up the initial schedule shown on the left in Fig. 2 (deliver `parcel2` by 10am, `parcel1` by 3pm and `parcel3` by 11pm).

The courier agent takes this schedule information and begins by building plans for the delivery of `parcel2`. It finds a plan that takes the route `Glasgow`$\rightarrow$`Edinburgh` $\rightarrow$`Dundee` $\rightarrow$ `Inverness`, which it begins executing by travelling towards Edinburgh.

During the journey, it receives information that the road is blocked between Edinburgh and Dundee. By looking at its possible options for delivering this parcel, it finds another route, `Glasgow`$\rightarrow$`Edinburgh` $\rightarrow$`Aberdeen` $\rightarrow$ `Inverness`. This journey will take longer to complete, so the agent cannot deliver the parcel by 10am as expected, but still by the deadline, 12pm.

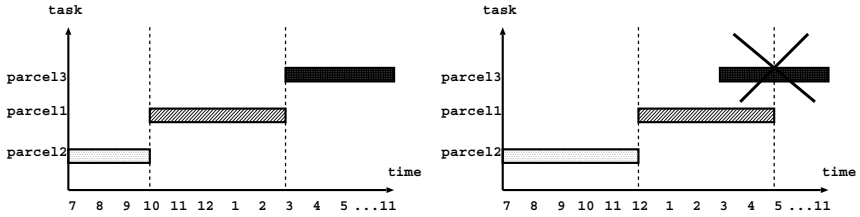


Figure 2: *Schedule before and after BDI loop information*

It reschedules its other tasks accordingly with this new information, and finds that it can only deliver one of the remaining parcels, `parcel1` by the deadline (5pm) or `parcel3` by its deadline (11pm). Since `parcel1` has a greater payoff, it decides to try and exchange or delegate the other delivery, `parcel3` (right diagram in Fig. 2).

The Agent broadcasts the task of delivering `parcel3` to others in the environment to ask if there is anyone who can exchange this task, or have it delegated to them. Another agent operating in the same environment, `AgentB`, broadcasts that it is willing to exchange one of its parcel deliveries, `parcel12`, that involves taking a parcel from Glasgow to Newcastle by 7pm, as it cannot complete this task given its own constraints. `AgentA` builds a preliminary schedule with `parcel12`, along with `parcel1` and `parcel2`, and finds that it can incorporate this new task in with its existing ones. `parcel12` has a payoff of 5, which is less than that of `parcel3`, but still means that the agent will receive some benefit from delivering it. An exchange of tasks takes place and the agent continues on with its planning and execution for delivering `parcel2`.

Our research represents each of these tasks as constraints, and explores the decision making processes that the agent takes when deciding on which plan to implement for each task, and more importantly, the decision making process which leads to the specific choice of plan implementation. By modelling this decision making process as a CLP, we can show that this representation is effective in situations where many different disparate sources of knowledge need to be considered when reasoning, and that this representation provides an effective way of providing the agent with enough autonomy to deal with unexpected events and occurrences as and when they happen.

3 The BDI Agent Architecture

3.1 The Agent Architecture

The agent architecture is split into 2 parts (Fig. 3), the coordination/scheduling layer, which interacts with other agents and schedules multiple tasks, and the BDI layer, based on the Beliefs, Desires, Intentions model of a deliberative system (Rao and Georgeff, 1995). We have tried, in the design of this architecture, to allow as much scope to use the agent deliberation process and BDI layer (and its separate parts) in different scenarios with the minimal amount of change.

The reason for this layered approach is to make the agent coordination and scheduling mechanisms independent of the constraint based BDI agent implementation we have constructed. By separating the CLP deliberation process from the coordination/scheduling layer, we show that it is not restricted to one specific coordination or scheduling model and can be re-used within existing agent communities with minimal alterations.

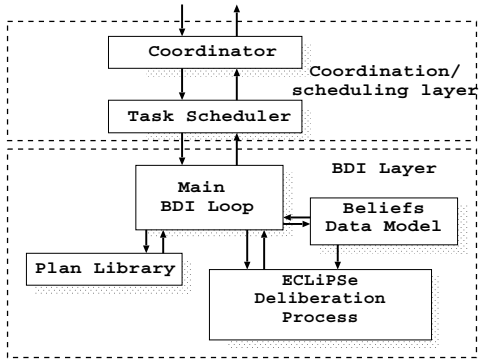


Figure 3: *Agent architecture.*

3.2 Interacting Layers

The coordination/scheduling layer in each individual agent manages the agent's tasks at a high level and coordinates the agent's communication and interaction with others while the BDI layer constructs and executes plans for individual tasks and manages the agent's interaction with the environment.

Information is passed between the two layers as the agent progresses in its task scheduling and execution. The coordination and scheduling layer gives the BDI layer the current task to be completed. The BDI layer chooses and executes an action or subplan accordingly.

Information flow in the opposite direction provides details about the execution of the current plan (such as what action/subplan the agent has committed to, current completion time estimates for the task). Using this information, the coordination/scheduling layer can build up a more detailed schedule which will take into account new information gathered as the agent progresses (as seen in the example in section 2.2).

This information passed to the scheduler is comparable to that used in (Clement and Durfee, 2000). In their system, the planner constructs *summary* information by instantiating abstract plans and finding possible conflicts at lower subplan levels. This information is then used to coordinate multiple agents' actions.

3.3 The Main BDI Loop

```
Do
     options := option_generator(event_queue,B,D,I)
     selected-options := deliberate(options,B,D,I)
     update-intentions(selected-options,I)
     execute(I)
     get-new-external-events();
     drop-succesful-attitudes(B,D,I)
until quit
```

Figure 4: *Main BDI loop*

Fig. 4 shows a pseudo-code version of a BDI agent algorithm, which forms the basis for the design of the BDI layer. This algorithm controls the agents planning and its in-

teraction with the environment and operates on the main data structures in the BDI layer agent architecture.

The BDI algorithm is based on the idea of a deliberative system, which uses a symbolic representation of what it perceives the current state of the world to be. This representation can be used to reason logically about the system's views and its actions (Genesereth and Nilsson, 1987).

In a BDI loop, the agent receives events internally from the desires (in this case packages to deliver) and externally (information about road closures, delays etc.) from the environment, and uses these, along with its current Beliefs (B), Desires (D) and Intentions (I) to generate a set of possible next options represented as executable plans. It then selects one of these possible options based on the status of its B, D and I values. This is represented by our constraint mechanism, which takes the Beliefs and maps them to finite domains in a constraint problem. It then uses the Desires and Intentions (as constraints) in the constraint solving mechanism to choose and then execute the best plan from those available. Finally, any new external events (caused by the execution of the plan or otherwise) are added to the event queue and the agent checks to see whether it can mark as completed any of its goals that have been achieved, before carrying on.

BDI agent behaviour can be explained in terms of human-like attitudes (Intentions). These attitudes provide a level of abstraction that is easy to understand and can be represented symbolically in a deliberative system. These attitudes can be grouped into 2 distinct types (Wooldridge and Jennings, 1995) which, when combined provide the basis of the agents deliberative system :

- **What you know** (*information attitudes*): belief, knowledge etc.
- **Reasons for actions** (*pro-attitudes*): desires, intention, obligation, etc.

BDI agents use a combination of information and pro-attitudes, Beliefs, Desires and Intentions (Rao and Georgeff, 1995; Bratman, 1987; Bratman et al., 1999; Cohen and Levesque, 1990). Various views and interpretations of the meaning of each attitude exist, but the following definitions are generally accepted:

- **Beliefs** represent the views held about the current state of the world, as perceived by the agent.
- **Desires** are the goals and aims you wish your agent to achieve.
- **Intentions** are the actual processes being executed so that desires may be fulfilled (or sub-goals which are needed for the completion of a desire).

We have constructed a planning and execution model for our courier scenario based upon this BDI architecture and the algorithm in Fig. 4. It receives the task of delivering a parcel from the coordination/scheduling layer and chooses and executes a plan to complete this task.

The desires of our agents are the tasks that it is given to complete (in the courier scenario this is the parcels it has to deliver). We formulate these as constraints specifying a *desired state*, or how we want the agent's beliefs about the state of the environment to be (what parcels are where). The process of choosing and executing a succession of plans gradually alters the agent's beliefs about the state of the environment (and the environment itself) until the constraints representing the desires are satisfied.

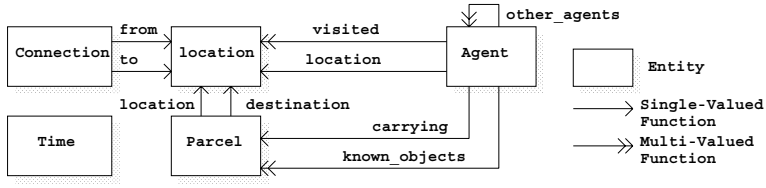


Figure 5: *Agent's Belief data model*

3.4 Representing Beliefs, Desires and Intentions

We use a data model representation for the agent's beliefs (Fig. 5). This holds information such as the current status of the environment and the status of the agent itself. The entities (Connection, Square, etc.) hold the actual data, while the functions represent the relationships between the entities (e.g. the connections to and from a location). Our data model representation of the agent's beliefs is constructed using P/FDM[1], an object database constructed on semantic data modelling principles from the functional Data Model (Shipman, 1981). This database is constantly updated to reflect the changes to the agent and the environment as and when they happen. The advantage of this approach is that the data model can be used to express the semantics of the agent's beliefs as well as the beliefs themselves, and can express complex relationships between objects and data values in those beliefs.

There is an existing P/FDM constraint language, COLAN (Bassiliades and Gray, 1994) (based on Shipman's DAPLEX language), that can be used to express the desires in terms of the beliefs. Initially COLAN was used for specifying database integrity constraints, but has been shown to be a powerful and expressive language for representing and describing knowledge. The following shows an example task in COLAN, describing the delivery of a parcel to Glasgow by 7pm along with its First Order Logic equivalent.

```
Constrain each p in parcel
such that name(p) = parcel1
and time(p) = the t in time
such that current_time(t) = 1900
to have location(p) = glasgow

(∀p,t) parcel(p) & name(parcel1,p) & time(p,t)
& current_time(t,1900) → location(p,glasgow).
```

There has been extensive work carried out on integrating P/FDM data model frameworks into finite domain CLP problems in the KRAFT project (Hui and Gray, 2000) and in using First Order Logic to express and generate CLP code in terms of this data model. Our techniques for transforming the information from the desires and beliefs into a finite domain constraint problem are largely based on this work, but in KRAFT, constraints are used as problem specifications to find solutions to complex distributed design problems.

[1] For further details, see http://www.csd.abdn.ac.uk/∼pfdm

4 The Agent Deliberation Process as a Constraint Problem

The agent's BDI layer generates possible courses of action as a set of plans, then deliberates on and chooses the most suitable one. This choice is dependent on the agent's deliberation process, and the information available to this process at that time. We have formulated this deliberation process as a CLP, using the agent's beliefs as the basis for the construction of the domains in the CLP and the agent's desires as the constraints on those domains.
During the deliberation process, the agent must be able to choose its next course of action. It must deliberate on a number of possible plans available to it and decide on the most appropriate, based on a number of factors. These include:

- The current status of the task being executed.
- The agent's beliefs about the current environment.
- The deadline to be met for the task currently being executed.

In the courier scenario, these factors include the current parcel or message the agent is delivering, the agent's knowledge about road delays, traffic jams etc., the deadlines that the parcels or messages must be delivered by and what other parcel deadlines the agent has to meet. Building on existing finite domain formulation techniques we have developed a way of combining these disparate information sources by constructing them as a CLP and then using this as the agent's decision making process.

4.1 Constructing Possible Worlds

The agent's beliefs about the state of the environment, the tasks the agent has to do (its desires) and its current task status (intentions) are used as a basis for the deliberation process (from Fig. 4, the B,D and I variables in the deliberate function of the BDI algorithm).

The beliefs are a reflection of how the agent views the current state of the world so they naturally form the basis for the agent's deliberation process. Since we are using a data model approach to hold the agent's beliefs, and to provide the semantics to describe the domain, we formulate agent desires in terms of the entities, attributes and relationships contained in the data model and use these desires as specifications of the required tasks.

The advantage of using this data model approach is that each entity class in the data model can be easily viewed as a finite domain, with the object instances themselves as the elements in that domain. The object attributes and functions can then be used to form the basis for variables in the constraints.

Our agent's deliberation process needs to take into account possible future actions and events, as well as the current state. It must be able to judge how the current action will affect future events, and how the current choice of action will affect the decisions it has to make in the future. Given that the current environment is constantly changing, we are unable to definitely predict future events, but we *can* plan the agent's actions to a specified depth of lookahead and then refine and replan as we encounter changes.

Although the environment changes, the agent still maintains long term, high level desires which will remain unaffected (e.g. a parcel must be delivered by a deadline, but the actual method of delivery is not specific). What *will* be affected by the changes is the way in which these high level desires are carried out. What we are using the CLP process to do is to allow the agent to exercise a degree of autonomy in carrying out the high level

desires it has. We provide it with the various implementations of each high level task and give the agent an intelligent method for choosing the most appropriate implementation.

When the agent begins its deliberation process it first constructs a possible worlds model (Rao and Georgeff, 1995) which contains all possible solutions and the interim states needed to get to those solutions in a connected tree like structure. To populate the tree with candidate solutions, we find all the actions possible from the current state, then apply these actions and add these as new branches to the tree. Therefore, if the agent is currently at time *t*, that node of the tree will lead to x_1 possible states the agents beliefs could have at time *t+1* if a specific action towards the completion of a plan was carried out.

We then take these x_1 states and find the actions available from each of them. We can then apply these, and add the resulting new states to the tree (x_2 solutions at time *t+2*). This process can be continued to a given depth (i.e. how far you want your agent to plan ahead).

The resulting populated structure contains all the possible states of the agents beliefs for the next *n* specified steps (what states the agent's belief data model would take were it to choose a specific way of carrying out all possible ways of doing these *n* steps towards completing the high level task). We therefore have a possible worlds tree like structure which preserves the information about the hierarchical position of each state in relation to the others, with each state containing the agent's beliefs, if the action leading to the desired goal state were to be carried out.

4.2 From Possible Worlds to CLP

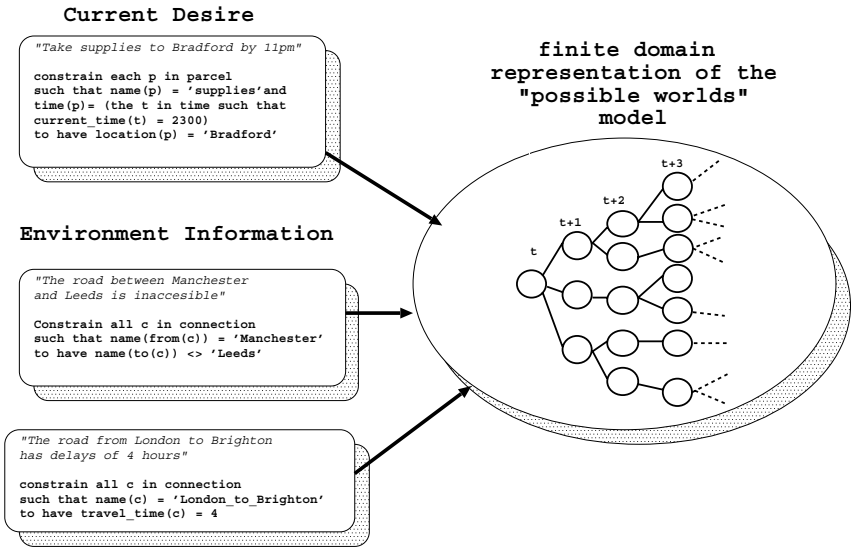


Figure 6: *The Agent's Deliberation Process as a CLP*

To construct the deliberation process as a CLP, we transform this structure into finite domains in ECLiPSe (ECRC and IC-Parc, 2000). The constraints are then posted against these finite domains and any invalid states are eliminated. Fig. 6 shows an example of the constraints that can be posted in the CLP. The constraints come from various sources, but can all be combined to influence the agent's reasoning process.

Each node in the possible worlds structure holds a populated data model (Fig. 5), representing the agent's beliefs for that particular state. The code fragment below shows part of the P/FDM data representation for a small part of one of these states. The entities and their values in the data model are represented using the `object/3` construct, and the functions and their relationships to other objects are shown by the `fnval/3` construct. The representation of the objects and the object functions as term structures gives us a uniform representation of the agent's beliefs, and allows for easy modification given any additions or changes to the data model representation.

```
object(agent,agent1).
fnval(agent_id,agent1,ozzy).
fnval(agent_time,agent1,1800).
fnval(carrying,agent1,parcel1).
fnval(agent_location,agent1,location12).
fnval(visited,agent1,location3).
fnval(visited,agent1,location2).

object(location,location12).
fnval(location_name,location12,glasgow).

object(parcel,parcel1).
fnval(name,parcel1,supplies).
```

4.3 Solving the CLP using the ECLiPSe Propia Library

The ECLiPSe *Propia* library (ECRC and IC-Parc, 2000) supports *generalised constraint propagation*. Using the `infers most` operator we can specify the finite domains for the CLP problem using the following ECLiPSe goals (see section 6):

```
object(agent,agent_id,ID) infers most.
fnval(agent_id,ID,AGENT_NAME) infers most.
fnval(agent_time,ID,AGENT_TIME) infers most.
fnval(carrying,ID,PARCEL_ID) infers most.
fnval(agent_location,ID,LOCATION_ID) infers most.
fnval(visited,ID,LOCATION_ID) infers most.

object(square,LOCATION_ID) infers most.
fnval(location_name,LOCATION_ID,LOC_NAME) infers most.

object(parcel,PARCEL_ID) infers most.
fnval(name,PARCEL_ID,PARCEL_NAME) infers most.
```

This makes the possible values of each attribute of each object (as well as the variables representing the objects themselves) into a finite domain CLP variable. For example, the above code fragment would create ten finite domains, one for each object and one for each `fnval` construct (so `LOC_NAME` from the above `fnval` would represent the finite domain containing all location names from every state in every node in the tree that contains similar constructs). We can now post constraints from our various sources over these domains. Any elements which are eliminated from the solution space by the constraints will propagate through the rest of the domains, eliminating all belief states which do not

satisfy the required constraints. The remaining belief states are the valid options available to the agent in the possible worlds model. What is left therefore is a pruned version of the possible worlds tree structure containing the remaining valid states given the constraints imposed.

From the remaining tree structure, a valid chain of actions and subplans can be found that, when executed, will lead the agent to the desired state, and to the completion of the given task. This can then be returned by the deliberation process as the chosen plan to execute in the main BDI loop. If the problem is over-constrained, there is the possibility that no solution is available for this plan, in which case the agent will try to coordinate, by delegation or exchange, this task with another agent that is able to complete it.

5 Scheduling and Cooperation

Lesser, when discussing multi-agent coordination (Lesser, 1998), argues that a multi-agent system must exhibit the following characteristics:

- A high level representation of its state and goals (*medium to high-grain granularity*)
- Problem solving must be at this high, abstracted level so that the agent has the ability to react quickly to change and can use its *autonomy* to decide at *run-time* on the method of problem solving most appropriate for the goal, given the current situation.
- The agents must have a broad representation, not only of the current state of the environment, but (possibly incomplete) knowledge of the capabilities, problem-solving progress and current status of other known agents so that an agent does not repeat or re-derive existing results or transmit outdated or unrelated information to other agents.

Our agent architecture fulfils these requirements as follows: Since constraints are used as a declarative representation of the tasks, the *high-grain granularity* is accomplished, because no task is given a specific method of completion. Tasks can also be split into subtasks by agents, allowing smaller *medium-grain* tasks to be exchanged. Any exchange of tasks between the agent's cooperation and scheduling layers will only deal with tasks specified in these high level terms. The actual low level execution of the tasks is done by the agent's BDI layer.

We provide the representation of other agents and their capabilities by the use of the data model as a representation of the agents beliefs. This belief schema allows the agent to represent information on other agents through the cyclical `other_agents` relationship back to the Agent entity (Fig. 5).

5.1 Agent Coordination model

We are using existing Generalised Partial Global Planning techniques (GPGP) (Decker and Lesser, 1995) as a basis for the agent communication model. Our main aim is to show the advantages of using constraints and CLP in an agent architecture, so we are basing our work on an existing coordination mechanism, rather than constructing a new model, which we believe is beyond our scope. The GPGP approach is appropriate in this scenario for a number of reasons. Firstly, it has been used and tested in similar domains such as the DVMT (Durfee et al., 1987). Secondly, the architectural base suggested for the agents is also similar to our own constraint based architecture (consisting of a scheduler,

coordination module and a beliefs database). Lastly, the GPGP coordination mechanisms are designed to be domain independent, which allows us to expand the current work to other scenarios in the future.

GPGP consists of five main coordination mechanisms which generalise and extend the Partial Global Planning algorithm. These mechanisms are designed to be used in any combination, as they each address a particular coordination feature of a task environment.

We have classified the main message primitives communicated between agents in the courier scenario as follows:

- **Inform Agents about environment information** - used to tell other agents of any developments in the environment such as flight delays and traffic jams.
- **Tell Agent's current task completion status** - used to help with the scheduling of other agents' tasks and to help the decision process during task delegation.
- **Notify Agents of task completion** - used when your agent has completed a task, the outcome of which is relied upon by another agent.
- **Delegate Task/Accept Task** - used to give an agent a task and to notify an agent of acceptance of a task.

Three of the five coordination mechanisms from GPGP (Update non-Local Viewpoint, Communicate Result and Handle Hard Coordination relationship) are closely related to the types of information we are exchanging and we have used these as the basis for communication and coordination of our CLP based agents. We have also added our own *delegate/accept* mechanism. We believe these 3 GPGP primitives (along with our extra primitive) are adequate enough to form the basis of a coordination model, given the current test environment, and we have the option of extending the work to encompass other primitives if needed.

5.2 Scheduling Agent Desires

Each agent has a given set of tasks, each with an associated payoff, which they will attempt to complete within the given constraints. The agent's aim is to complete as many tasks as possible, so that they will receive the maximum payoff possible. This also means that each agent will willingly *cooperate* with others if possible (in that they will accept another's task if they can) because it will increase the total payoff that the agent will receive. Communication between agents takes place when a single agent cannot complete its given set of tasks and requires the assistance of other agent(s) to complete one or more of them. The agent must find out *what* task or tasks it cannot complete by creating, as it plans and executes, a schedule showing the start and estimated completion times of current and future tasks. The agent's scheduler therefore takes the agent's desires and tries to find a suitable schedule for completing these desires given the specified constraints. The constraints on the schedule come from various sources:

- The maximisation of total task payoff.
- The location of the task in relation to the current agent location.
- The agent's estimated completion time for the task
- Precedence of tasks (i.e. Task_1 must be completed before Task_2).

The scheduler is written using the ECLiPSe finite domain constraint library. We represent the above constraints as Prolog facts, which are then incorporated into the CLP solving process. The following set of facts shows the internal scheduler representation of two tasks, the first to move a parcel from Inverness to Aberdeen and the second to move a parcel from London to Glasgow, given to an agent with the starting location Plymouth.

```
task(move_token(parcel1, inverness, aberdeen), 4).
task(move_token(insert, glasgow, inverness), 3).
task(move_token(parcel2, london, glasgow), 2).
task(move_token(insert, plymouth, london), 1).

desire(4, 2400, 400).
desire(3, 2000, 400).
desire(2, 2900, 900).
desire(1, 2000, 500).

payoff(4, 5).
payoff(3, 0).
payoff(2, 1).
payoff(1, 0).

before([plymouth, london],[london, glasgow]).
before([london, glasgow],[glasgow,inverness]).
before([glasgow,inverness],[inverness,aberdeen]).
```

The `task/2` structure specifies the task and its associated `task_id` in the scheduler. The `desire/3` structure shows the `task_id` along with the latest possible completion time and task length and the `payoff/2` structure shows the payoff associated with each task. The `before/2` structure is created at runtime to show the agent's initial scheduler ordering, based on its location and the spatial layout of each of its given tasks. The agent starts with a generic map of the test scenario, showing the layout, position and connections of all major points. From this map the agent can get the necessary information for the constructs used by the scheduler. The map is continually altered by the agent to reflect new information gathered as it progresses through the environment (such as blocked roads).

Although the agent has only two tasks to complete, there are two extra tasks inserted into the schedule as *insert* tasks. These insert tasks allow the agent to account for the movement from the completion point of one task to the start point of another (Fig. 7). It also allows the agent to incorporate getting from its initial location to the starting point of the first task. All of the above facts are generated dynamically each time the agent is given a set of tasks to schedule.

Each task is represented as a finite domain of start times over which the scheduler tries to construct a schedule. If no schedule can be found which allows for the completion of each task, then the schedule which *maximises the total payoff* is returned and used as the basis for the selection of tasks to delegate and exchange during coordination between the agents.

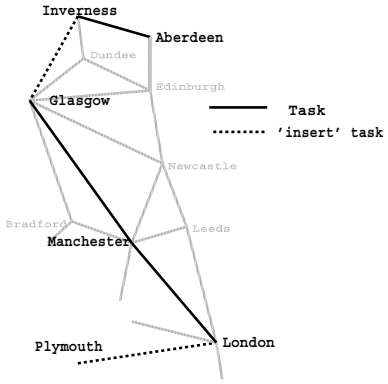


Figure 7: *Schedule with 'insert' tasks*

5.3 Task Delegation and Agent Behaviour

When the scheduler is unable to find a complete schedule for all the agent's given tasks, the agent needs to decide on one or more candidate tasks to delegate or exchange with other agents. The method we are using is to find the schedule with the *greatest total payoff*. When a full task schedule cannot be found, a selection process judges suitable candidates for delegation by first looking at the task with the *lowest payoff* and then the tasks with the *most overlaps* with other tasks. The scheduler is re-run without this task to check that there is now a complete and executable schedule. If the schedule is still not viable, then a selection process finds the best combination of lowest payoff/most overlaps for each task to get a suitable candidate. Although not an optimal scheduling heuristic, this method has provided the necessary means for demonstrating our agents abilities in a cooperating environment. The agent will try to maximise its payoff and only try to delegate tasks when absolutely necessary.

The `delegate` protocol is used to broadcast the task for delegation to others in the system, and the `accept` protocol is used to notify the agent of others willing to take on the task. At present, if two or more agents choose to accept the task, an arbitrary decision is made as to the recipient. This could be extended to take into account the beliefs the agent currently has about the status of the other agents.

As well as sending delegate messages, the agent may also receive delegate messages. If an agent receives a delegate message from another agent then it will do a preliminary schedule run (as described above) with the new task added to check whether it is able to slot in this task alongside its existing ones. If it is possible, then it will send an accept message to the agent and wait for a response (the agent will receive either the task itself or a no-task notification).

Modifying the scheduler and the way in which the scheduler selects and delegates tasks can be seen as modifying the agent's behaviour, as the scheduling algorithm decides which tasks to delegate and which to accept.

6 Implementation

The constituent parts of the agent are implemented as independent Prolog processes, which communicate through a socket based LINDA architecture (Carriero and Gelertner, 1989). This is a Prolog blackboard communication model which allows asynchronous

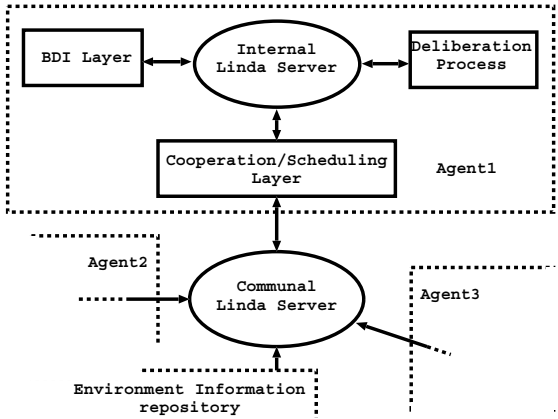


Figure 8: *Agent communication through LINDA servers*

message passing between individual Agents (Fig 8). Messages can be posted to the communal server, and messages can be retrieved by individual agents querying the server. The constituent agent processes also use this LINDA blackboard model for internal communication, allowing processes to post and retrieve messages within the agent in private. In our example scenario, we also use an *environment information repository*, which acts as a source of constantly changing environment information. This essentially acts as the eyes and ears of our agent, passing information (road connections, traffic jams, etc.) particular to that agent's location.

We are using CCQL (Constraint Communication Query Language) (Preece et al., 1999) as the agent communication language, which is based on the KQML knowledge sharing language (Finin et al., 1994).

The agent deliberation process is written in EcLiPse, using the *Propia* constraint library. This library is an EcLiPse implementation of generalised constraint propagation. The idea is that a collection of facts containing any number of variables can be expressed as finite domains and generalisations can be made about those variables (if possible).

```
location(aberdeen, agent1, parcel1).
location(edinburgh, agent1, parcel12).
location(aberdeen, agent2, parcel10).
location(glasgow, agent2, parcel12).
```

In the example above, there are four facts describing two agents, their possible locations and what they could be carrying. Each variable in the facts is turned into a finite domain using the `infers most` command.

```
location(CITY,AGENT,CARRYING) infers most.
```

This specifies `CITY`, `AGENT` and `CARRYING` as finite domains containing the elements specified in the `location/3` construct:

```
CITY = CITY{[aberdeen,edinburgh,glasgow]}
AGENT = AGENT{[agent1, agent2]}
CARRYING = CARRYING{[parcel1, parcel10,parcel12]}
```

If we apply the constraint `CARRYING #\= parcel12`:

```
CITY = aberdeen
AGENT = AGENT{[agent1, agent2]}
CARRYING = CARRYING{[parcel1, parcel10]}
```

We can see that the constraint mechanism removes `parcel12` from the `CARRYING` domain. It also infers that `CITY` is restricted to `aberdeen` by using the relationships between the variables specified by the `location/3` construct to propagate this information. Although only a simple example, when used in the deliberation process described in section 4, where we are dealing with several hundred facts and complex relationships between those facts, this generalisation process greatly reduces the domain sizes, and allows us to infer as much information as possible from whatever is available. It also means that a set of facts (such as those from the P/FDM database we are using) can be easily transformed into finite domains for use in a CLP solver.

The scheduler is also written in ECLiPSe and is constructed as a finite domain constraint problem using the ECLiPSe finite domain library. The Belief data model is held in a P/FDM database. P/FDM is a Prolog based implementation of the functional Data Model (Shipman, 1981) which incorporates many object-oriented features. It is developed and maintained at Aberdeen university, and has been used on a number of large projects, from distributed knowledge manipulation to three dimensional protein structure representation. The Desires are represented in COLAN (based on Shipman's Daplex language), a high-level query and constraint language used initially to specify database integrity constraints in P/FDM. We have found this constraint language, and the data model representation, to be independent of the problem domain and it has proved useful in its ability to represent first order logic constraints over finite domains of objects in a database, or subranges of integers or enumerated types.

7 Evaluation

The main constraint solving mechanism has been used extensively in the KRAFT project as a way of solving configuration tasks. This has proved to be an effective solution and has been tested on real world scenarios from the communications industry. The constraint mechanism provides a way in which product specifications from numerous vendors can be used, along with user defined specifications to provide effective solutions to complex configuration problems.

We have tested this mechanism in a courier scenario, where the problem is of a temporal nature, rather than spatial. In this courier environment, we have used the techniques to deliberate on constraint problems containing between 500 and 5000 facts of the type described in section 4.3. These facts represent from 1–5 lookahead actions for the agent, with 3–5 options at each stage of the decision process (in total up to 770+ options). We have looked at various representations of these options in the decision making process, and have received near instantaneous results from the CLP process when retrieving answers. Each agent has its own CLP, so there is no centralised planner, meaning the average sizes of the tree structures created by each agent are manageable.

Whilst the scheduler implementation is basic, and the scheduling strategies used not comparable to most dedicated schedulers, it has been built for the purpose of showing the capabilities of the CLP solver and the autonomy which it provides, and the use of constraints in the representation of the desires, rather than to show sophisticated scheduling procedures. The modularity of the system means that new scheduling algorithms can be substituted for the current one if needed. Currently, there is no facility for task splitting (e.g. an agent may take a parcel some distance towards its destination, then pass it to

another agent which finishes the journey). We have also to look at the idea of task integration, where one task may be completed during the execution of another. This also brings in the idea of redundancy, which has not been considered in this scenario (although the GPGP has a *Handle Simple Redundancy* method).

We are also concerned with the representation of the tasks themselves in the scheduler. We are looking at making them more flexible, and allowing the scheduler the ability to 'relax' the constraints. This would involve using a task quality metric (other than the currently implemented 'payoff') which would allow the agent to complete the task at a later time, but would give the resulting task a lower quality of completion.

With a more sophisticated scheduler, we could investigate ideas such as task splitting, redundancy, task integration and constraint relaxation but we believe that the current scheduler helps demonstrate the main focus of our research.

8 Related Work

We are modelling the agent's behaviour and deliberation using techniques from the KRAFT project. Here, constraints are used as problem specifications, combined with a data model approach, to find solutions to distributed design problems. The various data sources used have their own constraints that are fused together and re-used in the context of a solution-space data model. Much of the agent deliberation process is based on this work, and in particular the techniques outlined in (Hui and Gray, 2000). Whereas KRAFT has been concerned with spatial configuration problems, we are applying the techniques developed to temporal problems and situations.

The Distributed Intelligent Agents Group, headed by Edmund Durfee at Michigan University have used GPGP (Generalised Partial Global Planning) and other planning algorithms, such as hierarchical planning and plan summary information, to implement multi-agent planning systems that work on an evacuation travel scenario testbed (Clement and Durfee, 2000). Whilst being similar in approach, the agents used by Durfee et.al are not intentional, and we aim to use information inferred from the agent's constraints in place of summary plan information. O-PLAN (Tate et al., 1996) also uses an evacuation scenario as a testbed environment. It uses a hierarchical planning system, along with domain constraint knowledge to direct its search for plans. Again, we are using intentional agents rather than specifically designed planning agents and we are making use of the agent's data model to represent beliefs and constraints specific to those beliefs, as well as more general domain constraint knowledge. In (Hofmann et al., 1998) they also explore giving an agent enough autonomy to act and make decisions independently. Although using mobile agents rather than BDI agents, they state that the agent must have "the ability to operate independently of the user and persist in their determination to execute their assigned task".

The main idea common to all these projects is that the agent has to be able to adapt to change in a highly dynamic environment, while still planning and scheduling events on a larger, more long-term scale.

Our current work is concerned with the use of this agent architecture on a transport domain. In future, we wish to look at the way in which the formal and practical aspects of constraints and this reasoning and the constraint solving methods developed can be used with cooperating *and* competing agents in larger e-commerce scenarios. We also wish to look at developing the ideas of relaxing and negotiating constraints touched upon in section 7.

9 Conclusions and Future Work

We have described a method for giving an agent autonomous decision making capabilities in a dynamic, multi-agent environment. We have shown that using range-restricted, First Order Logic constraints provides a powerful and expressive method for describing Desires in a BDI agent architecture. The use of a data model means that we can express agent Beliefs using complex semantic relationships and provide the agent with a local ontology which can be used as a basis for transforming and combining Desires, represented as constraints, from many disparate sources. Thus the agent is able to take into account information from the environment, from other agents as well as its own Beliefs, giving it an intelligent, autonomous mechanism for planning and deliberating.

From our initial tests in a courier scenario, we have seen that the deliberation process allows an agent to plan and execute given tasks, based not only on the task's delivery deadline, but also on the current information it has on road closures, flight delays etc. and to make informed decisions based on its current situation.

While not an always optimal planning mechanism, we believe that our CLP approach gives an agent the ability to reason and plan in an environment where it may have incomplete information which can change over time, and the ability to deal with a rich variety of constraints in many combinations and with new and unplanned events as and when they occur.

Our current work is concerned with the use of this agent architecture on a transport domain. In future, we wish to look at the way in which the formal and practical aspects of constraints and the reasoning and constraint solving methods developed can be used with cooperating *and* competing agents in larger e-commerce scenarios. We also wish to look at developing the ideas of relaxing and negotiating constraints touched upon in section 7.

Acknowledgements

This work is supported by an EPSRC grant under the supervision of Prof. P.M.D. Gray. The work is based on work previously completed during the KRAFT project, and in particular the work of Kit-Ying Hui and Alun Preece. The KRAFT project was funded by grants from BT and EPSRC.

References

Bassiliades, N. and Gray, P. M. D. (1994). Colan: a functional constraint language and its implementation. In *Data and Knowledge Engineering 14*, pages 203–249. See also http://www.csd.abdn.ac.uk/ ∼pfdm/user_manual/section2_3_4.html.

Bratman, M. E. (1987). Intentions, plans and practical reason. Harvard University Press, Cambridge, MA.

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1999). Plans and resource-bounded practical reasoning. In *Computational Intelligence*, volume 4, pages 349–355.

Carriero, N. and Gelertner, D. (1989). Linda in context. In *Communications of the ACM, 32(4)*.

Clement, B. J. and Durfee, E. H. (2000). Performance of coordinating concurrent heirarchical planning agents using summary information. In *Pre-Proceedings of the Seventh International Workshop on Agent Theories, Architectures and Languages*, pages 202–216.

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. In *Artificial Intelligence 42(3)*.

Decker, K. and Lesser, V. R. (1995). Designing a family of coordination algorithms. In *UMass Computer Science Technical Report 94-14*.

Durfee, E. H., Lesser, V. R., and Corkill, D. (1987). Coherent cooperation among communicating problem solvers. In *IEEE Transactions on Computers C-36(11)*, pages 1275–1291.

Eaton, P., Freuder, E., and Wallace, R. (1998). Constraints and agents: Confronting ignorance. In *AI Magazine 19(2):50-65*.

ECRC and IC-Parc (2000). *ECLiPSe Library Manual*. www.icparc.ic.ac.uk/eclipse/.

Finin, T., Fritzon, R., McKay, D., and McEntire, R. (1994). KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94)*.

Genesereth, M. R. and Nilsson, N. (1987). Logical foundations of AI. Morgan Kaufmann Publishers: Santa Mateo, CA.

Gray, P. M. D., Embury, S. M., and Kemp, G. J. L. (1999a). The evolving role of constraints in the functional data model. In *Journal of Intelligent Information Systems*, volume 12, pages 113–117. Kluwer Academic Press.

Gray, P. M. D., Hui, K., and Preece, A. D. (1999b). Finding and moving constraints in cyberspace. In *Proc. AAAI Spring Symposium on Intelligent Agents in Cyberspace (SS-99-03)*, pages 121–127.

Hofmann, M. O., McGovern, A., and Whitbread, K. R. (1998). Mobile agents on the digital battlefield. In *Proceedings of the Second International Conference on Autonomous Agents*. ACM Press.

Hui, K. and Gray, P. M. D. (2000). Developing finite domain constraints – a data model approach. In *Proceedings of the 1st International Conference on Computational Logic*, pages 448–462. Springer-Verlag. See also http://www.csd.abdn.ac.uk/∼apreece/Research/KRAFT.html.

Lesser, V. R. (1998). Reflections on the nature of multi-agent coordination and its implications for an agent architecture. In *Autonomous Agents and Multi-Agent Systems*, pages 89–111. Kluwer Academic Publishers.

Preece, A. D., Hui, K., Gray, W. A., Marti, P., Bench-Capon, T. J. M., Jones, D. M., and Cui, Z. (1999). The KRAFT architecture for knowledge fusion and transformation. In Bramer, M., Macintosh, A., and Coenen, F., editors, *Research and Development in Intelligent Systems XVI (Proc ES99)*, pages 23–38. Springer.

Rao, A. S. and Georgeff, M. P. (1995). BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*.

Shipman, D. W. (1981). The functional data model and the data language DAPLEX. In *ACM Transactions on Database Systems 6(1)*, pages 140–173.

Tate, A., Drabble, B., and Dalton, J. (1996). O-Plan: a knowledge-based planner and its application to logistics. In *Advanced Planning Technology*. AAAI Press.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. In *The Knowledge Engineering Review*, volume 10, pages 115–152.

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Using Cognition and Learning to Improve Agents' Reactions

Pedro Rafael Graça and Graça Gaspar
Department of Computer Science
Faculty of Sciences of the University of Lisbon
Bloco C5 - Piso 1 - Campo Grande, 1700 Lisboa, Portugal
E-mail: prafael@di.fc.ul.pt, gg@di.fc.ul.pt

Abstract

This paper proposes an agent architecture to deal with real-time problems where it is important both to react to constant changes in the state of the environment and to recognize generic tendencies in the sequence of those changes. Reactivity must satisfy the need for immediate answers; cognition will enable the perception of medium and long term variations, allowing decisions that lead to an improved reactivity. Agents are able to evolve through an instance-based learning mechanism fed by the cognition process that allows them to improve their performance as they accumulate experience. Progressively, they learn to relate their ways of reacting (reaction strategies) with the general state of the environment. Using a simulation workbench that sets a distributed communication problem, different tests are made in an effort to evaluate the utility of the multi-agent system architecture and the importance of the individual features of agents, the utility of using a set of different strategies, and the significance of the learning mechanism. The resulting conclusions point out the most significant aspects of the generic model adopted, helping to put it in perspective as a solution for other problems.

1 Introduction

1.1 Motivation

Reaction, cognition and the ability to learn are among the most fundamental aspects of human behaviour. Daily, we react to a non-deterministic and constantly changing world, often facing unknown situations that nevertheless need immediate answer (for example, crossing an unknown street for the first time); we constantly rely on our cognitive ability to classify the surrounding environment (for example, choosing the best place to cross the street); we use our experience to select actions for specific situations (for example, quickly crossing the street when the sign turns red). Generally, cognition and the ability to learn lead to the accumulation of experience, allowing better decisions that improve the selection of actions. This is the central idea of the agent architecture proposed in this paper: the agents have a reaction module that allows them to answer in real-time, a cognition module that successively captures and classifies images of the environment, and a learning module that accumulates experience that progressively allows a better selection of actions.

1.2 Environment

A real-time group communication simulated environment (reproduced by a simulation workbench) supported the development and study of the proposed architecture. Prone to non-deterministic and continuous changes (traffic fluctuations), such an environment emphasizes the need for immediate reactions. At the same time, the cyclic occurrence of similar environment states (for example, periods with low traffic level) and the repetition of changing patterns (for example, brisk increases of traffic level) point to the utility of a cognitive system that enables a form of learning, allowing the use of accumulated experience.

In this distributed communication environment, where the traffic level is variable and messages can be lost, each agent is responsible for sending and eventually resending (when losses occur) a set of messages. Each agent's goal is to optimise the timeout interval for resending lost messages, in such a way that the sending task is completed as soon as possible and the communication system is not overloaded by unnecessary resending. The agent chooses from a set of tuning strategies that over time it learns to relate to the state of the communication system, progressively improving its performance.

1.3 Related work

In the context of concrete applications of multi-agent systems in traditional telecommunication problems, our main goal is to put in perspective the relationships between the generic problems observed in a distributed application and the generic answers that a multi-agent system is able to offer, in an abstraction level more concerned with properties than with detail. Even though no studies similar to ours were found (involving reaction, cognition and machine learning in a telecommunication problem), many other investigations in the field of multi-agent systems and telecommunications address problems in real-time environments that share many of the essential characteristics. A wide diversity of studies address problems such as routing, traffic congestion, scalability, fault location, and cooperative work, to mention only a few. Work on this area has been conducted by Albayrak (1999) and Hayzelden et al (1999).

Mavromichalis and Vouros (2000) and Malec (2000) propose layered agent architectures to address the problem of controlling and balancing reactivity and deliberation in dynamic environments requiring real-time responsiveness. These perspectives show some similarities to our work, but they do not incorporate a machine learning mechanism.

Weiβ (2000) discusses the relationship between learning, planning and reacting, proposing an extension to a single-agent architectural framework to improve multi-agent coordination. The learning mechanism is used in order to determine the best way of alternating between reaction-based and plan-based coordination. In this particular, our study is significantly different: our learning mechanism concerns how to react in environments where reaction is a necessity rather than an option.

Work concerning a learning approach in some regards close to ours can be found in the writing of Prasad and Lesser (1999). They propose a system that dynamically configures societies of agents, using cognition and/or communication as the basis for learning specific-situation coordination strategies.

2 A group communication problem

2.1 Motivation

The communication problem used in this investigation was idealized following two main requirements:

- the preservation of the generic properties of a real-time distributed application;
- the avoidance of complex situations that could make the interpretation of results a more difficult task.

To meet the first requirement, we selected a group communication problem, a typical and intuitive real-time distributed situation, offering a high degree of versatility concerning the desired complexity level involved. To meet the second requirement, we used a simulation workbench that reproduced the selected problem, simplifying accessory aspects and preserving all the essential properties that ensure the accuracy and expressiveness of the results.

As a good example of the benefits introduced by the simplifications that took place, it is considered that, although normal messages can be lost in the communication process, acknowledgments cannot. Since from the message sender point of view both of these losses are equivalent and undistinguishable, the occurrence of acknowledgment losses would increase complexity without enriching the study or its results.

2.2 Description

The problem in question was conceived in order to serve a generic purpose, but the description of a real and specific situation will help to clarify its outlines. Imagine a team of stockbrokers, each of them working on a different stock market. Suppose that, in order to coordinate the team's global action, there is a synchronization rule that establishes that each individual can only perform an action after being informed of every other team member's intention. Consider that it is important to perform as many operations as possible and that the communication between stockbrokers takes place in a telecommunication system where the traffic level is variable and messages can be lost. This scenario corresponds to the distributed communication problem adopted in this investigation.

Each agent is associated to a communication node, having the responsibility of sending and resending (when losses occur) a message to each other node. When a message arrives to its destination, an acknowledgment is sent back to the message sender. In each communication *season*, the users (each associated to a node) exchange messages with each other. One season ends when the last message (the one that takes more time to successfully arrive) reaches its destination.

The traffic level on the communication network varies over time, influencing the reliability: an increase of traffic level causes a decrease of reliability, increasing the occurrence of message losses; a decrease of traffic level has the opposite effect. The better the agents are able to adapt to the sequence of changes, the more accurate becomes the chosen instant for resending lost messages. Increased accuracy improves the communication times, causing the duration of seasons to decrease.

It is important to notice that the communication problem described takes place at application level. In environments where the sequence of events is very fast (imagine a millisecond time scale) the ability for reacting very quickly is often more important than the ability for choosing a good reaction. The time needed to make a good choice can actually be more expensive than a fast, even if worse, decision. Because of this, the

agent architecture proposed in this paper is better suited for problems where the time scale of the sequence of events justifies efforts such as cognition or learning. This does not mean that quick answers to the environment are not possible: deliberation (recognising, learning and deciding) can easily become a background task, only showing its influence on the quality of reactions when there is enough time to identify environmental states and use previously acquired experience. On the worst case (millisecond time scale) this influence will tend to be null and agents will react without deliberating. The better the deliberation process can accompany the sequence of events, the greater will this influence be.

3 Agent architecture

3.1 Introduction

Considering the communication problem adopted, the agents' task is to tune the timeout interval for resending lost messages, so that the duration of the communication seasons and the occurrence of unnecessary resending are both minimised. In their tuning task, agents must deal with information at different time scale perspectives: they must immediately react to constant variations in the state of the environment and also be able to recognize tendencies in the sequence of those variations so that a learning mechanism can be used to take advantage of accumulated experience.

To react to constant variations, each agent uses one of several *tuning strategies* at its disposal. To evaluate the quality of a tuning strategy in a communication context (for example, during a low traffic period) the period during which that strategy is followed cannot be too short; to allow the search for better strategies, this period should not last too long. These opposing desires led to the introduction of the *satisfaction level* concept, a varying parameter that regulates the probability of a strategy change decision (the lower the agent's satisfaction level is, the more likely it will decide to adopt a new strategy). As will be briefly explained below, this satisfaction level depends on two additional aspects:

- the detection of changes in the state of the environment (communication conditions);
- the self-evaluation of the agents' performance.

To recognize non-immediate tendencies in the sequence of environment state changes, the agent uses its cognition system. The information collected in each communication season is gathered in a memorization structure. This structure is periodically analysed in order to abstract from the details of basic states, fusing sequences of those basic states into generic states classified in *communication classes* and detecting important variations in the state of the communication system (for example, a transition from a traffic increase tendency to a traffic decrease tendency). The result of this analysis influences the satisfaction level; for example, a change of the communication class, or the detection of an important variation, can cause the satisfaction level to decrease, motivating the agent to choose a new tuning strategy (possibly fitter to the new conditions).

Progressively, agents learn to relate the tuning strategies and the communication classes. The establishment of this relationship depends on two classification processes: the classification of the agents' performance and the classification of the generic state of the environment (the communication classes processed by the cognitive system).

A scoring method was developed in order to classify the agents' performance. As the duration of the timeout interval depends on the tuning strategy in use, the qualifica-

tion of an agent's performance in a sequence of seasons is a measure of the fitness of the strategy used to the communication class observed during those seasons.

The diversity of states of the environment emphasizes the utility of studying a multi-agent system where different individuals may have different characteristics. While an *optimism level* regulates the belief in message losses (the more pessimistic the agent is, the sooner it tends to conclude that a message was lost), a *dynamism level* regulates the resistance to stimulation (the less dynamic the agent is, the less it reacts to changes, the longer it keeps using the same strategy). Each different agent has a specific behaviour and interprets the surrounding environment in a different way.

In this section, after introducing some terminology (subsection 3.2), the details of the proposed agent architecture are presented in the following order:

- the tuning strategies (subsection 3.3);
- the scoring method (subsection 3.4);
- the cognitive system (subsection 3.5);
- the learning system (subsection 3.6).

Finally, a diagram (subsection 3.7) and an illustration of an operating agent (subsection 3.8) give a global perspective of the architecture.

A more detailed description of this agent architecture is given by Graça (2000).

3.2 Terminology

The period of time needed to successfully send a message (including eventual resending) and to receive its acknowledgement is called *total communication time*. When a message successfully reaches its destination at the first try, the *communication time* is equal to the total communication time; otherwise, it is the time elapsed between the last (and successful) resending and the reception of the acknowledgement.

The ending instant of the timeout interval is called *resending instant*. It is considered that the *ideal resending instant* of a message (the instant that optimises the delay) is equal to the communication time of that message.

The difference between the resending instant and the ideal resending instant is called *distance to the optimum instant*.

A high increase or decrease of traffic level immediately followed by, respectively, a high decrease or increase is called a *jump*. A high increase or decrease of traffic level immediately followed by stabilization is called a *step*.

3.3 Tuning strategies

To follow the fluctuations of the communication system, each agent constantly (every communication season) adjusts the resending instant. It is possible to imagine several ways of making this adjustment: following the previous communication time, following the average of the latest communication times, accompanying a tendency observed in the succession of communication times, etc. A *tuning strategy* is a way of adjusting the resending instant. It is a simple function whose arguments include the communication times observed on previous seasons and the optimism level, and whose image is the resending instant to be used on the following season.

A set of ten tuning strategies is available to the agents, including for example: a *reactive* strategy (according to this strategy, the communication time observed in season t is used as resending instant in season $t+1$), an *average* strategy (as shown in Figure 1, each resending instant is defined according to the average of all previous communica-

tion times), a *semi-reactive average* strategy (the last communication time is weighted by one tenth in the average calculus), an *almost reactive average* strategy (as shown in Figure 2, the last communication time is weighted by one third in the average calculus), a *reactive ignoring jumps* strategy (works like the reactive strategy but keeps the same resending instant when jumps occur). A *TCP* strategy, reproducing the real procedure adopted by the TCP/IP protocol (see Peterson and Davie (1997) for details), was also included in this set. According to this strategy (Figure 3) the more unstable the communication conditions are, the bigger is the safety margin used (higher resending instants).

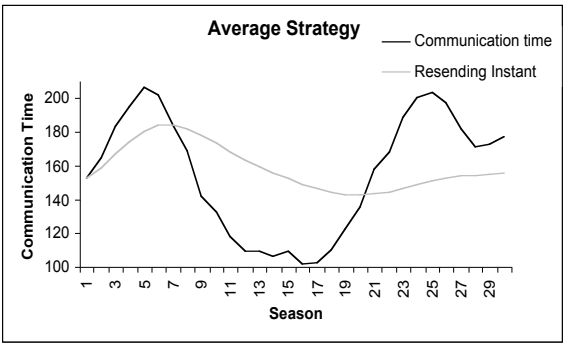


Figure 1: *Average tuning strategy*

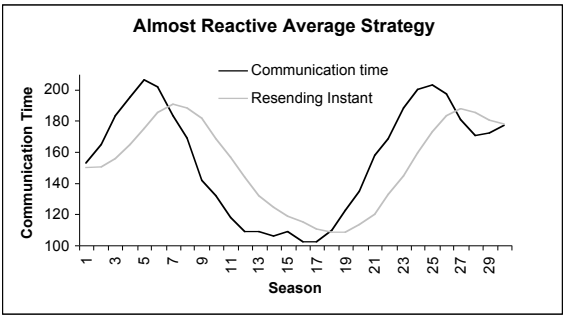


Figure 2: *Almost reactive average tuning strategy*

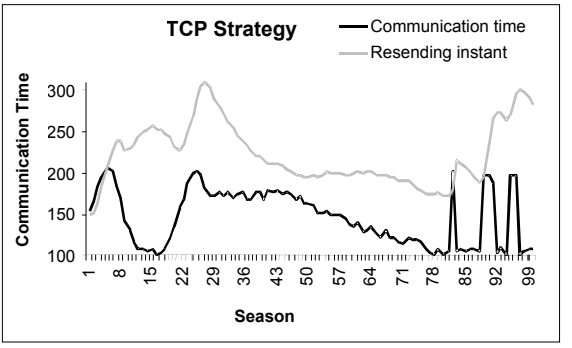


Figure 3: *TCP tuning strategy*

It is expected that the diversity of strategies helps to match the diversity of environmental states: different tuning strategies will be differently fitted to the different communication classes. For example, when traffic fluctuations are unpredictable, a re-

active strategy will probably produce better results than an average-based strategy; the opposite will probably occur if the traffic level tendency is to vary within a sufficiently well determined interval. The goal of the learning system is precisely to select for each communication class the strategies that produce better results.

As mentioned before, the optimism level regulates an agent's belief in message losses: the more pessimistic the agent is, the sooner it tends to conclude that a message was lost, the sooner it tends to resend it. Agents with different optimism levels use the same tuning strategy differently: based on the same unmodified resending instant given by the strategy, a pessimistic agent shortens the delay and an optimistic agent widens it (Figure 4 shows this effect).

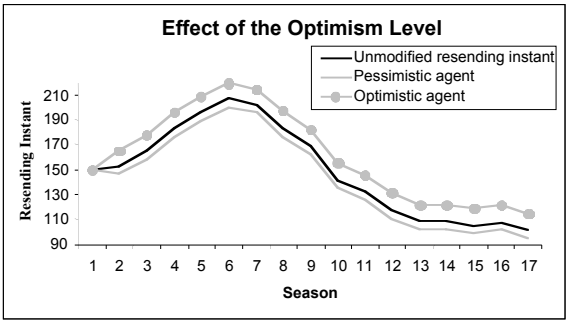


Figure 4: *Effect of the optimism level on strategies*

This differentiation (along with the dynamism level) opens the possibility of establishing relations between certain types of agents (for example, highly pessimistic) and certain communication conditions (for example, constant traffic level).

3.4 Performance evaluation

The evaluation of the agents' performance has three main goals:

- to measure the significance of different individual features: it may be possible to determine a measure of the fitness of certain agents' characteristics to the communication conditions, if a specific type of agent (for example, very optimistic) tends to receive higher or lower scores under those conditions;
- to allow the agents to adjust their procedure: poor performance causes a decrease of the satisfaction level, and eventually leads to a change of strategy;
- to support the classification of training examples: this will be detailed further ahead, in the learning system subsection.

At the end of each season, each agent has information that allows it to qualify its performance. The score obtained helps each individual to answer the following question: how accurate were the resending instants chosen? As it will be detailed in the learning system subsection, the successive scores will help to answer another question: how fitted is the chosen strategy to the communication class currently observed?

The *main* performance evaluation score is obtained considering the sum of distances to the optimum[1] instants, adding a penalty for each message unnecessarily resent (the lower the score, the better the performance).

[1] The concept of optimum instant was presented at section 3.2.

To clarify this system, consider the following example:

- agent A sends 3 messages during a communication season, to agents B (*mAB*), C (*mAC*) and D (*mAD*);
- the resending instant for *mAB* is 150 time units;
- the resending instant for *mAC* is 180 time units;
- the resending instant for *mAD* is 140 time units;
- the acknowledgement from *mAB* arrives after 146 time units;
- the acknowledgement from *mAC* arrives after 183 time units (an unnecessary re-send occurred);
- the acknowledgement from *mAD* arrives after 270 time units (this means the first message wasn't acknowledged, and the second one's acknowledgement arrived after 130 time units).

Given this situation, agent A would get:

- 4 points for *mAB* (150-146);
- 6 points for *mAC* (183-180=3, penalty for unnecessary resend[2]: 3*2=6);
- 10 points for *mAD* (140-130);

Agent A's final score is 20 points. For example if the resending instant for *mAB* were 155 time units (further away from the optimum instant by 5 time units), the score would have been 25 points (worse than 20).

Two additional *auxiliary* scoring mechanisms are also used in order to evaluate the agents' performance[3]. Each of these mechanisms ranks the agents on each season (from first to last place), according to each of the following criteria:

- the necessary time to complete the sending task (the best one is the first to receive all acknowledgements);
- the quality of the chosen resending instants (same criteria as the main method described above).

At the end of each season, each agent is scored according to its rank (the first of *n* agents gets 1 point, the second 2 points, and so on until the last that gets *n* points).
The information about every agent's performance is gathered and processed at the end of each season in order to establish the rank. On a simulation workbench this is a trivial task because all the information is locally available. On a real situation, a way of gathering the information and broadcasting the results would have to be studied.

The purpose for these auxiliary mechanisms is to allow the agents to compare each other's performance. When the communication conditions are unstable the resending instants are more difficult to set and, although the agent's performance may be good (considering the complexity involved), the main score (determined in an absolute way) will tend to be lower. In these cases, the auxiliary scores (determined in a relative way) can help each agent to correctly evaluate the quality of its performance.

3.5 Cognitive system

The information memorized after each season is continuously analysed and processed in order to provide the agent with an image of the world. The memorization structure,

[2] The penalty for unnecessary resend equals the distance to the optimum instant: the larger the distance the bigger the penalty.
[3] These auxiliary scoring mechanisms weren't considered in the classification of training examples, but they also influence the satisfaction level.

more than just an information repository, is a fundamental part of the cognitive system; among other information (arguments for the tuning strategies), it stores:

- a *short memory block*: contains each ten consecutive average communication times and the average performance score during those ten seasons (state of the environment in the last few seasons);
- a *global memory block*: the result of a process of synthesis of past short memory blocks (state of the environment during a wider period).

Every ten seasons, a short memory block is processed in order to obtain information that is then added to the global memory block. This synthesised information includes: a set of parameters that characterize the traffic oscillation (for example, how many increases of traffic level were observed during the ten seasons), the communication class observed, and the average performance score.

A *communication class* is a generic classification of the state of the environment. Such a classification is determined from the parameters that characterize the traffic conditions (obtained from each short memory block), and has three dimensions: the *traffic level* (high, medium, low, variable), the *traffic variation tendency* (increase, decrease, constant, alternating, undetermined) and the *sudden variations occurrence* (jumps, steps, jumps and steps, none).

The detection of variations in the communication system is based on the following principle: the greater the difference between the global communication class (the communication class of the global memory block) and the communication classes of the last short memory blocks, the more likely it is that a significant variation is occurring. A metric of three-dimensional distance between communication classes was developed in order to apply this idea (considering, for example, that the difference between a high and a medium traffic level is smaller than the difference between a high and a low traffic level). The distance between two communication classes is obtained by adding the distances between each dimension's members.

The detection of variations causes the satisfaction level to progressively decrease, motivating the agent to choose a new tuning strategy more adequate to the new communication class. When a variation is first detected, the decrease of satisfaction is generally small; in this way, if the variation is merely transitory its impact will be minimal. However, if the variation is progressively confirmed, the decrease in the satisfaction level is continuously accentuated: the more obvious and significant the variation is, the larger becomes the satisfaction level decrease.

3.6 Learning system

The agents must be able to adapt to the conditions of the communication system, selecting the best strategies for each communication class. This requirement appeals for a learning mechanism that builds up and uses accumulated experience. When its performance isn't satisfactory (bad score), an agent must learn that the strategy used in the current communication class is probably not adequate. If the performance is good, the agent must learn the opposite.

The learning mechanism is based on the following cyclic process associated to internal state transitions:

- Finalization of the previous state (a new training example is stored);
- Prevision of a communication class and selection of a tuning strategy for the next state;
- Beginning of the new state.

When the satisfaction level is low (bad performance and/or variation detected), an agent may decide to finalize its current state. The *dynamism* level associated to each agent makes it possible for two agents to act differently when facing the same conditions: a more dynamic individual is more likely to feel dissatisfied and will consequently change its state more often then a more conservative individual.

An agent's state (from the learning mechanism perspective) consists of a sequence of communication seasons, characterized by a global communication class, a strategy in use, and a performance score. When an agent decides to end the current state, this characterization is used to build a new *training example*, a <*communication class, tuning strategy, performance score*> triple. The training examples are stored into a two-dimensional table (the *experience table*) that contains the average score for each <communication class, strategy> pair, recalculated every time a correspondent training example is added. The more training examples (concerning a particular communication class) an agent stores, the higher is its *experience level* (in that class).

This form of case based learning has some specific characteristics: the learning cases are processed instead of being stored (only necessary information is preserved); it admits an initially void base of cases; it is dynamic, in the sense that new learning cases cause previous information to be progressively adjusted (recalculation of the average score). This learning mechanism has therefore some resemblance to a simple way of reinforcement learning[4].

Before initiating a new state, the agent must predict a communication class and choose a new strategy. The prediction of a communication class for the next state is based on the following complementary ideas: if the state transition was mainly motivated by bad performance, the communication class remains the same; if it was motivated by the detection of variations, then those variations are analysed in order to predict a new communication class. This analysis considers the latest three short memory blocks, using their data to determine a new communication class (assuming that the lately detected variations characterize the new state of the communication environment). If transition patterns were to be found in the sequence of states, this prediction process could be enhanced by the use of another learning mechanism that were able to progressively determine which transitions were more likely to occur.

To select a new strategy an agent may consult the experience table (selecting the best strategy according to the predicted communication class), choose randomly[5] (when it has no previous experience or when it decides to explore new alternatives), or consult a shared blackboard (described ahead). The random selection of strategies is the preferred option when the agent has a low experience level, being progressively abandoned when the experience level increases (even when experience is very high, a small probabilistic margin allows random selections).

The shared *blackboard* is used as a simple communication method for sharing knowledge between agents. Every ten seasons, the most successful agents (those who receive better scores) use it to register some useful information (strategy in use and communication class detected), allowing others to use it in their benefit. When the agents do not use this communication mechanism, learning is an individual effort; if it is

[4] Our initial idea was indeed to use reinforcement learning, but a more careful study revealed incompatibilities between this learning mechanism and the addressed problem: on the state transition process, there is no necessary relationship between the action an agent selects (new tuning strategy), and the following state (that includes the communication class, which is generally not influenced by the agent's choice); moreover, a state is only identified at the moment of its conclusion. These facts oppose the most basic principles of reinforcement learning.

[5] Random selection could be replaced by another specific distribution (such as Boltzmann distribution).

used in exclusivity (as the only way to select a new strategy), learning does not occur[6]. When it is optional, it enables a simple way of multi-agent learning: by sharing their knowledge, the agents allow others to benefit from their effort, eventually leading each other to better solutions earlier than it would happen otherwise.

When a new state begins, the agent's memory (short and global blocks) is initialised. If, in a short period of time (first thirty seasons), the predicted communication class proves to be a wrong prevision (because the analysis was wrong or because the conditions changed), the agent may choose to interrupt the new state to correct it. In this case, regarding the interrupted state, no training example is considered.

3.7 Agent architecture diagram

Figure 5 summarizes the agent architecture presented in this paper.

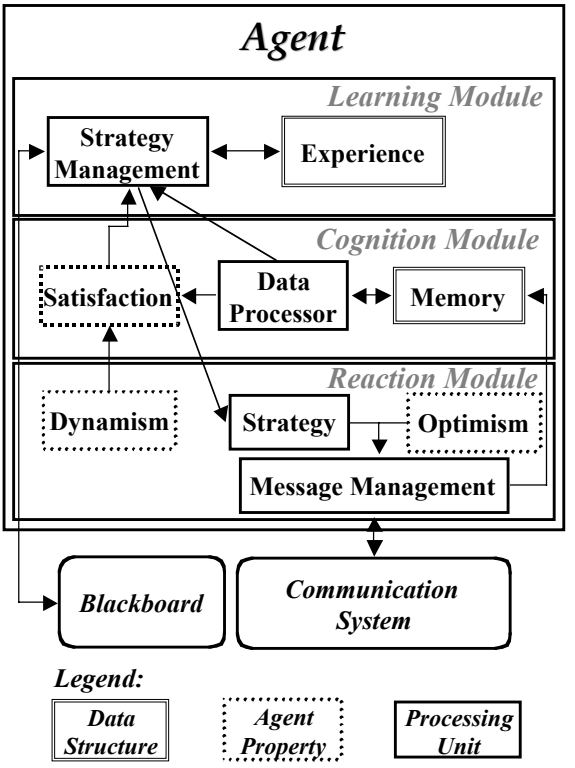


Figure 5: *Agent architecture diagram*

The *reaction module* includes a message management unit, responsible for sending and resending messages according to the strategy in use (modified by the optimism level). The strategy is changed when the strategy management unit produces such a decision.

The data processor unit included in the *cognition module* is responsible for analysing the information that is constantly memorized, evaluating the environment (communication classes) and the performance (score). Conditioned by the dynamism level, the result of this analysis influences the satisfaction level. Whenever a state transition oc-

[6] An agent whose only method of strategy selection is consulting the blackboard is considered opportunistic: he develops no learning effort and always relies on other agents' work.

curs, the data processor sends necessary information to the learning module so that a new training example can be created and stored.

The strategy management unit included in the *learning module* is responsible for the state transition decision (conditioned by the satisfaction level) and its related operations. It stores the training examples in the experience data unit.

3.8 Learning with agent Adam

The following description illustrates a sequence of operations of a specific agent during a learning state. This example can help to clarify the concepts presented along this section, showing how they are applied in a practical situation.

Agent Adam, a very optimistic and very dynamic individual, was using an average-based strategy to tune the resending instants. Since he is very optimistic, the resending instants are delayed accordingly (a less optimistic agent following the same strategy would resend earlier than Adam). The strategy was producing good results (he was receiving high scores) in a communication environment characterized by a low and constant traffic level, with no sudden variations; this communication class was being observed in the last 50 seasons (that far, the duration of the current state). Adam's satisfaction level was high and a change of strategy was not in consideration.

After ten more seasons had passed, the latest short memory block was analysed and some jumps were detected. Initially, detection of this new condition caused only a small decrease of the satisfaction level; but when it was confirmed by two additional short memory blocks and reinforced by low scoring (the adopted strategy was now producing bad results), it motivated a quick decrease of the satisfaction level that led to a strategy change decision. If Adam were less dynamic, the decrease of the satisfaction level would have been slower and such decision would probably take a longer time to occur.

Following the strategy change decision, a new training example was stored, describing the good results of the previous strategy under the previous communication class (low and constant traffic level, with no sudden variations). If the same conditions were met in the future, this information could then be helpful for the selection of a tuning strategy.

Considering the latest short memory blocks, a new communication class was predicted: low and alternating traffic level, with jumps. Since Adam had no memory of operating under such conditions, he couldn't rely on previous experience to select a new tuning strategy. So, putting aside a random selection alternative, he decided to consult the blackboard. Understanding that Amy, a very successful agent that had been receiving the highest scores, had detected the same communication class, Adam selected the same tuning strategy that she was using, and then begun a new state.

Even if Amy's strategy were a bad choice for Adam, he would have in the future (when the same communication class were detected and a random selection of strategy decided) the opportunity for testing other strategies (explore other possibilities) and find out which one would serve him better in this situation. Moreover, he would (given enough opportunities) eventually select untested strategies even if a good strategy were already found (this would allow him to escape local minimums in local search). However, if Amy's strategy were a good choice for Adam, it would allow him not only to perform better but also to accelerate his learning effort (a good reference point in terms of tuning strategy would allow him to quickly discard worst alternatives).

4 Tests and results

4.1 Introduction

Using a simulation workbench for the group communication problem described, a significant number of tests were made. In this section we describe the most relevant of those tests and discuss their results. To support these tests, several traffic variation functions were created. Some reflect traffic variation patterns as they are found in real communication situations; others set interesting situations that help the evaluation of the different aspects of the agent architecture.

Each simulation is composed by a sequence of one thousand communication seasons. Each test is composed by a sequence of five hundred simulations. The average of the agents' added distances to the optimum instants (from hereon referred simply as *distance*) is the value chosen to express the results.

4.2 Tuning strategies and cognitive system

The initial tests were focused on the tuning strategies. The original set of strategies was tested separately (no cognition or learning) under different traffic conditions. These tests led to the introduction of additional strategies (to cover important specific situations) and to an important (and expected) conclusion: different communication classes have different more adequate strategies.

The tests made to the cognitive system allowed its progressive refinement. In its final version, the system was able to classify the communication conditions in a very satisfactory manner. The possibility of capturing the essential aspects of a complex real-time environment in a classification system opened the door to the introduction of the learning mechanism.

4.3 Learning mechanism

The learning mechanism produced a clear and important result: the progressive decrease of the *distance*. The more complex the traffic variation function is (in other words, the greater the number of communication classes needed to capture the evolution of traffic conditions), the slower is this decrease.

In simple situations, where a single tuning strategy is highly adequate to the traffic function, the learning curve tends to approach the results that would be obtained if only that strategy was used[7] (figure 6). In more complex situations, when the diversity of the traffic variation function appeals to the alternate use of two or more strategies, the advantage of the learning mechanism becomes evident (figure 7).

Figure 7 emphasizes an important result: the alternated use of different strategies can match the diversity of communication classes, producing, in some situations, better results than those produced by any single strategy available.

To confirm this result, special tests were made. Three different sets of available tuning strategies were considered in these tests: *set A* included the best strategy (the one producing better global results if used in exclusivity on the chosen context) and four other strategies (chosen randomly); *set B* included the five remaining strategies; a *full set* always included all ten. In each test, each of these sets was used separately and the

[7] When we mention that only a single strategy is used, we mean that the same tuning procedure is kept throughout the simulations. In these cases, the learning mechanism remains inactive.

respective results were compared. These tests showed clearly that, in some situations, the diversity of strategies is more important then the global fitness of any particular strategy. The *full set* often produced the best results (especially on complex traffic variation contexts) and, in some cases (as the one in figure 8), *set A* produced the worst results (even though it contained the best strategy).

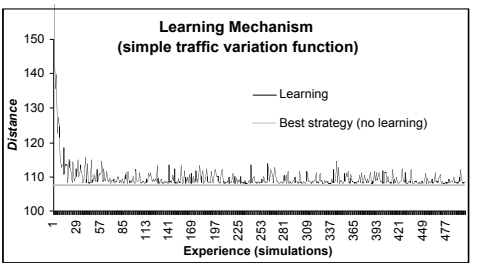


Figure 6: *Learning in a simple traffic variation context*

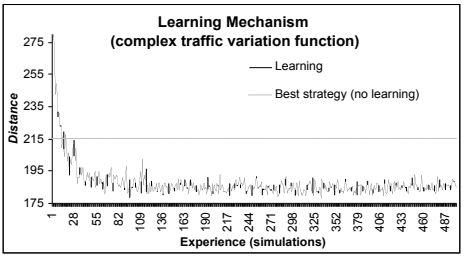


Figure 7: *Learning in a complex traffic variation context*

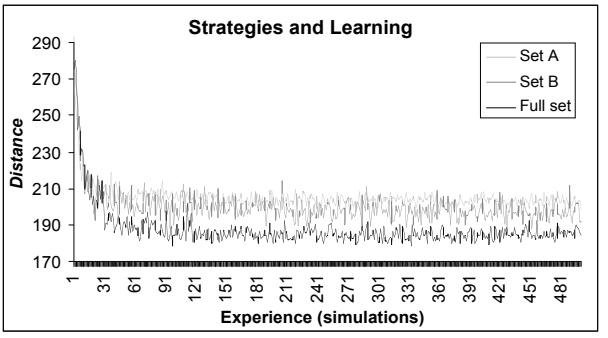


Figure 8: *Comparing the performance of different sets of strategies on the same traffic variation context*

From these results emerges the idea that, in certain traffic variation contexts, there is no single strategy that can be used to match the performance of alternating a set of different strategies.

4.4 Optimism and dynamism levels

Various tests showed that the optimism level could clearly influence the agents' performance. When the traffic level has a low variation or while it continuously decreases,

pessimistic agents usually have a better performance; when the traffic level has a high variation or while it continuously increases, an optimistic posture tends to be better.
The next two figures show the results of testing five different sets of agents grouped by their optimism levels (all having neutral dynamism levels). Figure 9 refers to testing on a traffic variation context where the traffic level predominantly increases: periods of traffic increase are intercalated with abrupt and instantaneous traffic level decreases, producing a typical sawtoothed pattern. Since optimistic agents tend to delay their resending instants, they have better chances to avoid unnecessary resending under such conditions[8] and achieve a better performance.

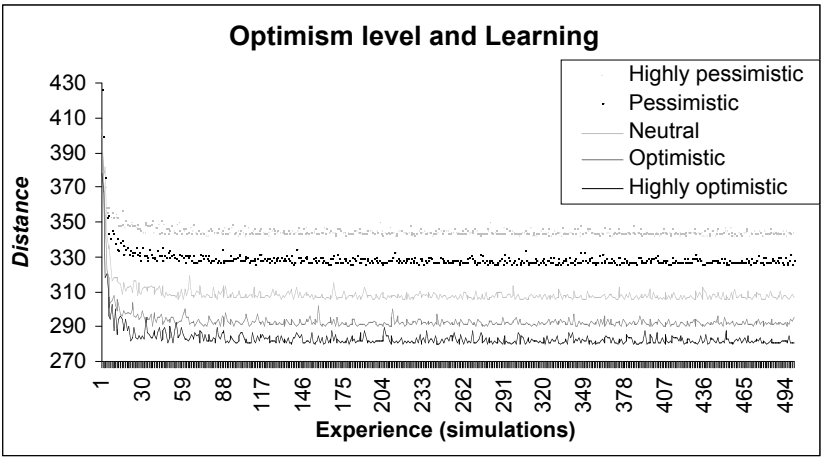


Figure 9: *The influence of the optimism level (sawtooth traffic variation pattern)*

When a traffic variation context in which there is no predominant variation is considered, high optimism or pessimism postures are usually not adjusted (figure 10).

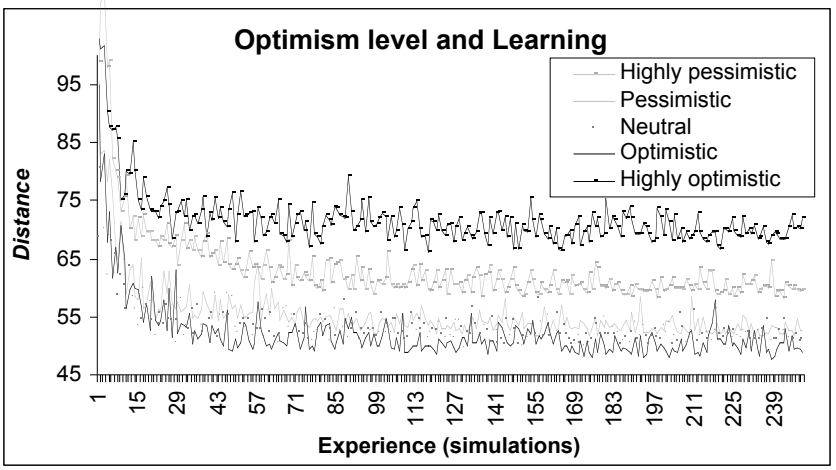


Figure 10: *The influence of the optimism level (traffic context with no predominant variation)*

[8] A pessimistic posture, according to which delays are believed to result from message losses, tends to anticipate the resend. Such a posture is generally penalized when the delay is a consequence of a traffic level increase. In these cases, it pays off to wait a little longer before resending (optimistic posture).

As is evidenced by figure 10, the relation between the optimism level and performance can be complex: although an optimistic posture produces the best results, a highly optimistic posture is worse than a highly pessimistic posture.

The effect of the dynamism level on the agents' performance wasn't made clear by the tests. It was observed that extremely dynamic agents had more difficulty in learning (they eventually achieved the same *distance* of others but took more time to do so).

4.5 Communication between agents

As described before, the agents may use a simple communication method (blackboard) as an alternative way of selecting a new strategy. To test and evaluate the impact of introducing this alternative, we considered two different sets of agents: agents that only used their individual experience table (no communication), and agents that alternated between their experience table and the blackboard. Results showed that the alternation between consulting the experience table and consulting the blackboard could improve the agents' performance (figure 11).

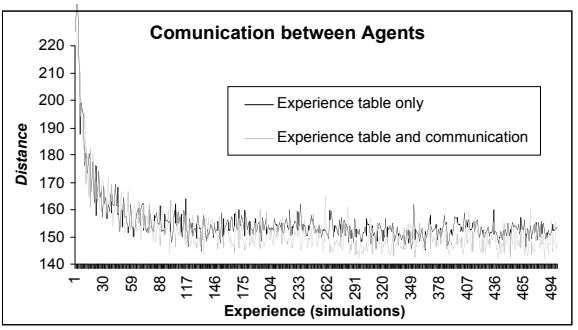


Figure 11: Learning with and without communication

Encouraged by this observation, new tests were made in order to compare the performance of a set of agents that learn without using the blackboard with the performance of an opportunistic agent that only uses the blackboard (only uses the experience of others and doesn't learn by itself). These tests showed that, regardless of the traffic function in use, the opportunistic agent's performance was never worse then the learning agents' performance. Moreover, when complex traffic variation functions were used, the opportunistic agent clearly beat the agents that were able to learn (figure 12).

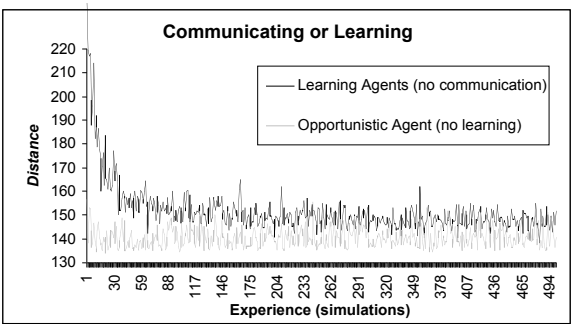


Figure 12: *Performance of an opportunistic agent*

It is important to notice that, in this case, the opportunistic agent achieves a clearly better performance since the first simulation, and reaches its best performance level within the first ten simulations. This is a clear sign that the learning task could be optimised, that the interactions between agents (namely the knowledge sharing) can improve global performance (even at early stages), and that having agents with different tasks or roles can benefit the collective effort and enhance the results.

5 Conclusions

The good results achieved with the proposed agent architecture in a simulated group communication problem showed its adequacy to a real-time environment. Reacting through the use of different tuning strategies, classifying the environment through a cognitive system, and progressively determining the fitness of those strategies to specific communication classes, the agents were able to significantly improve their performance. Furthermore, in some situations, the alternation of strategies allowed by the learning mechanism achieved results that were clearly superior to those obtainable using a single strategy, leaving the idea that having an expressive variety of options can be a good way of addressing the complexity and dynamism of the environment.

The optimism and dynamism levels added versatility and adaptability to the agents. The optimism level revealed a special relevance, significantly influencing the agents' performance in various situations. The accurate characterization of these situations could motivate the online variation of this level, allowing increased adaptation to the environment.

The use of the blackboard as a knowledge sharing method improved overall performance. Furthermore, especially under complex traffic variation functions, opportunistic non-learning agents had better performance than learning non-communicating agents.

6 Final discussion

The success achieved by opportunistic agents indicates that it would be interesting and potentially useful to study the use of a mixed agent society in which the existence of different roles could lead to an improvement of collective performance. More than that, the members of this society could be exchanged according to their performance (from time to time, the worst active agents would be replaced) or even according to the collective experience (for example, under unknown conditions agents with the ability of learning would be preferred, but under well known conditions the opportunistic agents would become predominant).

The study of ways of coordination and interaction between agents to optimise the learning task is a promising field of development of the proposed architecture towards further improvement of global performance. The expressive results obtained with a simple communication mechanism suggest that an additional effort towards basic coordination could easily introduce a distributed learning perspective into the proposed model. This, along with the introduction of specific agent roles, could allow the reduction of the collective learning cost.

The generic properties of a multi-agent system successfully matched the generic problems found in a typical telecommunication problem, reinforcing the idea that the affinities between Distributed Systems and Distributed Artificial Intelligence justify further research. Globally, more than showing the utility of the proposed agent architecture to the problem in question, the encouraging results indicate that the generic

model is a potentially adequate solution for similar problems, namely for those where a real-time environment constantly demands immediate reactions and continuously appeals for cognition.

To help to put in perspective the generic aspects of the architecture, consider the following real-time communication problem. Imagine a multimedia conference where it is important that the participants keep visual contact with each other. During the conference, the video image frames are continuously transmitted on a communication system prone to traffic fluctuations. This problem is also concerned with the imperfection of the telecommunication system in a group communication situation. In this case it becomes important to set an adequate frame transmission rate so that the video image's quality is as good as possible (it is expected and possibly unavoidable that on high traffic situations this quality decreases, being advisable to decrease the frame transmission rate so that congestion doesn't worsen). To apply the proposed agent architecture to this problem, a set of *transmission strategies* (for example, frame repetition strategies, frame skipping strategies, fixed transmission rate, etc.) and a method of performance evaluation (based on the quality of the video image) would have to be defined. Other than that, the essential aspects of the architecture would be easily applicable.

On a first glance, and as an example of a problem belonging to a different area of study (not centred on the communication process), our architecture seems to match the generic properties of the Robocup environment. Robocup sets a constantly changing environment that requires real-time responsiveness and, at the same time, strongly appeals for cognition. The alternative ways of reacting (running towards the ball, stopping, shooting at goal, passing, etc.) could be converted into strategies, and a learning mechanism could progressively determine their fitness to specific states of the environment. To determine the real extent of this simplistic and superficial analogy, further investigation is obviously required.

If reaction, cognition and the ability to learn are among the most fundamental aspects of human behaviour, they may well emerge as fundamental aspects of artificial agents that dwell on artificial worlds that become more and more similar to our own.

Acknowledgement

This work was supported by the LabMAC unit of FCT

References

Albayrak, S. (Ed.) (1999). Intelligent agents for telecommunication applications. *Springer*.

Graça, P. R. (2000). Performance of evolutionary agents in a context of group communication. *M. Sc. thesis, Department of Computer Science of the University of Lisbon (in Portuguese).*

Hayzelden, A. L. G., Bigham, J., Wooldridge, M., Cuthbert, L. (Eds.) (1999). Software agents for future communication systems. *Springer.*

Malec, J. (2000). On augmenting reactivity with deliberation in a controlled manner. *In proceedings of the workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems, Fourteenth European Conference on Artificial Intelligence, Berlin. 89-100.*

Mavromichalis, V. K. and Vouros, G. (2000). ICAGENT: Balancing between reactivity and deliberation. *In proceedings of the workshop on Balancing Reactivity and Social*

Deliberation in Multi-Agent Systems, *Fourteenth European Conference on Artificial Intelligence, Berlin. 101-112.*

Peterson, L. L. and Davie, B. S. (1997). Computer networks: a systems approach. *Morgan Kaufmann Publishers.*

Prasad, M. V. N. and Lesser, V. R. (1999). Learning situation-specific coordination in cooperative multi-agent systems. *Autonomous Agents and Multi-Agent Systems,* 2:173-207.

Sutton, R. S. and Barto, A. G. (1998). Reinforcement learning: an introduction. *The MIT Press.*

Weiβ, G. (2000). An architectural framework for integrated multiagent planning, reacting, and learning. *In proceedings of the Seventh International Workshop on Agent Theories, Architectures, and Languages, Boston.*

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Designing Agents for a Virtual Marketplace

Kaveh Kamyab, Frank Guerin, Petar Goulev and Ebrahim Mamdani

Department of Electrical and Electronic Engineering,
Imperial College of Science, Technology and Medicine
London, SW7 2BT, United Kingdom
k.kamyab@ic.ac.uk ; f.guerin@ic.ac.uk ;
p.goulev@ic.ac.uk ; e.mamdani@ic.ac.uk

Abstract

This paper is concerned with the design of animated agents for applications in e-commerce virtual marketplaces. This brings challenges such as personalisation, believability, conversational skills and multimodal interaction. We have developed a multi-agent architecture for a virtual sales assistant. Agents use high level abstractions for representing, reasoning and planning human-like dialogs. These include mental abstractions such as belief, desire and intention as well as social abstractions such as role relationships and commitments. We design interactions through the development of protocols which specify conventions of behaviour and endow agents with social awareness. Fuzzy modelling is used to model a user's preferences and subjective perception of product attributes. This synergy between existing technologies has the potential to fulfil the requirements for the next generation of virtual marketplaces.

1 Introduction

A portal is an access point to internet resources and services. The resources and services provided can include, for example, a directory of services, a search facility, news, weather information and stock quotes. Portals also provide personalisation, changing the content and appearance to suit the needs of the user. Current portals such as Yahoo, Excite and Netscape, present the user with a 2D interface. However, a great deal of effort is being placed on developing 3-D virtual environments, for example virtual marketplaces and chat rooms, that the next generation of portals could offer. Users would be able to navigate through this virtual space and enter shops to inspect goods and interact with other users (via their avatar) or with embodied software agents. The technology required to implement these embodied agents includes the realistic animation of synthetic faces and bodies as well as adequate conversational skills. This integration of technologies is being undertaken in the SoNG project (PortalS Of Next Generation). Our project partners have been working on designing the virtual world, the user interface and the facial and body graphics and animation (for synthetic characters). The synthetic characters are driven by agent technology; the design of the agents is the subject of this paper. SoNG is ongoing and is currently in its final integration phase. In this article, we comment on our architecture for the agents, in particular how the inputs to an agent are interpreted according to conversation protocols and appropriate output is generated in the form of speech and animations for the agent. In addition we outline how a user profile is built to model the user's preferences in terms of products and interaction.

Within the virtual marketplace of SoNG, several demonstration applications shall be developed, including a theatre booking system, a clothes shop and a telephone shop. Here we report only on the latter. This is primarily considered to be an online information site, where the user has access to several services, which present typical telecommunication equipment. Products will be presented by an Embodied Conversational sales Agent (ECA). This paper is primarily concerned with the design of ECAs with particular emphasis on providing enhanced interaction to human users.

Animated embodied agents capturing some of the emotional qualities and related behaviours associated with humans are being increasingly employed in the development of a new generation of interfaces for a wide variety of applications (e.g. online tutoring, storytelling, online help, e-commerce and so on). There are many researchers who are advancing this new generation, for example (André et al., 2000), (Bates, 1994), (Lester et al., 1997b) and others. Embodied agents promise to improve the user's subjective experience while interacting with computer applications. These improvements include providing the user with intuitive multimodal interfaces consisting of natural language understanding and generation, augmented by the use of realistic facial and body animations. In addition, the use of affect, "computing which relates to, arises from or deliberately influences emotions" (Picard, 1997), can play a key role in ensuring that such characters are able to generate animation appropriately selected for the emotional context of the interaction. Finally, personalisation of services enables the simplification of the interface by adapting behaviour to the user and presenting the user with information which he or she is genuinely interested in.

The use of embodied agents introduces a whole new set of social implications. In particular, a debate has flared up between advocates of embodied interface agents and advocates of Direct Manipulation (DM) interfaces. The latter maintain that over anthropomorphising can result in the user generating false mental models and false expectations of the actual capabilities of the system (Norman, 1997), (Maes and Shneiderman, 1997), (Dehn and van Mulken, 2000). This is due to a human tendency to respond socially to agents perceiving them as competent and capable (Reeves and Nass, 1996). In addition, DM advocates believe that the use of animated graphics can be more of a *distraction from* rather than an *enhancement of* interaction. This would lead to more time-consuming applications and less user satisfaction. On the other hand, embodied agents may help lower the 'getting started' barrier by allowing for the emulation of human-human communication (Adelson, 1992). Furthermore, some compelling results have been found supporting the positive impact embodied agents have on factors such as likeability, engagingness, perceived intelligence, and many more. The relationship between users and applications is discussed in section 2.1.

Aside from the presumed advantages of using embodied agents, they introduce a multimodality into the interface that provides flexibility to the user, but also introduces considerable design and implementation overheads to the designers of such systems. In particular, we consider contextually aware, believable agents with the capability of conversing with the user via natural language. Thus support for context sensitivity, synchronised face and body animation and Text To Speech (TTS) output and dialogue capabilities must be considered at the design stage. In this paper we describe a design approach which addresses these issues, linking agent behaviour, user dialogue and personalisation. Communication is considered from a social perspective through the design of interaction protocols.

Here is a short description of the content. In section 2, we give a brief overview of the issues arising in the design of agents with personality and personalisation. We also discuss issues of particular relevance in an e-commerce application. In section 3, we give an

overview of the agent architecture employed. In section 4, we describe how interactions are designed, beginning with a storyboard and progressing to an interaction protocol. Section 5 describes the low level implementation details and includes a short walkthrough describing how the various different components interact to produce a behaviour. Section 6 discusses some related work. Finally, section 7 outlines issues that must be tackled in the future and then summarises and draws some conclusions.

2 Agents with Personality and Personalisation Know-How

The use of ECAs in an e-commerce application such as the one developed on SoNG requires designers to consider how agent behaviour must be correlated with the underlying functionality of the e-commerce system. In addition, how will agents' behaviours be generated and how might this effect the user of the system?

The agent's function is to provide users with personalised product suggestions matching their needs. In order to provide this information, the agent must learn about user's preferences in terms of the products available. The agent's behaviour must also be appropriately selected to support the function of a shopkeeper. As shown in section 2.1, there is evidence that a correlation exists between users' perception of agent behaviour (e.g. quality and frequency of interactions and animations) and their liking of and trust [1] in the underlying system. We can think of such behaviour as being a function of the agent's 'personality'. This result may call for the need to personalise the agent's personality to the user's preferences.

2.1 User Modelling and Personalisation

User modelling is a technique used to build and maintain user profiles containing information about the user which is relevant to the application domain. User profiles may contain user data such as name, gender and address, but they may contain what the system presumes are the user preferences, knowledge and goals.

The importance of user modelling emerged throughout the eighties largely from research in dialogue systems. One of the earliest such systems was ELIZA, which responded to keyword matching. Often its human communication partners assumed they were communicating remotely over Teletype with another human (Weizenbaum, 1966). However, ELIZA didn't take context into account, which sometimes caused the system to generate inappropriate responses when it was unable to "understand" the user.

At a later stage in the development of dialogue systems, it was recognised that a system should generate assumptions about the user that may be relevant to the task domain at hand. In particular, these include the user's goals, the plans with which the user intends to achieve these goals and the knowledge or beliefs the user has about the domain. The assumptions were to aid the system to converse more naturally (e.g. to supply additional relevant information, avoid redundancy in answers and explanations and detect the user's wrong beliefs about the domain (Kobsa, 1990)).

Throughout the 70's and 80's, another AI application which generated much interest was Intelligent Tutoring Systems (ITS). The predecessors to ITS, Computer Assisted Instruction (CAI) systems, already recorded and evaluated the student's interactions, but no emphasis was put on educational psychology or, more importantly, the student's level

[1] Here we use the everyday definition of the word trust, i.e. the user's reliance on the ability and integrity of the system.

of expertise. (Sleeman and Brown, 1982) first coined the term ITS to distinguish these systems from the traditional CAIs. ITSs brought the concept of knowledge representation to student modelling to aid the tutoring process.

In addition, research on intelligent help systems, game playing and in the field of Human-Computer Interaction (HCI) also identified the need for user modelling. As a result of such a proliferation of use, user modelling evolved into a discipline in itself. More recently, it has been applied to the broad discipline of personalisation, where services and application content are tailored in order to satisfy user preferences. These include Information Retrieval (IR) and Filtering (IF) systems, customisable interfaces and so on.

Despite its promise to solve the development issues in many disciplines, user modelling came under a lot of criticism in the early 90's for failing to deliver (McCalla, 1992) and this was attributed to a number of causes. Intrusiveness of data gathering methods, privacy issues and failure to take the wider user context into account resulted in systems that often made erroneous assumptions and couldn't win over user confidence.

However, utilizing personalisation and the 'one-to-one' marketing paradigm underlying it, is of great importance for today's businesses (Peppers and Rogers, 1993). For example, good sales assistants have the ability to make accurate assumptions about the user and use the information gathered to tailor their service. Furthermore, regular customers receive an even more personalised service, not only because of their tastes, but because the sales assistant knows how best to interact with them. For instance, a good restaurant waiter is one who knows which is your favourite table and knows what your favourite dish is. However, this knowledge comes from passive observation of the customer's behaviour and it comes with the awareness that some people like to choose their own table and change dish at every visit. In order to provide this quality of service, information must be gathered regarding the customer's tastes, background, personality and habits. Not surprisingly, great advances have been made in the area of personalisation for use on e-commerce websites (e.g. Amazon.com, CDnow, eBay, etc), and there are indications that there are substantial commercial benefits to this (Fink and Kobsa, 2000). Thus, a user-modelling component is an essential part of an e-commerce system such as the SoNG application. However, measures need to be taken to avoid the pitfalls highlighted in previous user modelling systems. In particular, we need to address issues of data collection, privacy and the wider user context.

2.1.1 Data Collection and Privacy

The issues of data collection and privacy are intrinsically related. Acquiring data about a user is not a trivial task. A lot of data can be acquired from a user's conversation and by observing a user's behaviour (Orwant, 1996). (Kay, 1993) identifies three distinct categories of user model information.

1. The information *explicitly given* by the user to the system, either by request or voluntarily. For example the user states that he/she is a novice user of the system.

2. The information gathered by *observation* including monitoring user actions and dialogue history.

3. The information that the user has been *told about* and hence is aware of.

Indeed, requesting information about a user *from* the user can have extremely detrimental effects. Firstly, it may appear to be an invasion of the user's privacy. Users may be wary of how and by whom the information will be used. Secondly, users actually have a poor

ability to describe their own preferences. Often, what they say they like and what they actually like are not the same thing. Finally, as stated in the paradox of the active user (Carroll and Rosson, 1987), people are more motivated to start *using* things than to take the initial time to learn about them or to set up a lot of parameters. So, information should be gathered via the observation of user interaction (what does the user ask for, what does the user need) or when a user explicitly states that he or she likes a specific product attribute.

A non-invasive method of data collection has the advantage that users are less prone to be aware of an invasion of their privacy. This helps maintain a relationship of trust between the user and the agent. In any case, a user will be more willing to reveal information to an entity it trusts rather than one it mistrusts. The user's willingness to give up his/her private information helps a user modelling system, but does not address the problem of privacy. Deeper considerations must be made. Many commercial websites are beginning to self-regulate their own handling of confidential user information by adhering to and contributing to initiatives such as (TRUSTe, 2001) and (P3P, 2001). This shows that privacy must be guaranteed in order to attract customers by establishing relationships of trust.

2.1.2 User Context

Just as the first dialog systems were criticised for not taking user context into account at all, many user modelling systems can be criticised for not taking *enough* of the user context into account. User needs and interests depend on many parameters, including personal and sociological aspects (personality, momentary needs, moods, etc) and these parameters change over time (Hanani et al., 2001). Traditionally, the context modelled has been strictly that relating to the application. In Information Filtering systems, user modelling has been content-based, in ITS systems it has been knowledge-based, in e-commerce systems it has been product-based, and so on. Recently, however, this trend began to curb. For example, the realisation that learning is greatly dependant on motivation and emotions has resulted in the application of Affective User Modelling (AUM) to the area of ITS (Elliot et al., 1999), (Paiva and Martinho, 1999). It is possible that AUM can be applied to e-commerce systems too (a satisfied customer is a happy customer, or more to the point, a happy customer is a satisfied customer), but its use needs to be evaluated. The difficulty lies not only in gathering the data, but also in knowing what to do with it. It is clear, though, that each application domain necessitates its own unique considerations with regard to the content of the user model.

Due to its deeper social implications, the use of embodied agents introduces a new set of dimensions that weren't considered for DM interfaces. These dimensions are bound to have an effect on the user models employed. Many studies have been carried out which attempt to evaluate the effect of likeability, engagingness, perceived intelligence, believability, adaptability and motivational qualities of an embodied agent on a user (Koda and Maes, 1996), (Moon and Nass, 1996), (King and Ohya, 1996), (Lester et al., 1997b), (Charlton et al., 2000). Some compelling results have been found supporting embodied agents. (Koda and Maes, 1996) found that the use of animated faces in an interface improved the likeability of the system and engaged users more, leading them to attribute higher levels of intelligence to the agent. The results were more pronounced if the face used a realistic animation of a human face. (Charlton et al., 2000) found a strong relationship between likeability and perception of intelligence (the more a system is liked, the more it is perceived to be intelligent) indicating a need for personalisation of the embodied agent. In fact, (Moon and Nass, 1996) suggest that users prefer adaptable animated

interfaces that become more similar to the user over time, a result which perhaps stems from the natural human tendency to present oneself in a manner which pleases the person we are interacting with (Sproull et al., 1996). (King and Ohya, 1996) found that animated agents were more likely to be seen as intelligent, but this could give a false impression of the real system intelligence. Take, for example, the SoNG embodied agent. The agent's personality and behaviour can influence the user's perception of it and the service it provides. Thus, as found by (Charlton et al., 2000), if a user likes the character they are interacting with, he or she will consider it to be more intelligent and competent, and as a result the user will trust the agent more. It is thus important to model the user's preferences in terms of personalities and behaviours which they like to interact with.

2.2 Personality

Personality is widely studied in human psychology, yet no single defining theory exists. Two broad approaches are taken when defining personality: explanatory and descriptive. The latter and most popular approach concentrates on creating taxonomies of lexical descriptions of behavioural traits, generally associated with personality, which can help identify individual differences between people. This is referred to as the trait theory. The lexical approach assumes that socially relevant personality characteristics have become encoded in our language (Fujita, 1996), and is hence limited to the description of surface behaviour rather than the underlying mechanisms of personality. Many variations in the trait theory have been studied such as the Big Five (John, 1990) and the Giant Three (Eysenck, 1991). In the SoNG project, we make use of a subset of the Big Five theory, a description of which is given below. For an example of explanatory approaches to defining personality, see (Moffat, 1997).

2.2.1 The Big Five Theory:

The most widely used of the aforementioned theories is the Big Five, which is generally agreed to define the following categories:

1. Extroversion: activity and energy level traits, sociability and emotional expressiveness.
2. Agreeableness: altruism, trust, modesty, prosocial attitudes.
3. Conscientiousness: Impulse control, goal directed behaviour.
4. Neuroticism: emotional stability, anxiety or sadness.
5. Openness: Breadth, Complexity, and depth of an individual's life.

Although the names given to the five categories vary according to different theorists, their meanings remain largely unchanged. (John, 1990) presents a survey of a range of approaches to the Big Five theory and the variations in terms used.

The Big Five theory has been successful in categorising personalities empirically by the use of a personality questionnaire, although its success is only measured relative to other taxonomies of personality. In fact, of 300 adjectives used in the personality check only 112 have been successfully associated with one of the five categories (John, 1990). Thus, large parts of the personality sphere are not considered by the theory. Other problems highlighted by (Fujita, 1996) include high correlation between the five categories and, most importantly, the criticism that the theory may only be the structure of the personality lexicon, and not personality. However, no competing theories have proved to be

any more complete than the Big Five theory. In fact, theories such as the Giant Three are essentially a subset of the Big Five, where Agreeableness and Conscientiousness are replaced by Psychoticism, and Openness – sometimes linked with intelligence – is omitted. In particular, the Big Five theory is a progressive research program, and has been subject to a much greater extent of practical experimentation than other theories, making the Big Five theory the most widely used.

The many taxonomies and theories of personality leave us in the dark when it comes to stating what personality is. According to (Picard, 1997), personality can be thought of as a predisposition governing the likelihood of focused emotional responses. The stance taken on SoNG is that personality accounts for consistent patterns of behaviour (Pervin and John, 1993) and thus has a bearing on the agent's behaviours. We use a subset of the big five traits to describe the personality of embodied agents, which includes Extraversion (activity, sociability and expressiveness), Agreeableness (prosocial attitudes) and Openness (depth of the agent's background encapsulated in a persona). Conscientiousness and Neuroticism are not considered, because the agent must exhibit impulse controlled, emotionally stable behaviour.

3 Architecture Overview

Here we describe the general architecture employed for our agents. There are three different agents involved in a typical interaction with the user: an embodied sales agent, a disembodied search agent and a disembodied database agent. The user sees only one, the embodied sales agent. The user may ask for a product in natural language, the sales agent then sends a request to a search agent who queries one or more database agents and returns with the required information (see Figure 1). The sales agent is the most complex since it will interact with the virtual world to find out about changes in the world (such as a user speaking or moving) and to effect changes (facial and body animations and speech output for the agent's virtual body).

As depicted in Figure 1, the architecture of agents is divided into three parts, which are clearly labelled for the sales agent. Firstly there is the *Observation* part (top left), which is responsible for interpreting natural language from a human user and messages from other agents through the *Event Interpreter*. It also interprets other events in the virtual world such as the movement of the human's avatar. The interpretation of these events is recorded as a change to the agent's *Mental State* (which constitutes the second part). The mental state contains an *Internal State* which uses an explicit representation of beliefs, desires and intentions (Rao and Georgeff, 1992). Among the agent's beliefs are beliefs about the beliefs, desires and intentions of other inhabitants of the virtual world. For example the preferences of the human users (see section 5.1). In addition to representing its internal mental attitudes, an agent also maintains a copy of the *Social State*. The social state represents common knowledge and endows the agent with an awareness of its social context (see section 4.1). Finally, the *Action Execution* part uses the current mental state to decide what actions the agent should do next. In addition to selecting actions, the *Action Planner* may also modify the mental state, for example, it may decide to add a new desire or intention.

The architecture of the search and database agents contains some of the same modules as the sales agents, but does not require those modules which deal with human interaction. The modular architecture allows each of our agents to be built with re-usable components. For example, both embodied and disembodied agents will use the same module for interpreting the meaning of events in the world, but embodied agents will use

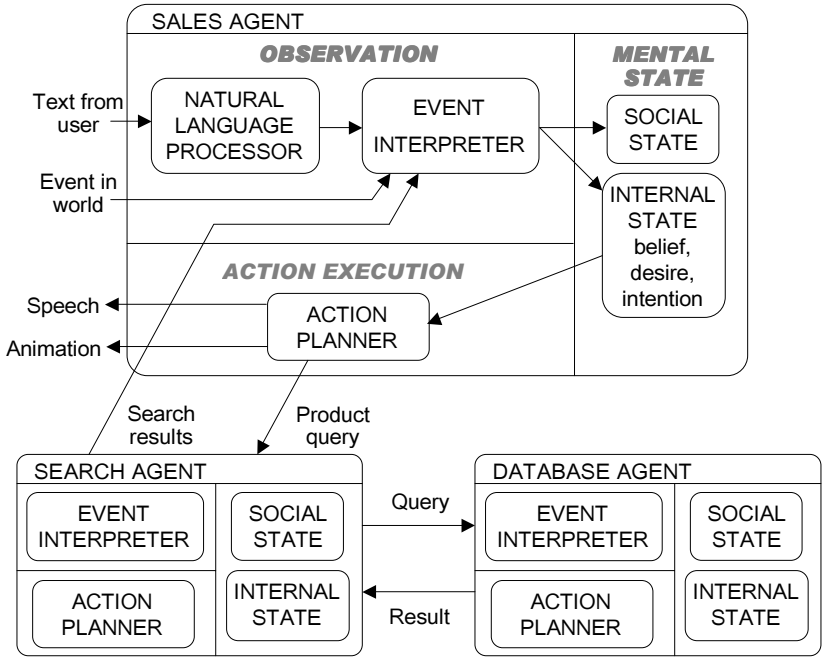


Figure 1: Modular Agent Architectures.

a natural language processor while disembodied ones will not. The agents communicate among themselves using an agent communication language (ACL). The natural language processor translates a natural language sentence into an ACL message so that the sales agent's event interpreter treats this message in the same way as a message from another agent. An aim of this design is to raise the level of agent-agent interactions. An agent will process a message from another synthetic agent in the same way as a message from a human user, considering its effect on mental attitudes and also the social perspective of the society of agents. Thus agent communication involves high level abstractions and is not simply the exchange of procedural directives. This is important since we are designing agents for an e-commerce application, and this technology might be applied in scenarios where different vendors contribute their own database agents. In such competitive markets, self interested agents might not be trusted to give the search agent the lowest price, as discussed in (Singh, 1998).

4 Designing Interactions

Our embodied agents interact with humans in the virtual world by observing natural language text input and responding using both verbal and non-verbal communication. We wish to go beyond a simple stimulus-response behaviour with keyword matching. In order for an artificial agent to be able to communicate effectively with a human, it must be able to interpret the meaning of incoming communications, update its mental state and plan its own communications. In fact, it must understand something of the human-level meaning of the communication. Clearly, achieving a human level understanding is beyond our reach, but we can take some steps towards this goal by identifying certain high level men-

tal abstractions (such as belief, desire and intention) and social abstractions (such as role relationships and commitments) which can be used to characterise the meaning of human communications. Thus the semantics of communications is defined in terms of these high level abstractions. The agent maintains an explicit representation of these abstractions (in its mental state) and reasons using this.

In addition, the interpretation and planning of communication should be sensitive to the context. We concentrate on giving our agents an awareness of three aspects of context as follows:

1. The domain of discourse: The agent will be aware that both the communications it observes and those it generates should be relevant to the shopping domain in which it is situated.

2. The history of the discourse: The agent maintains a record of the discourse, which helps it to interpret and plan communication. For example, if the agent has just proposed to demonstrate a product for the human and the human responds positively, then the agent can interpret this response as an acceptance of the proposal.

3. The social roles occupied by the conversational partners: The agent is aware of its role as a sales assistant and its associated obligations, for example it must greet a human user who enters the shop and it must respond to queries.

4.1 Developing Social Rules for an Interaction

We begin our development by inventing storyboards for interactions. A story can be analysed and generalised to a class of interactions which follow a similar pattern. This pattern constitutes a protocol for the interaction which can be encoded in terms of a set of rules which describe the social conventions governing such interactions. These rules can be encoded logically for our sales agents and will constrain the agent's planning process while it is executing such a conversation. In this way, we simplify the choices available to an agent and ensure that it will behave in a way compatible with the expectations of the human users who are accustomed to dealing with socially aware individuals.

We identify roles for each participant. In our story (see Figure 2), there are two roles: the sales agent is playing the role of *Shopkeeper* and the human user is playing the role of *Customer*. We chart out the paths the conversation can take as shown in the protocol diagram (see Figure 3), taking care to reuse existing states as much as possible. The diagram shows a UML-style statechart diagram where states are states of the conversation (partial social states) and the arcs between them describe the action in the virtual world which effects the state change. The lower part of each state bubble lists actions to be performed whilst in that state.

We now develop rules to describe the effects of communication. It has long been recognised by philosophers that the rules governing communication are *constitutive rules*. Searle describes these rules as follows:

> "Constitutive rules constitute (and also regulate) an activity the existence of which is logically dependent on the rules." (Searle, 1965)

These rules take the form "X counts as Y", for example making a promise counts as an undertaking to do some action simply because this is the convention of usage. Thus it is a social convention that a certain speech act has a certain meaning, and these conventions have evolved over time. It is Searle's contention that:

<user enters shop>	
Phone Agent :	"Good Morning John." "Welcome back to the phone shop." "We have some new phones for you: the Nokia 6110 and the Nokia 3210." "Would you like to see them?"
User :	"Yes, the Nokia 6110 please."
<Demonstration>	
User :	"hhhmm. . . I would like something a little lighter"
Phone Agent :	"How about the Nokia 3330?"
<Demonstration>	
User :	"How much does it cost?"
Phone Agent :	"It costs 150 pounds."
User :	"OK, I'll take it."
Phone Agent :	"One Nokia 3330 has been added to your shopping cart."
User :	"Do you have any matching leather case ?"
Phone Agent :	"No, I'm sorry, we don't sell leather cases." "However, we have some nice faxes."
User :	"No thanks." "Bye!"
Phone Agent :	"Good-bye! I hope you come back soon."

Figure 2: A Storyboard for an Interaction in the Phone Shop.

> ". . . the semantics of a language can be regarded as a series of systems of constitutive rules. . . " (Searle, 1965)

The idea of basing the semantics of communication on conventional meaning is now receiving attention in the field of multi-agent systems (Singh, 1998). The necessity to endow agents with social role awareness (Prendinger and Ishizuka, 2001) and to represent social commitments (Traum and Rickel, 2001) has also been recognised by those designing embodied agents for virtual worlds.

Our approach uses an explicit representation of the *social state* which represents social facts that are currently true. Social facts include commitments and *expressed mental attitudes* (Guerin and Pitt, 2000). The notion of expressed mental attitudes allows us to distinguish between an agent's publicly expressed mental attitudes and its personal internal mental attitudes, which need not be identical. For example, if an agent makes a *request* we will say that the agent has expressed a desire, but we do not really know if the agent has that desire internally. Thus the semantics of our speech acts are defined as social rules which capture the conventional meaning of communicative acts in our virtual society and describe the social relations (for example roles and commitments) that hold between its inhabitants. Table 1 lists the speech acts that will be needed, and their semantics.

The left side of Table 1 gives the speech act name only (we assume the act is successfully performed and contains a sender, receiver and content) and the right side gives the

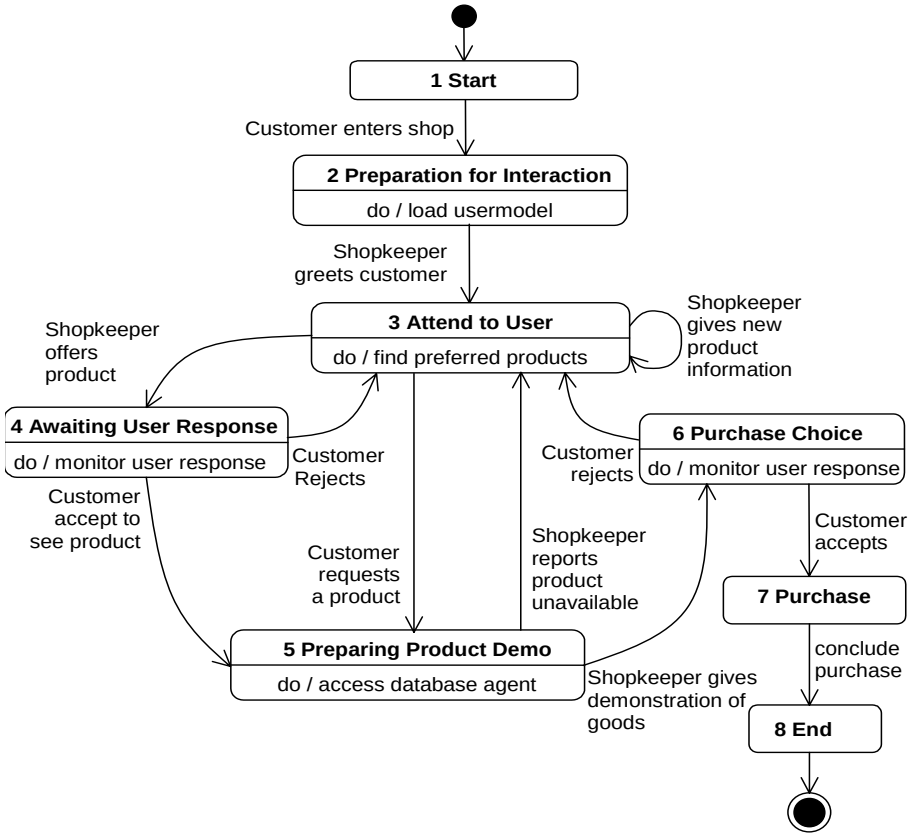


Figure 3: Shopkeeper-Customer Protocol.

Speech Act Name	***Expressed Mental Attitude***
query	E-DESIRE *sender* KNOW *sender content proposition=content*
request	E-DESIRE *sender* DO (*receiver*, *content*)
offer	E-WILLING *sender* DO (*sender*, *content*) E-DESIRE *sender* KNOW *sender* DESIRE *receiver* DO (*sender*, *content*)
inform	E-BELIEF *sender content*
confirm	E-BELIEF *sender proposition*
disconfirm	E-BELIEF *sender* NOT *proposition*
salute	E-HAPPY *sender* DONE (*receiver, enter shop*)

Table 1: Speech Act Semantics

semantics of the act. The expressions on the right are written in a social facts language. E-DESIRE and E-BELIEF denote a publicly expressed desire and belief respectively (the E- prefix stands for *expressed*). E-HAPPY denotes an expression of the state of happiness. E-KNOW

is for knowledge, meaning the agent believes that a proposition is true or that it is false. E-WILLING describes the willingness of an agent x to adopt an intention, provided that another agent y desires it. The semantics describe the facts that are added to the social state after the performance of the speech act. Both the sender and receiver of a message maintain a copy of the social state and update it as messages are sent or received. The state is updated first with the speech act semantics and secondly with the protocol rules (see section 4.2). The terms *sender*, *receiver* and *content* are replaced by their values in the speech act. The **query** speech act adds to the social state an expressed desire of the sender to know if the content is true. It also assigns the term *proposition* a certain value in the social state. When this term occurs in the semantics of an act it is replaced by the existing social fact. Thus the acts **confirm** and **disconfirm** (which need no content) are only meaningful after a **query**. The **offer** speech act adds two propositions to the social state. Firstly, the sender expresses a willingness to do the action specified in the content. Secondly, the sender expresses a desire to know if the receiver desires this action to be done. Where *content* appears within a DO (...) or DONE (...) expression, it is an expression in the action language described in the Appendix, otherwise it is a true or false expression in the social facts language. This type of semantics can be formalised using denotational semantics (Guerin and Pitt, 2001).

Note that the social state is merely a set of propositions which can be represented by text strings. There is no requirement that these propositions are logically consistent and logical implication does not follow automatically, it must be explicitly specified by a rule if desired. The absence of a proposition from the social state does not entail the validity of its negation. Thus if an agent does not believe a proposition or its negation, its attitude with respect to that proposition is in an undefined state. This can be formalised with a three valued logic, but in this paper we focus only on implementing a system. The propositions in the social state will be explicitly used by the protocol rules to define new states when a communication happens.

4.2 Protocol Constraints

The speech act semantics above define speech acts as changes to the social state. The protocol can define a set of commitments for each conversational participant based on the current social state. An analysis of our storyboards leads to the identification of those social facts which are relevant to a description of the state of the conversation and hence constrain the possible future paths. The social facts identified in different storyboards can combine in different ways to describe the social state for a more general class of conversations. Since the total number of possible social states can be large, we restrict our attention to partial states of the conversation. A partial state describes some aspect of the conversation which is relevant to the commitments that arise in a given state. In this way, we generalise from our stories to produce a set of rules which describe the social conventions governing such interactions. These rules are encoded logically for our sales agents and constrain the agent's planning process while it is executing such a conversation. This simplifies the choices available to an agent and attempts to make it behave in a way compatible with the expectations of human users. The rules are described informally in Table 2. The left hand side shows the condition characterising the partial social state and the right hand side describes the social commitments which arise in such partial states. All these commitments are for the Shopkeeper, in this protocol we cannot constrain the actions of the human Customer. Each entry on the right hand side should be prefixed with "COMMITTED *in role:Shopkeeper*" meaning that the agent playing the role of Shopkeeper is committed.

State Condition	*Social Commitments Arising*
Initial (empty) state	greet the Customer AND inform the Customer that this is the phone shop
elapsed time exceeds *limit*	give new product information OR offer to demo a product for the Customer
Customer E-DESIRE to know about the availability of a product	(disconfirm AND offer an alternative) OR (confirm AND demo the product)
Customer E-DESIRE product demo	demo the product
Customer E-DESIRE to purchase	inform that the product was added to cart

Table 2: Speech Act Semantics

Our protocol is initiated by a non-speech action, namely the entrance of a user avatar in the shop. When the interaction begins, there is a blank social state and this creates a commitment for the Shopkeeper to express happiness that the Customer has entered, and to tell the Customer that this is the phone shop. Included in the social state is a timer variable, each agent's copy of this fact is periodically updated by the agent platform. Another social fact *elapsed* keeps track of how much time has elapsed since the last speech act in the conversation. As shown in table 2, if this value exceeds a certain predefined *limit* then the Shopkeeper is committed to perform a speech act so as to avoid a long silence. In some states, the Shopkeeper is committed to more than one action, for example if the Customer queries the availability of a product and the Shopkeeper decides that it is available, then the shopkeeper is committed to confirm the availability and offer to give a demonstration.

5 Implementation

The implementation is built using JADE (JADE, 2000), a FIPA (FIPA, 2000) compliant agent platform that provides the infrastructure for agent communication using the FIPA Agent Communication Language (ACL), although it does not implement the FIPA semantics. SoNG agents implement a cyclic behaviour which involves checking new inputs, followed by an internal planning phase after which chosen actions are executed. To implement internal planning and reasoning we use JESS (JESS, 2000) to build a rule based action planner for each agent. JESS operates on a Knowledge Base (KB), which represents each agent's public data (social model) and private data (beliefs about the state of the world, the user and the discourse, as well as desires and intentions). Each agent is able to make inferences about events in the environment and update its beliefs. Furthermore, each agent makes decisions according to its internal state. A rule base is constructed and loaded into the JESS engine at runtime. A rule is activated if its pre-conditions are true. In addition, the JESS KB can be stored as data in relational databases to maintain product information in the e-commerce application.

5.1 User Model

The use of user modelling in SoNG is apparent in two contexts: the personalisation of content, i.e. which goods match the user's preferences, and the personalisation of inter-

action. Modelling a user's product preferences allows the agent to tailor suggestions to the user's specific requirements, and modelling a user's interaction preferences allows the agent to vary the intensity and quality of conversation and animation. The user model is constructed as a set of JESS facts, which are loaded into the agent's knowledge base at the start of a session. Three categories of facts are stored: firstly, those that represent a user's beliefs, goals and preferences, secondly, those that represent circumstantial information that the system gathered about the user and, finally, those that represent fuzzy variables whose membership functions are determined by the user.

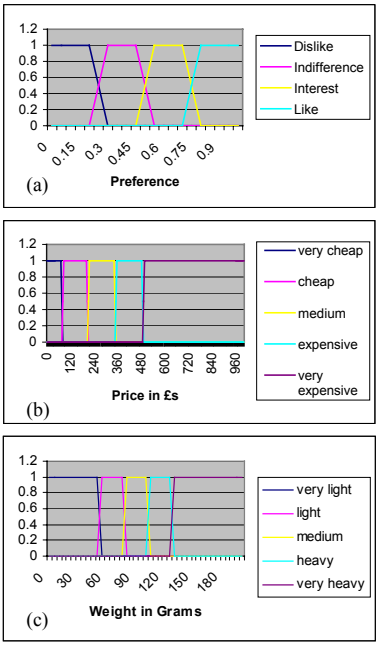


Figure 4: Examples of Fuzzy Set and constituent membership functions for (a) user preference, (b) product price and (c) product weight.

The first category is represented as a set of beliefs that the agent holds about the user. These include beliefs about the user's presumed knowledge of the domain, i.e. products or places that the agent has told the user about during the course of the conversation and hence the user knows exist. Any object that is stored in this category can be referred to directly in conversation, without the need for any introduction. In addition, the user's presumed goals will be stored as well. Finally, the agent holds beliefs about the user's preferences. As the agent's interaction with the user will be mainly through natural language, a linguistic representation of a user's preferences with respect to specific product attributes (e.g. brand, model, price, colour, etc.) was selected.

An ideal candidate for such a representation is a fuzzy variable for which linguistic expressions can be used to describe fuzzy concepts in an English-like manner. Four such terms are used: like, interest, indifference, and dislike (see Figure 4(a)). Two types of preferences are modelled: preferences with respect to particular product attributes and preferences with respect to the personality of the agent. The latter will determine the intensity and quality of the agent interaction (see Section 5.4). Currently, SoNG agents support an explicit model of users' domain specific goals and knowledge and product and personality preferences as detailed in Appendix B.

Updates to a user's product preferences will assign a fuzzy value from the aforementioned fuzzy variable to specific product attributes. When suggesting a product to the user, the agent will select the product with the most "liked" attributes. In the event that more than one product is selected by this method, the agent will compare the remaining attributes of each product, selecting the product with the most attributes of "interest" to the user, and so on. Attributes that the user "dislikes" are not to be recommended.

Another factor that needs to be modelled is the user's interaction pattern. In particular, when was the last time the user visited the shop. Was it recently or did some time elapse. Also, what product did the user buy at his or her last visit, if any. This sort of information will allow the agent to greet the customer with utterances such as: "Hi John, I haven't seen you for a long time. How are you finding the Motorola WAP phone you bought last month?". A user's response will allow the agent to gather feedback about its model of the user's preferences. The second category of facts stored in the user model represents

circumstantial data such as that collected about the user. In particular, three types of information are stored: the user's profile information (e.g. gender, age, nationality, etc.), the time and date of the user's previous sessions, and the products the user has bought in the past. These are represented as per the following example:

```
(user-profile (user-id id001) (name John) (gender male)(age 23)
(address nil) (country UK) (nationality UK))

(session-info (arrival-time "2001-09-19 05:35:26")
(departure-time "2001-09-19 05:37:11") (session-id 6))

(purchase (product-type "mobile") (model "7110") (brand "Nokia")
(price 200) (features "WAP" "large display") (session-id 6))
```

The user profile serves the purpose of identifying possible interaction preferences the user might have (i.e. as a function of gender, age and nationality). Keeping track of the regularity of a user's visit can indicate to the agent the relative satisfaction or dissatisfaction with the service provided. Finally, the purchase information is used to update user's preferences for various attributes of the product.

Additional fuzzy sets are used to model qualitative descriptions of products referring to attributes such as price (ranging between 0 and 1000) and weight (ranging between 0 and 1000 grams). For example, a product can be described by linguistic terms such as "cheap" or "expensive" and "light" or "heavy". Users' perception of the meaning of these qualitative product descriptions may differ and can be modelled by varying the distribution of membership functions within a given fuzzy set. A wealthy user will have a different perception of a "cheap" phone than a poorer user. As the membership functions for these concepts are determined by the user, their fuzzy set data is stored in the user model and loaded at the beginning of each session. Examples of the membership functions can be seen in Figure 4(b) and (c).

Amongst other things, we have identified the need to model the user's preferences about the personality and behaviour exhibited by the agent. The agent's personality is represented in JESS by means of three fuzzy variables, each describing a personality trait (e.g. closed, neutral, open), defined in a JESS fact as follows:

```
(personality (openness "open") (extraversion "extravert")
(agreeableness "agreeable"))
```

This is thus stored as a fact in the user model, to be loaded at the start of a session. Two methods are available for inferring the user preference. Firstly, by monitoring the user's utterances, the agent identifies utterances such as "Go away","You're annoying!" as indications that the user thinks the agent is trying to interact too frequently and in an annoying manner. This will cause a decrease in extraversion and agreeableness in the agent's personality. On the other hand, utterances such as "I need help!" and "You're too grumpy?" will have the opposite effect. Secondly, as it is not necessarily the case that the user will provide the agent with such information, the agent can make use of the Law of Similarity Attraction (Reeves and Nass, 1996). This states that people tend to like personalities similar to their own, in particular when people try to become like them. Thus by monitoring user utterances and their behaviour in the virtual world the agent can attempt to adapt its personality to the user. A user who speaks a lot and moves frenetically around the shop (as a measure of the average number of *position_changed* event notifications per minute) is deemed to be more extraverted and a user who is polite is deemed to be agreeable. Also, a user who is willing to talk about matters outside the shop domain is deemed to be more open. As mentioned above, (Moon and Nass, 1996) found this method of personality adaption to be effective.

5.2 Natural Language Parser

The Natural Language Processor (NLP) is responsible for the accurate recognition and interpretation of a client's utterances. The NLP module creates an ACL message in order to represent the user's communication in a manner compatible with the SoNG specifications (Figure 5). The information flow through NLP is influenced by three main factors - the parser, rules and dictionaries. All of them can be used for tuning the recognition process. The parser, based on the Link Grammar (Lafferty et al., 1992) parser, is the most stable component. However, for some inputs, it can generate multiple outputs. A rule base is used to identify which of these outputs is the most correct for the current conversational context. The products and service dictionaries make the specialisation of the agent possible. Generally speaking, if the NLP is more specialised, it has a better chance of identifying the customers utterances. Therefore, this dictionary has to be built carefully to achieve optimal functionality with a limited number of words. The NLP uses two different methods to recognise human input. The first one is based on ELIZA and uses keyword matching for simple categories such as salutation and confirmation. The main problem with this approach is that context cannot be taken into account. However, it can still be used if, for example, we assume that "yes" always means confirmation and "Hi" is always a salutation. Despite these problems, this method is the fastest and most simple for programming and debugging. For more complex sentences, a rule based system is implemented. For a better understanding, an example is shown on Figure 5 which is based on the story board in Figure 2.

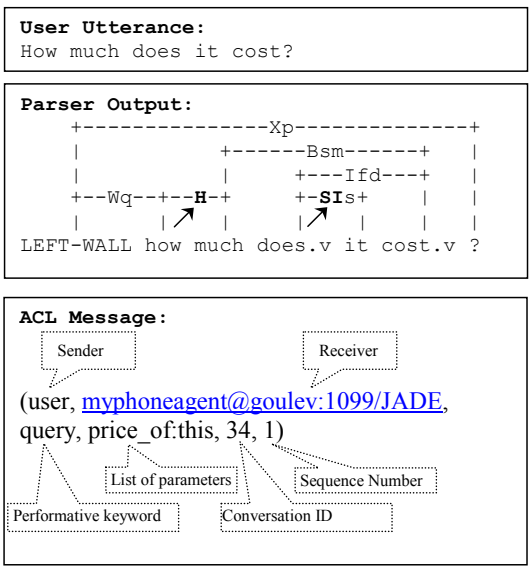


Figure 5: A User Utterance and its Coding Stages.

As shown in the example, the parser recognises six links (Xp, Wq, H, Bsm, Ifd and SIs, the meaning of the different links is beyond the scope of this paper) but we are only interested in two of them. They are marked with arrows and represent a question of type "HOW + ". The linkage "H" connects the word "how" to "much" or "many". "SI" means subject-verb inversion. In order to convert this linkage information to the ACL message displayed in the bottom right corner of Figure 5 we have to define a rule which is activated when both linkages H (associated with "much") and SI are present. The word on the right hand side of the SI linkage is taken to be the subject of the conversation. The rule actually inserts new facts that may be used later. A performative is identified (see Section 1) and a speech act is constructed corresponding to the user input. In some cases, one NL utterance corresponds to more than one speech act.

5.3 Environment Awareness

The SoNG embodied agent is aware of events occurring in its 3D environment. Events are communicated by the SoNG MPEG4 player via an External Authoring Interface (EAI)

(VRML, 1997). The agent is equipped with four sensors, which alert it of events in the environment and some user states. These are:

1. Proximity Sensor: alerts the agent of the position and orientation of the user's avatar with respect to a fixed point in the 3D scene. This enables the agent to face and approach the user's avatar. In addition, it alerts the agent when the user's avatar enters or leaves the shop.

2. Visibility Sensor: alerts the agent when a particular object is in the user's field of view.

3. Collision Sensor: alerts the agent when the user's avatar collides with an object. This is especially useful for detecting when the user's avatar collides with the agent's.

4. Touch Sensor: alerts the agent when the user clicks on a object, for example a phone.

These sensors allow the agent to have awareness of its context. This is helpful in disambiguating utterances received from the user, which refer to objects in the 3D world. For example, when the user refers to "*this* phone", the agent has a knowledge of what object the user is clicking on, and matches that with the reference. Furthermore, the agent can keep track of the user's interaction with objects and movements in the 3D space.

5.4 Action Planner

The agent uses an explicit representation of beliefs, desires and intentions (BDI architecture) to reason about its contextual information and plan communicative acts. In our system, desires and intentions can make use of a language which we have developed for representing agent actions (see Appendix A). We design our agents to respect their social commitments, which constrain the agent's choices and simplify the planning process by providing a limited set of possible actions. The decision making within these choices is made based on the values returned by other modules within the agent, for example the choice of product to suggest to the user is based on the products preferred by the user according to the user model. When the agent has decided what action it will take, the speech and gestures it has chosen (for example pointing) will be performed with the characteristic behaviour of the agent.

In addition, the agent's personality has a bearing on the outcome of the action planner. Its effect is twofold: firstly the agent's levels of extraversion will have a direct effect on the social fact *elapsed* time limit. The more extraverted the agent is, the lower the limit, and viceversa, the less extraverted the agent is, the higher the limit. In addition, within the limit, the frequency of agent utterances also vary in a similar fashion. As the agent's personality is a function of user preferences, it is assumed that higher levels of extraversion correspond to increase help and assistance towards the user.

Secondly, the personality also has an effect on the quality of the agent interaction. An increase in levels of extraversion also increases the degree of emotional expressiveness in the agent's utterances, just as increased agreeableness will increase the politeness. Other than the agent's utterances, personality also has an effect on the intensity of the agent's animation.

When the agent needs to control its embodied representation, the output of the Action Planner – in terms of speech and behaviour scripts – are merged into an Avatar Markup Language (AML) script (Kshirsagar et al., 2002). AML is an XML based format, also developed within SoNG, which allows for the timely synchronisation and merging of face animations, body animations, lip synching and text-to-speech. Furthermore, AML allows for the specification of parameterised behaviour calls such as walking, pointing and facing, which each take 3D coordinates as input. To support this functionality, the AML Processor maintains an internal record of the current position of the agent's avatar in the 3D scene.

5.5 Current Status: System Walkthrough

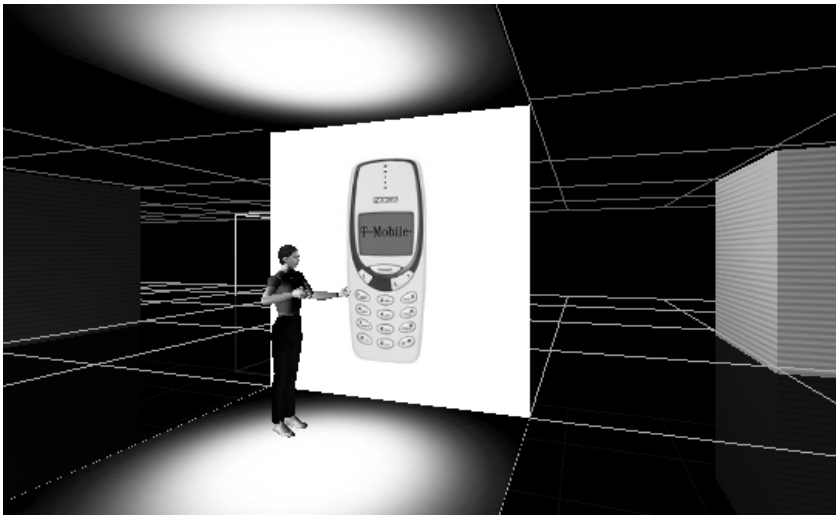


Figure 6: Sales Agent in Phone Shop.

Here we analyse part of a simple interaction which shows how all the modules described above interact to produce the desired behaviour. The first event is the entrance of a user in the phone shop. The event semantics for this initiates an interaction following the `Shopkeeper-Customer` protocol, with the user that entered playing the role of Customer, and the agent playing the role of Shopkeeper. The existence of an empty partial state leads to the creation of a commitment for the Shopkeeper to greet the Customer. A planning rule in the agent is triggered by this commitment and the belief that the agent knows the name of this user, creating the intention to send a speech act which results in "Good morning John" being sent to the output terminal. The agent loads the user model for the Customer that entered, this includes loading the fuzzy preferences of the user. To make it an effective sales agent, the agent has the following desire:

```
(in shopkeeper-customer
    (find list_of: telephone: date:
        last_interaction(in role: customer));
    (tell in role: customer list_of: telephone: price:
        preference(in role: customer))
)
```

This is an action expression. The syntax of this language is given in Appendix A. When the agent is engaged in a Shopkeeper-Customer, protocol the desire is asserted for this specific interaction, giving a composition of two actions to be executed sequentially:

```
(find list_of: telephone: date: last_interaction(in role:customer))
```

The agent playing the role of Customer is substituted and the date of his last interaction is found, giving:

```
(find list_of: telephone: date: 2000-02-27 01:52:27)
```

At present, the agent does not have knowledge about the required subject, but it believes that the search agent does, so it sends a query to the search agent:

```
query,"list_of: telephone: date:2000-02-27 01:52:27"
```

This starts an interaction with the search agent who is now committed to reply to the phone agent. Since it doesn't have the required information itself, it forwards the query to a database agent who it believes knows about phones. In the database agent, the query is asserted and a list of phones is generated, this is then passed back to the search agent who replies to the phone agent with the list as content. This is a list of phones released since the user last visited but not necessarily matching the user's preferences. When the phone agent receives the list of phones, it asserts each phone as a fact, matching each parameter of the phone with the user's fuzzy description for that parameter. For example, where the price is represented by a numerical value in the phone list received from the search agent, the phone agent will assert this as an "expensive" or "cheap" price according to the user's fuzzy terms stored in the user model. Now that the agent has completed the find part of the action expression, it proceeds with:

```
(tell John list_of: telephone: price: preference(John))
```

The agent refines the phone list according to the user's preferences and informs the user of any new phones which match his preferred price range. In the meantime, the user model states that John's last session was more than two months ago. The agent formulates a speech output by applying its personality traits to a template expansion mechanism that selects an appropriate output for the current dialogue context. Finally, the agent fulfils its social obligation and utters:

"Good Morning John. Welcome back to the phone shop. We have some new phones for you: the Nokia 6110 and the Nokia 3210. Would you like to see them?"

6 Related Work

The design of embodied conversational agents is a topic which has attracted a great deal of interest from the research community in recent years. Embodied agents are used in a number of application domains including online tutoring, interactive storytelling, online help, news reading and e-commerce systems. Although these agents take on a variety of forms, some of the underlying techniques are very similar. However, the research emphasis is often quite different.

Peedy the parrot (Ball et al., 1997) is a character developed by Microsoft Research in a project called Persona, to assist a user in selecting and playing musical tracks. Its implementation is based on the Microsoft Agent API, which supports the development of desktop based animated agents. Its developers outline three requirements for assistive

interfaces which motivate the work on Persona: 1) "Support interactive give and take" through support for mixed initiative interaction. 2) "Recognise the cost of interaction and delay". An assistive interface should only interact with the user if the cost of making a mistake is higher than that of interrupting the user. Finally, 3) "Acknowledge the social and emotional aspects of interaction". This is just like human assistants learning what the appropriate behaviour is depending on time of day, task and the mood of the people they are interacting with.

REA (Cassell et al., 1999) is a fully immersed animated agent that acts as a female real estate advisor. It is built on experience gained from Gandalf and Animated Conversation and combines the qualities of both those systems. In particular, aside from generating verbal and non-verbal output and supporting turn taking, REA is able to recognise verbal and non-verbal input directly from the human user by means of sensors. In addition, REA is able to give signals that indicate the state of a conversation. Empirical studies have been carried out to identify appropriate signals (Cassell et al., 2001).

REA's architecture is based on the Functions, Modalities, Timing and Behaviour (FMTB) conversational framework. Here (conversational) functions can be of two types: interactional, i.e. serving the purpose of regulating conversation, and propositional, i.e. functions that convey the content of the conversation. Modalities include speech as well as facial animations and gestures such as hand waves and head movements. Timing highlights the need for prompt response as well good synchrony between output behaviours. Finally, behaviours are identified as those building blocks used to convey conversational functions. Many other systems have attempted to map gestures and animations to conversational functions. COSMO (Lester et al., 1997a), for example, uses pointing as a means of resolving ambiguity in conversations when an object is referred to in the environment.

Although most efforts have been spent on studying the relationship between animated agents and human users, some research has looked at the relationship between multiple animated agents conversing and interacting with each other. Indeed, (André et al., 2000) have been looking at using teams of animated agents in a car sales scenario interacting with each other and a human user simultaneously. This metaphor enforces the need for explicit models of social roles and personalities in order to vary the behaviour of the participants in the conversation.

7 Conclusion and Future Work

The work described here is concerned with the design and development of 3D embodied agents which are capable of carrying out conversations with users and tailoring services to user preferences, where conversations are enriched by meaningful facial expressions and body animations.

We have presented an architecture and implementation for agents situated in a virtual marketplace. We have designed a distributed modular architecture for agents, which means that the various different agents can be built from re-usable blocks. The issue of believable interaction is central to our study. In order to achieve this and create the impression of intelligence, we need to break away from prescripted interaction and develop an architecture which allows emerging behaviour. This is made possible by incorporating a model of personality in the agent, which varies along the dimensions of extraversion, agreeableness and openness.

In our implementation, we have successfully integrated all modules described in this paper to create an architecture for a believable embodied agent. This agent was inserted into a multi-agent system the purpose of which is to provide database search facilities

to the end user. Finally, we interfaced the system with an MPEG4 3D player to which animations are streamed and from which event notifications are received. The contribution of this work is twofold. Firstly, to identify the issues involved in the development of embodied agents for e-commerce applications. Secondly, to demonstrate a design which effectively tackles these issues.

Now that the system is fully integrated, we must study the effectiveness of this type of system on real users. In particular, tests will be carried out to evaluate the use of personalisation and personality traits to adapt the behaviour of the embodied agent to the user. Also, in the next stage of development we plan to investigate the possibilities of enriching the agent's personality model by developing it on top of an explicit emotion engine. By the same token, we need to be able to acquire users' emotional states and we plan to develop the NLP module to allow for this functionality.

Acknowledgements

We are very grateful for the extensive comments given by the anonymous reviewers which led to significant improvements in the paper. This work has been undertaken with the financial support of the SoNG project (IST-1999-10192), part of the EU-funded Information Societies Technology (IST) programme.

References

Adelson, B. (1992). Evocative agents and multi-media interface design. In *Proceedings of the UIST'92 (ACM SIGGRAPH Symp. on User Interface Software and Technology), Monterey*, pages p352–356.

André, E., Klesen, M., Gebhard, P., Allen, S., and Rist, T. (2000). Integrating models of personality and emotions into lifelike characters. In *Proceedings of Affect in Interactions*. Springer-Verlag.

Ball, G., Ling, D., Kurlander, D., Miller, J., Pugh, D., Skelly, T., Stanosky, S., Thiel, D., and Wax, T. (1997). Lifelike computer characters: The persona project at microsoft research. In Bradshaw, A., editor, *Software Agents*, pages p191–222. The AAAI Press/The MIT Press, Cambridge, MA.

Bates, J. (1994). The role of emotion in believable agents. *Communications of the ACM*, 37(7):122–125.

Carroll, J. and Rosson, M. (1987). The paradox of the active user. In Carroll, J., editor, *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. MIT Press, Cambridge, MA.

Cassell, J., Bickmore, T., Camphell, L., Vilhjalmsson, H., and Yan, H. (1999). The human conversation as a system framework: Designing embodied conversational agents. In Cassell, J., editor, *Embodied Conversational Agents*. MIT Press.

Cassell, J., Nakano, Y., Bickmore, T., Sidner, C., and Rich, C. (2001). Annotating and generating posture from discourse structure in embodied conversational agents. In *Proceedings of the Workshop on Multimodal Communication and Context in Embodied Agents, Autonomous Agents*.

Charlton, P., Kamyab, K., and Fehin, P. (2000). Evaluating explicit models of affective interactions. In *Proceedings of the Workshop on Communicative Agents in Intelligent Virtual Environments*.

Dehn, D. M. and van Mulken, S. (2000). The impact of animated interface agents: a review of empirical research. *International Journal of Human-Computer Studies*, 52:p1–22.

Elliot, C., Lester, J., and Rickel, J. (1999). Lifelike pedagogical agents and affective computing: an exploratory synthesis. In Wooldridge, M. and Veloso, M., editors, *AI Today*.

Eysenck, H. (1991). Dimensions of personality. *Personality and Individual Differences*, 12(8).

Fink, J. and Kobsa, A. (2000). A review and analysis of commercial user modeling servers for personalization on the world wide web. *User Modeling and User-Adapted Interaction*, 10:p209–249.

FIPA (2000). Fipa specification. In *Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa2000.tar.gz*.

Fujita, F. (1996). The big five taxonomy, http://www.iusb.edu/%7Effujita/Documents/big5.html.

Guerin, F. and Pitt, J. (2000). A semantic framework for specifying agent communication languages. In *Fourth International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 395–396. IEEE Computer Society, Los Alamitos, California.

Guerin, F. and Pitt, J. (2001). Denotational semantics for agent communication languages. In *Autonomous Agents 2001, Montreal*, pages 497–504. ACM Press.

Hanani, U., Shapira, B., and Shoval, P. (2001). Information filtering: Overview of issues, research and systems. *User Modeling and User-Adapted Interaction*, 11:p203–259.

JADE (2000). Jade project home page available at http://sharon.cselt.it/projects/jade.

JESS (2000). JESS, the Java Expert System Shell. Sandia National Laboratories. http://herzberg.ca.sandia.gov/jess/.

John, O. P. (1990). The big five factor taxonomy: Dimensions of personality in the natural language and in questionnaires. In Pervin, L., editor, *Handbook of personality: Theory and research*.

Kay, J. (1993). Pragmatic user modelling for adaptive interfaces. In M. Schneider-Hufschmidt, T. K. and Malinowski, U., editors, *Adaptive user Interfaces: Principles and Practice*.

King, J. and Ohya, J. (1996). The representation of agents: Anthropomorphism, agency and intelligence. In *Proceedings of CHI'96*.

Kobsa, A. (1990). User modelling in dialog systems: Potentials and hazards. *AI and Society*, 4:p214–240.

Koda, T. and Maes, P. (1996). Agents with faces: The effects of personification of agents. In *Proceedings of HCI'96*, London.

Kshirsagar, S., Guye-Vuillème, A., and Kamyab, K. (2002). Avatar markup language, submitted to the 7th international conference on 3d web technology, web3d 2002.

Lafferty, J., Sleator, D., and Temperley, D. (1992). Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the AAAI Conference on Probabilistic Approaches to Natural Language, October*.

Lester, J., Voerman, J., Towns, S., and Callaway, C. (1997a). Cosmo - a life-like animated agent with deictic believability. In *Working notes of the IJCAI workshop on animated agents: Making Them Intelligent*.

Lester, J. C., Converse, S. A., Kahler, S. H., Barlow, S. T., Stone, B. A., and Bhogal, R. S. (1997b). The persona effect: Affective impact of animated pedagogical agents. In *CHI*, pages 359–366.

Maes, P. and Shneiderman, B. (1997). Direct manipulation vs. interface agents: A debate. *Interactions*, 4(6).

McCalla, G. (1992). The central importance of student modelling to intelligent tutoring. In Costa, E., editor, *New Directions in Intelligent Tutoring Systems*, pages p108–131. Springer-Verlag, Berlin.

Moffat, D. (1997). Personality parameters and programs. In Trappl, R. and Petta, P., editors, *Creating Personalities for Synthetic Actors: Towards Autonomous Personality Agents*. Springer-Verlag.

Moon, Y. and Nass, M. (1996). Adaptive agents and personality change: Complementarity versus similarity as forms of adaptation. In *Proceedings of CHI'96*.

Norman, D. (1997). How might people interact with agents. In Bradshaw, A., editor, *Software Agents*, pages p49–55. The AAAI Press/The MIT Press, Cambridge, MA.

Orwant, J. (1996). For want of a bit, the user was lost. *IBM Systems Journal*, 35:p398–416.

P3P (2001). Platform for privacy preferences (p3p) project, http://www.w3c.org/p3p.

Paiva, A. and Martinho, C. (1999). A cognitive approach to affective user modeling. In *Proceedings of the Workshop on Affect in Interactions Towards a New Generation of Interfaces, held in conjunction with the 3 rd i3 Annual Conference*.

Peppers, D. and Rogers, M. (1993). *The One-to-One Future*. Doubleday, New York.

Pervin, L. and John, O., editors (1993). *Personality: Theory and Research*. Wiley and Sons.

Picard, R. (1997). *Affective Computing*. The MIT Press.

Prendinger, H. and Ishizuka, M. (2001). Social role awareness in animated agents. In *Autonomous Agents 2001, Montreal*, pages 270–277. ACM Press.

Rao, A. S. and Georgeff, M. P. (1992). An abstract architecture for rational agents. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)*, pages 439–449. Morgan Kaufmann Publishers, San Mateo, CA, USA.

Reeves, B. and Nass, C. (1996). *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*. Cambridge University Press, Cambridge, England.

Searle, J. R. (1965). What is a speech act ? In *Philosophy of Language. edited by A.P. Martinich, Third edition.* Oxford University Press.

Singh, M. (1998). Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47.

Sleeman, D. and Brown, J. (1982). *Intelligent Tutoring Systems*. Academic Press, London, UK.

Sproull, L., Subramani, M., Kiesler, S., Walker, J., and Waters, K. (1996). When the interface is a face. *Human-Computer Interaction*, 11:p97–124.

Traum, D. and Rickel, J. (2001). Embodied agents for multi-party dialogue in immersive virtual worlds. In *Autonomous Agents 2001 Workshop on Multimodal Communication and Context in Embodied Agents, Montreal.*

TRUSTe (2001). Truste, http://www.truste.org.

VRML (1997). Vrml97, http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm.

Weizenbaum, J. (1966). Eliza: A computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9:p36–45.

Appendix A: Action Expression Language

<Action expression>	::= <Actor>, <AE>
<Actor>	::= <who>
<AE>	::= <AE1> ; <AE2> | (if <cond> then <AE>) | (in <prot> <AE>) | (assign var=<value>) | (get position <who>) | (point <coords>) | (walk <coords>) | (tell <who> <what>) | (find <what>) | (demo <object description>) | (buy <object description>) | (sell <object description>)
<cond>	::= (protocol=<prot>) | (var=<value>)
<prot>	::= shopkeeper-customer
<who>	::= role:<role expression> | in role:<role name> | <agent identifier> | me
<role expression>	::= <role name>, <interaction id>
<role name>	::= <string_value>
<interaction id>	::= <number>
<value>	::= <what>
<coords>	::= in front <who> | <number>,<number>
<what>	::= is_there: <object description> | list_of: <object description> | price_of: <object description> |

```
                                    is: <object description> |
<object description>        ::= <object type>: <parameters> |
                                gcomparative this <parameter> |
                                lcomparative this <parameter> |
                                comparative this <parameters> |
                                this
<object type>               ::= this |
                                <product type>
<product type>              ::= mobile |
                                fax |
                                telephone |
                                pda
<parameters>                ::= <parameter>:<parmvalue> +
<parameter>                 ::= manufacturer |
                                model |
                                weight |
                                price |
                                date |
                                colour
<parmvalue>                 ::= <string_value> |
                                last_interaction(<who>)
```

Appendix B: JESS Belief Structure

```
(belief
        (agent User)
        (content <Content>)
)
Content                     ::= exists <Concept>
                                preference <FuzzyPreference> <Concept>
                                goal <Goal> <Concept>

FuzzyPreference             ::= like
                                interest
                                indifference
                                dislike

Goal                        ::= buy
                                enquire
                                see

Concept                     ::= product <product type>
                                manufacturer <string value>
                                price <string value>
                                personality <string value>
```

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Modelling Simple Market Structures in Process Algebras with Locations

Julian Padget

Department of Computer Science, University of Bath
Bath, BA2 7AY, United Kingdom
jap@cs.bath.ac.uk

Abstract

One potential barrier to electronic trading is the almost complete absence of any guarantees, for either trading party, that each will meet their obligations. Trust can be acquired from the use of a well-known name with an established reputation, when people are involved, but if the traders are software agents, names mean very little in the absence of reputation models. Thus, we aim to establish confidence by the use of verifiable trading frameworks, which we call *electronic institutions*, and which specify rules for interaction, roles that (electronic) agents may play, the forms of discourse (via speech acts) in which agents may participate and the obligations that agents may acquire. We are aiming to ground our specification of electronic institutions in process algebras augmented with places and in this paper we examine the specification of some key aspects of a prototypical e-institution using the Seal and (typed Safe) Ambient calculi, in order to compare their properties. Places offer a nice correspondence with some aspects of our modelling approach, but although the type systems do permit the statement of some desirable (static) properties, our needs are not fully met, nor are there yet adequate tools to support our objectives.

1 Introduction

Our long term objective is the construction of electronic trading institutions from formal specifications (Rodríguez et al., 1997; Noriega, 1997) and as one aspect of that, in (Padget and Bradford, 1999; Esteva and Padget, 2000) we applied the π-calculus to the modelling of communication in an idealized auction house. However, although the exercise helped illuminate some aspects of interactions in the institution and lead to design improvements, the approach was ultimately frustrating because of the way in which purely technical issues affected what was supposed to be a high-level specification. Also, as has been widely observed, various technical features of the π-calculus simply do not fit the reality of networked computation[1]. Furthermore, while the conceptual modelling of the institution had evolved as a collection of linked scenes (the so-called performative structure), within each of which agents participated in conversations defined by a set of illocutionary acts formed into prescribed orders, and the movement of agents *between* scenes, these concepts and actions had no counterparts in our process algebraic specification. Hence, the appearance of the Ambient calculus (AC) (Cardelli, 1997), the Seal calculus (SC) (Castagna

[1] Specifically: synchronous communication and the passing of names (channels) imply the need to be able to solve distributed consensus.

and Vitek, 1999) and more generally, the augmentation of the π-calculus with locations (Wojciechowski and Sewell, 2000) seemed to offer new promise for our objective of a sound theoretical basis for electronic institutions, by providing explicit notions of place and mobility. Yet more positive still has been the evolving body of work on typing to establish mobility properties for these algebras, since mobility types offer an avenue to the statement and verification of static accessibility properties.

Our approach to specifying an electronic institution rests on the principle of the norm (North, 1991) which can be described as:

> a principle of right action binding upon the members of a group and serving to *guide*, *control*, or *regulate* proper and acceptable behavior (Merriam-Webster, Inc., 2001).

Norms are the principles that characterize an institution and act both to enable people to recognize an organization as being a particular kind of institution and to know approximately the form of the stylized conversations which typify interaction with that sort of institution. Norms can also be relatively abstract, expressing desires like "fairness" and "honesty" without any indication of how such situations might be achieved or maintained, but successive refinement of a norm, in the context of a particular trading scenario, leads to the definition of policies, then to protocols and eventually, as we aim to demonstrate later in this paper, to a level where process algebra may be employed. What we are expecting is that process algebras will enable us to specify the framework of the institution, being its static structure and topology, the mobility of the agents within it, being restrictions on which agents may go where, and provide the communication infrastructure.

The remainder of the paper is structured as follows:

- in *Institutions and norms* (section 2), we expand on the notion of a norm and the relationship between institutions, norms and organizations and follow with an overview of the FishMarket (Rodríguez et al., 1997) work to date;
- in *Mapping the market* (section 3), we outline some of the issues in realizing a market using the concepts of process algebra with places;
- in *Market structure* (section 4), we get down to the detail of the Seal and Ambient calculus, modelling agent communication and modelling agent mobility.
- in *Related and future work* (section 5) we give a short description of work that has influenced ours, mention that which we have yet to evaluate fully and outline our current activities.

2 Institutions and Norms

Most human interactions are governed by conventions or rules of some sort, having their origins in society or the laws that society has developed. Most of these rules have become so common-place that we follow them and operate with their defining framework while hardly being aware of their existence. As an example, consider the case of one person taking the role of a purchaser in a shop and another taking the role of an employee of the shop. Even from this simple scenario, we have already conveyed a large amount of contextual information, since we have the expectation that the interaction will move through approximately three scenes:

(i) identification of goods/service
(ii) evaluation of options
(iii) conclusion of the transaction

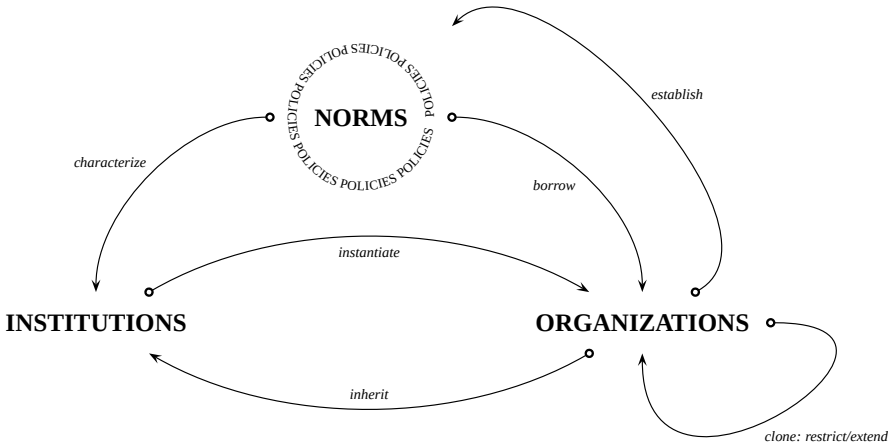


Figure 1: A possible relationship between norms, institutions and organizations

In addition, we have constrained the behaviour of the participants in that we suggested that each is playing a role, which creates expectations in the other party as much as it inhibits the acts of the role-player. But why should each party play a role and follow through a sequence of scenes? We are not qualified, nor do we want to pass opinion on the sociological aspects raised by the question, instead our focus rests on how keeping in character and adhering to the pattern is a means to maintain the over-arching norms of the (generic) trading institution, such as "fairness" and "honesty" mentioned in the previous section. However, as we hinted earlier, norms such as these are unenforceable, so there must be other forces at play which aim to uphold them, and if we were to drill-down to a more detailed, more precise description, we would find all manner of more concrete policies and protocols intended to fulfill those purposes. For example, the purchaser probably has a strong belief that s/he will not be defrauded because there may be a complex set of legal constraints which can be employed should the goods prove unsatisfactory or the service deficient. Likewise, the supplier also benefits from the support of a legal framework that ensures payment. At a lower level, a protocol dictates that the payment, say the swipe of a credit card, may normally only be taken at the conclusion of the transaction (practical exceptions to this appear in hotels or in car rental). Both of these situations are still testament to norms guiding, controlling or regulating acceptable behaviour, although the norms themselves are much more specific and we are inclined to use different terminology, namely *policy* and *protocol* (not that we intend to suggest that three levels is right or sufficient).

The aim of the preceding scenario has been to draw attention to the distinction between the *kind* of trading model (shop) and the *norms* (etc.) that enable it, so that we may take a step back and consider from a wider perspective the relationship between norms, the institutions that are characterized by particular sets of norms and then the organizations that may either be instances of institutions or – by dint of being unique or just few in number – organizations that combine a bespoke set of norms (see figure 1). In addition, to complete the picture, organizations may invent new business models, which may be copied by other organizations and either simplified or built upon further, while other organizations may invent new norms which may be put back into the pool. Observing the distinction between institutions and organizations from a computing perspective, we can

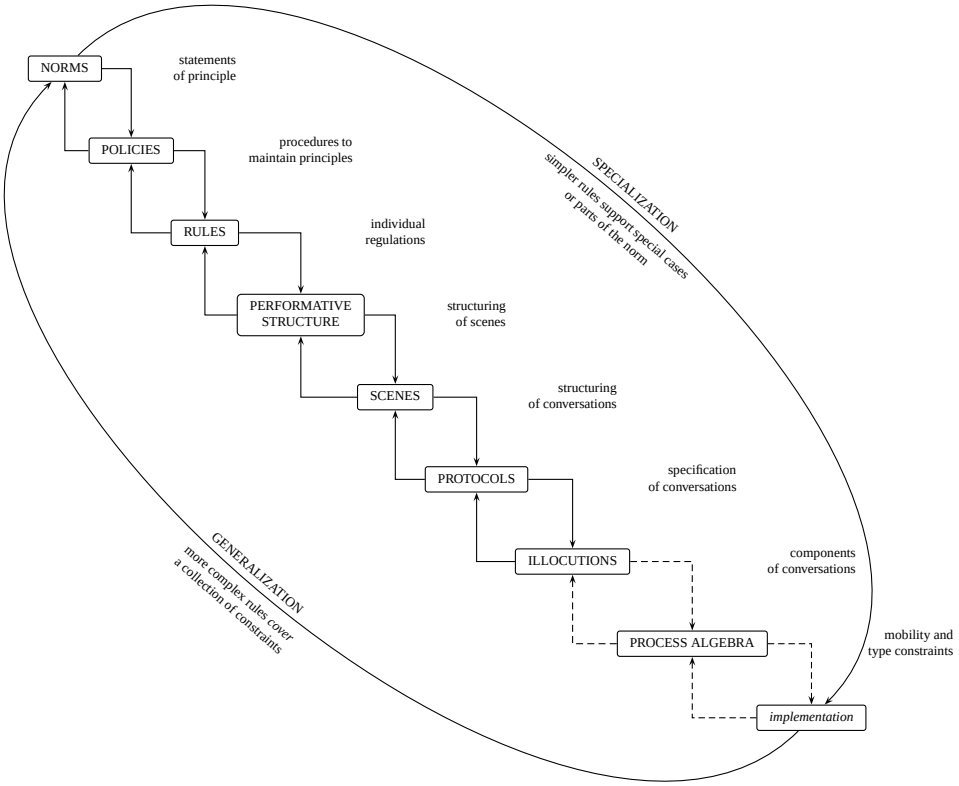


Figure 2: Refinement of norms

appeal to the object-oriented notions of class and instance, respectively, which to some extent mirrors their natures in practice due to their relative speed of change: institutions evolve slowly and change their norms rarely, while organizations tend to be much more dynamic (North, 1991). Finally, we emphasize the point made earlier that some norms are unenforceable, but even if they are not explicitly stated, we can infer their existence from the policies in place which maintain them. Thus it may be the policies that maintain norms which flow through figure 1 as much as the norms themselves.

We conclude this overview of institutions and norms by picking up an earlier theme in which we referred to norms, policies and rules as some of the different levels of detail that might emerge as a result of studying or designing trading structures. The conventions on human interaction typically cover language, meaning and behaviour with a consequential decrease in uncertainty, reduction in conflict of meaning, creation of expectations of outcome and simplification of the decision process by restricting the set of potential actions. As a result of considering these issues, we have broken down interactions into ordered collections of scenes (see for example the three scenes in the shop scenario), which we call a performative structure (see figure 3), scenes into protocols – effectively stylized conversations – and protocols into illocutions (speech acts in the sense of (Searle, 1969)). Thus a kind of hierarchy begins to emerge (see figure 2) which starts out with the norms, policies and rules, then as we descend into more and more detail, getting closer to the implementation, brings in the organizational notions just mentioned. The purpose of this

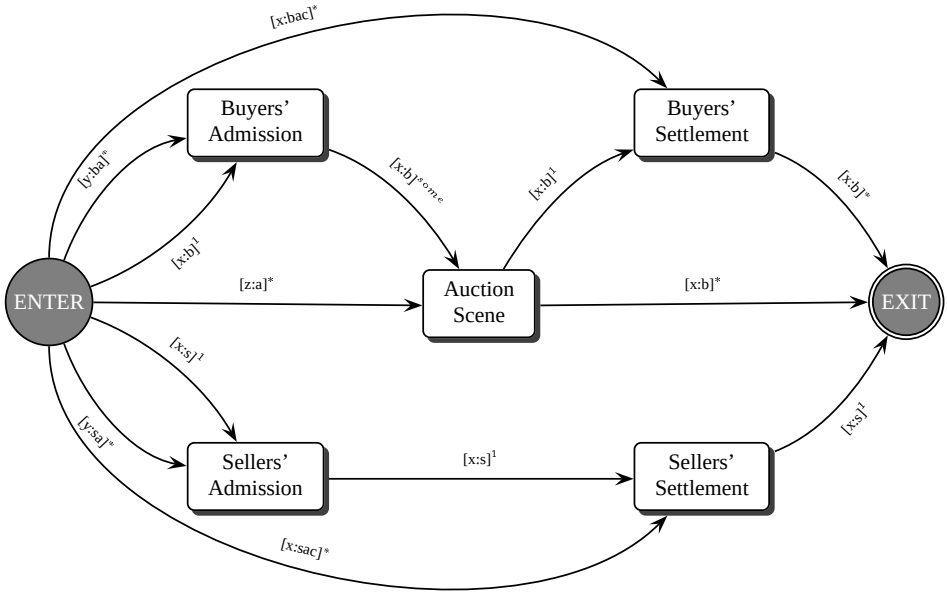


Figure 3: Performative structure of the Fish Market

diagram is to put the process algebra aspect in context and identify how it forms a potential link from the specification in terms of the performative structure, its scenes and its protocols both to a framework for verification and beyond that to a reliable implementation. The third component of our specification after the performative structure and the scenes are the normative rules, being the obligations that agents acquire as a result of their actions within the institutions. A fuller description of our declarative specification language appears in (Esteva et al., 2001).

2.1 FishMarket overview

Much of our work, since we started in 1995, has taken as a reference point the physical market for auctioning fish in the town of Blanes on the Costa Brava. From this physical model we have abstracted what we call *scenes*, for the admission of buyers, the admission of sellers, the auction room (where a standard downward bidding/Dutch auction format is employed), buyers' settlement, sellers' settlement and a back-room accommodating the accountant, quality assessor and other institutional functionaries. For a detailed account of the evolution, see (Rodríguez, 2001).

2.1.1 Performative structure

For each activity that can take place in the institution, there is a corresponding scene, in which interactions between agents are articulated through agent group meetings that follow a well-defined communication protocol – in fact, in our institutions, agents may *only* interact within the context of a scene. This has been described and discussed in detail in (Noriega, 1997), while a more technical treatment appears in (Rodríguez, 2001). This set of scenes and the connections between them – what roles agents may play in them, how many of each role, to which scenes they may move – constitute the *performative*

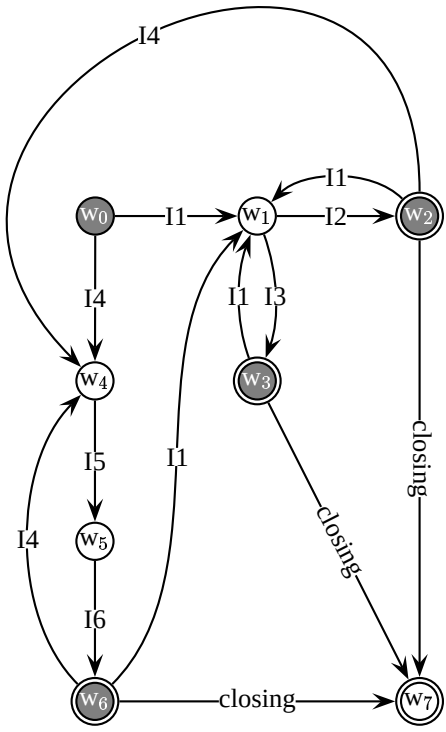
structure for the electronic institution (see Figure 3). The purpose of this diagram is to show the different scenes which comprise the institution by means of a transition graph. Thus, the black circle on the left hand side denotes the start scene and that on the right hand side surrounded by a line is the end scene. In between, there are scenes (rectangles with rounded corners) and arcs connecting them. The arcs are labelled with variable:role pairs, where role (in the case of the FishMarket) may be one of: **a:** auctioneer, **b:** buyer, **ba:** buyers' admitter, **bac:** buyers' accountant, **s:** seller, **sa:** sellers' admitter and **sac:** sellers' accountant. Additionally, it is important to know that the system is intended to be initialized by injecting the (staff) agents via the start scene, whence they traverse the performative structure to reach their allotted scenes. Subsequently, buyers and sellers will also enter the market via the start scene and follow the paths according their choice but under the constraints of the roles they have adopted and under the constraints of any obligations they may have acquired through their actions. To illustrate the last remark, note that a buyer may go either to the buyers' settlement scene or to the exit scene upon leaving the auction scene, but a buyer is only permitted to leave if they have paid for any goods for which they have bid successfully.

We have to admit to a slight simplification in the performative structure shown in figure 3, in that the full specification – as it currently stands – also incorporates so-called *synchronization* scenes between each of the institution-specific scenes. Their purpose is to capture constraints on the numbers and roles of agents traversing the performative structure and temporal constraints such as two agents (playing particular roles) having to be present before a scene can commence. However, since we concentrate on communication and mobility in this paper and do not address these aspects of the specification, here we prefer to keep to the simpler structure. A complete description of the synchronization language appears in (Rodríguez, 2001).

2.1.2 Scenes and conversations

While the objective of the performative structure is to convey the big picture of the institution, the purpose of scenes is to focus the designer's attention on the detail of an interaction limited to one topic. Thus it is that within each scene, we use a transition graph labelled with illocutions to define the structure of a conversation and to identify the states at which an agent may join or leave the scene and which agents may say which illocutions (see the Buyers' Settlement scene in Figure 4). The purpose of this diagram is to formalize what an agent may say, which agents may say what, in what order things may be said and at what points a conversation may begin and end (denoted by the access and exit nodes). It is extremely rigid and intentionally so, because the purpose of the conversation is to exchange or elicit particular information, rather than have a wide-ranging unstructured conversation. For this reason too, we also specify exactly the form of an illocution – there is not even any room for variation here, for the simple reason that it would unnecessarily complicate the design of the institution for, as far as we have been able to see, no qualitative gain.

It is not our intention to make this a tutorial on our notation, not least because it is still evolving, but the labels on the graph do demand some explanation. Each arc is labelled with an illocution, which are *expressions* in the communication language constructed as formulae of the type $(\iota\,(\alpha_i : \rho_i)\,(\alpha_j : \rho_j)\,(\varphi)\,\tau)$ where ι is an *illocutionary particle*, α_i (sender) and α_j (receiver) are terms which can be either agent variables or agent identifiers, ρ_i and ρ_j are terms which can be either role variables or role identifiers, φ is an expression in the *representation language* and τ is a term which can be either a time variable or a time constant. The representation language is a parameter to the dialogic



Where

a denotes an access node

b denotes an exit node

c denotes an access/exit node

and the illocutions are as follows:

I1 `(request (?x b) (?y bac) (update-credit ?value))`

I2 `(accept (!y bac) (!x b) (accept "update-credit"))`

I3 `(deny (!y bac) (!x b) (deny ?deny-code))`

I4 `(request (?x b) (?y bac) statement)`

I5 `(inform (!y bac) (!x b) (statement (?g1 ?p1) ... (?gn ?pn)))`

I6 `(pay (!x b) (!y bac) (payment ?value))`

Figure 4: Conversation graph and illocutions for Buyers' Settlement

specification and might typically be KIF (Genesereth and Fikes, 1992) or FOL (for first order logic). The intention is to allow for the encoding of the knowledge to be exchanged among agents using the vocabulary offered by the ontology. These propositions are then embedded in the language in accordance with speech act theory (Searle, 1969), by means of the *illocutionary particles*. In the example appearing in Figure 4, the particles are `request`, `accept`, `deny`, `inform`, and `pay`. Within the sender/receiver, in the concrete syntax of the example, the variable is annotated with a question mark to indicate a fresh binding (in the spirit of logical variables) or with an exclamation mark to indicate a reference to and comparison with the previous binding of that variable. The role identifiers are the same as those given for the performative structure of figure 3. As two examples, consider (i) I1, I2, which denotes a buyer requesting to update their credit line and receiving acknowledgement (ii) I4, I5, I6, which denotes a buyer requesting their statement of account, receiving it and then paying it. Again, more detail appears in (Esteva et al., 2001).

2.1.3 Governors

A novel feature of our design that is also due some explanation is the use of so-called *governor* agents, which mediate between external agents and the institution. The benefits of using these mediators are in improved security for both traders and the institution and a simplification of the institution itself, both of which are consequences of the software

engineering factors of reduced coupling and increased cohesion, as we now explain. Governors serve several purposes: (i) the governors move around the scenes of the institution on behalf of their external agents because neither do we want to give an external agent potential access to institution internals, nor do we expect that any agent would normally want to put itself in such a position (ii) they may be able to answer questions posed by the external agent about what is permitted or what is expected in different scenes and conversations (iii) they shall ensure adherence to the performative structure and the conversation protocol, thus simplifying the development of the institution as a whole (iv) they may communicate with other governor agents regarding the running of the institution.

3 Mapping the market

The outline in the previous section identifies two levels of specification: the performative structure, which is a linked set of scenes, and the scenes themselves, which are conversation graphs. Hence, we can see that the two levels are not substantially different, but that the levels serve to emphasize structure and hide detail – indeed, we expect to construct a scene library in future work, and that scenes may then be re-used in different electronic institutions. The graphical notation is being developed as a high-level means of capturing the requirements of a market designer, but there is also a corresponding textual representation, described in detail in (Esteva et al., 2001).

3.1 The role of process algebra

The discussion in section 2 explained how process algebra might fit into an hierarchy of norms. We now discuss in some more detail those aspects of process algebra research that might help to achieve those goals.

We have four objectives in trying to use process algebra with locations to formalize the specification of institutions:

(i) To obtain a precise description of the components of the institution and their interactions.
(ii) To provide a formal framework within which the design can be verified.
(iii) To provide a formal framework within which the design can be validated against standard correctness requirements for distributed systems.
(iv) To provide a formal framework within which the design can be animated to verify institutional norms.

However, it is the first and last issues that hold the most interest for us: the first, because without it we can do nothing and the last because we see norms as the key to generating a reputation for and trust in electronic trading institutions. We will now set out how we can relate the components of our institutional specification to the elements of process algebra with locations.

It may be debatable whether it is necessary or desirable to reflect in the process algebra model the two-level structure of scenes and conversations: the institution as a whole could be viewed as the composition of conversation graphs, where the exit nodes of one graph are linked to the access nodes of another, modulo the actions of the synchronization nodes. A first reaction might therefore be to propose the use of Petri nets; this direction is explored and rejected in (Rodríguez, 2001). The attraction of process algebra with locations is that it intuitively supports the idea of associating scenes (see above) with static locations and agents/governors with mobile locations. Also, within a typed framework – but these are still evolving very rapidly – it will be possible to prove that certain locations

are immobile, and that other locations have mobility within certain groups of locations (Castagna and Vitek, 1999; Cardelli et al., 2000; Ghelli et al., 2001). Furthermore, work on typing interactions within locations (exchange types (Cardelli and Gordon, 1999)) enable us to specify what illocutions may be exchanged.

Initially, we thought exchange types were insufficiently precise, because they specify *all* the objects that may be exchanged within a location, whereas we also wanted to make it clear that some illocutions could only be said by one party and heard by another (for example, only the auctioneer should be able to declare a bidding round open). However, the combination of exchange types and polymorphism (Amtoft et al., 2001) seems to resolve this, while adding the notion of *orderly communication* as a sequence of types where the sequence relates to the passage of time. This last approach may also support our aim of prescribing valid sequences of illocutions from an agent's perspective. The conversation graphs provide a means to show which illocutions may follow from one another, but in that sense they are a conversation-centric specification. For an agent, playing a particular role, it only makes sense to say certain things: for example, an auctioneer does not buy in its own auction, nor does a buyer announce new lots; such illocutions are incompatible with their roles. One way to tackle this issue might be through some adaptation of role-based access (RBAC) models (Sandhu et al., 2000) or dynamically typed access control (DTAC) (Tidswell and Jaeger, 2000). A rather different approach might assign types (labels) to every state of the conversation graph and then construct partitions based on possible conversation trajectories. A third strategy we are considering might use the illocution names as types and then examine a trace semantics behaviour of an agent again following the ideas set out in (Amtoft et al., 2001).

We are developing an institution specification language, which is described in (Esteva et al., 2001), but while this is serving to capture various requirements of electronic organizations, various aspects of it need to be grounded in different formalisms appropriate to need. For example, as we noted earlier, logic may form a part of an agent communication language. To ground the computational and mobility aspects, we are investigating process algebra since it could provide:

1. a computational foundation for a distributed implementation, using experimental programming languages like Nomadic Pict (Wojciechowski and Sewell, 2000), JavaSeal (Bryce and Vitek, 1999), or Ambients in JoCaml (Fournet et al., 2000).
2. type systems, which can be used to verify some kinds of norms. For example, *exchange types* might be used for conversations, *mobility types* might be used for accessibility, *sequence types* might be used for analyzing the progression of conversations.
3. logics on top of process algebra combined with model checking to verify higher-level norms. For example: modal logic (Cardelli and Gordon, 2000) can answer whether there is an ambient of type A *below* this ambient; temporal logic can determine whether an agent in scene α will eventually go to scene ω; deontic logic (Lomuscio and Sergot, 2001) can express the idea that an agent whose bid is accepted is obliged to buy the lot.

4 Market structure

The background to our work on institutional modelling, given in section 2.1, drew upon our experience with the Blanes Fish Market and our modelling of that as an electronic institution. Of course, in the process of that modelling, we have abstracted many aspects with the objective of establishing more general principles which can be applied in a variety

of contexts. For the purpose of this discussion we do not attempt to present a complete model of the Fish Market using ambient or seal calculus, but instead focus on exploring the issues surrounding communication (between external agent and governor, governor and governor and governor and staff agent) and mobility (between scenes).

We stated at the outset that one of the objectives of this paper was to compare the properties of the Seal calculus and the Ambient calculus and so in this section we will be sketching specifications of communication and mobility support without reference to any particular institutional structure. Hence we begin with brief sketches of the syntax and semantics of Seal and ambient calculus and then turn to its application, bearing in mind the twin objectives of wanting to specify a computational model and establishing sufficient type information to verify some of the low level norms.

4.1 (Safe) ambient calculus

The two axes of the Ambient calculus (AC) are *communication* and *mobility*. The fundamental unit of AC is the notion of an ambient, which is a place within which processes may interact by writing messages into the ambient and reading them from it – hence reading and writing are decoupled and asynchronous. It is not possible for processes in different ambients to interact. Thus communication is a localized activity and it is only via movement that two processes in different ambients can arrive in the same ambient and hence interact. The unit of mobility is the ambient – not individual processes, but rather, a collection of processes and the messages that may be in transit between them. A process within an ambient may execute a capability to (make the whole ambient) enter a sibling ambient or move out of its enclosing (parent) ambient – these are called objective moves, because it is the ambient that moves itself – or an ambient may be dissolved, unleashing its constituents into the enclosing ambient – a subjective operation, because a process executes a capability in one ambient to carry out the operation on another. The effect of each of these operations is conveniently imagined as reorganizations of a tree (see Figure 7):

- `in` detaches the sub-tree rooted at the moving ambient and re-attaches it as a child of the target ambient
- `out` detaches the sub-tree as above and re-attaches it as a child of the parent of the parent
- `open` attaches all the children of the subject ambient as children of the ambient performing the open

A sequence of mobility operations is called a path or a capability and may be passed as a message from one process to another. Attempting a move, when the target ambient is not present (that is, as sibling or parent) causes the process executing that operation to block until the named ambient appears. However, other processes may continue to execute and the ambient enclosing the blocked process may still undergo other objective or subjective moves.

A summary of the syntax of AC appears in Figure 5, from which it will be observed that we have adopted the extension of co-actions, introduced in Safe Ambients (Levi and Sangiorgi, 2000). Under this variant of AC, for every action (*in*, *out*, *open*) there is a corresponding co-action ($\overline{in}$, $\overline{out}$, $\overline{open}$) and for any action to succeed, the collaborator in the action (a sibling, child or parent ambient, respectively) must engage in the corresponding co-action, that is, both parties to the ambient operation must synchronize. The role of the co-actions is made explicit in the reduction rules of figure 6. For the sake of space, we have omitted the rules for structural congruence that complete the semantics by defining legal reorganizations of terms that bring reacting components together.

P	::=	$\mathbf{0}$	inactivity	α	::=	x	variable (read)
	|	$P \mid P$	composition		|	n	name (new)
	|	$!P$	replication		|	$in\ \alpha$	enter α
	|	$(\nu\ x_1, \ldots, x_n)\ P$	restriction		|	$\overline{in}\ \alpha$	allow enter α
	|	$\alpha\ .\ P$	action		|	$out\ \alpha$	exit α
	|	$n[P]$	ambient		|	$\overline{out}\ \alpha$	allow exit α
	|	(x)	bind input		|	$open\ \alpha$	open α
	|	$\langle\alpha\rangle$	output		|	$\overline{open}\ \alpha$	allow open α
					|	ϵ	empty path
					|	$\alpha\ .\ \alpha'$	path

Figure 5: Ambient calculus syntax from (Cardelli and Gordon, 1999) augmented with co-actions from (Levi and Sangiorgi, 2000)

$$
\begin{array}{rcll}
n[in\ m\ .\ P \mid Q] \mid m[\overline{in}\ m\ .\ R] & \longrightarrow & m[n[P \mid Q] \mid R] & \text{(in)} \\
m[n[out\ m\ .\ P \mid Q] \mid \overline{out}\ m\ .\ R] & \longrightarrow & n[P \mid Q] \mid m[R] & \text{(out)} \\
open\ n\ .\ P \mid n[\overline{out}\ Q] & \longrightarrow & P \mid Q & \text{(open)} \\
(x)\ .\ P \mid \langle\alpha\rangle & \longrightarrow & P\{x \leftarrow \alpha\} & \text{(communication)}
\end{array}
$$

Figure 6: Safe Ambient calculus reduction rules

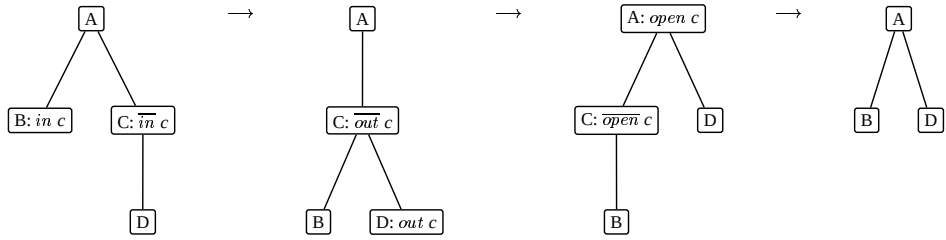


Figure 7: Safe Ambient operations as tree transformations

4.2 Seal calculus

The seal calculus (Castagna and Vitek, 1999) can be viewed as a reaction to the ambient calculus in that its design is based on a strong rejection of one of the key ambient calculus operations, namely *open*. The criticism is that opening an ambient unleashes unknown processes and messages into the ambient doing the opening including capabilities that may move the ambient somewhere else or permit the opening of sub-ambients. The adoption of co-actions, as described in the preceding section, ameliorates the situation, by at least ensuring agreement between the subject and the object of an operation. But in the absence of typing, neither calculus can say anything about the nature of processes in sub-ambients. Thus the key features to note about the seal calculus are: (i) that seals may not be dissolved, thus preventing one major source of security violations in the ambient calculus (ii) that seals may be trapped, since no seal may move without the assistance of its parent (enclosing seal), and also thereby increasing security by preventing autonomous movement.

A summary of SC appears in figure 8 and as with the ambient calculus earlier, this

P	::=	$\mathbf{0}$	inactivity	α	::=	$x^{\eta}(y)$	input
		$P \mid P$	composition			$\overline{x}^{\eta}(y)$	output
		$!P$	replication			$\overline{x}^{\eta}\{y\}$	send
		$(\nu\ x)\ P$	restriction			$x^{\eta}\{y\}$	receive
		$\alpha\ .\ P$	action	η	::=	$*$	local communication
		$x[P]$	seal			$\uparrow$	upward communication
						z	downward (name of subseal)

Figure 8: Seal calculus syntax from (Ghelli et al., 2001)

$$
\begin{array}{rcll}
x^{*}(u)\ .\ P \mid \overline{x}^{*}(v)\ .\ Q & \longrightarrow & P\{u \leftarrow v\} \mid Q & \text{(write local)} \\
x^{y}(u)\ .\ P \mid y[\overline{x}^{\uparrow}(v)\ .\ Q \mid R] & \longrightarrow & P\{u \leftarrow v\} \mid y[Q \mid R] & \text{(write out)} \\
\overline{x}^{y}(v)\ .\ P \mid y[x^{\uparrow}(u)\ .\ Q \mid R] & \longrightarrow & P \mid y[Q\{u \leftarrow v\} \mid R] & \text{(write in)} \\
x^{*}\{u\}\ .\ P \mid \overline{x}^{*}\{v\}\ .\ Q \mid v[R] & \longrightarrow & P \mid u[R] \mid Q & \text{(move local)} \\
x^{y}\{u\}\ .\ P \mid y[\overline{x}^{\uparrow}\{v\}\ .\ Q \mid v[R] \mid s] & \longrightarrow & P \mid u[R] \mid y[Q \mid s] & \text{(move out)} \\
\overline{x}^{y}\{v\}\ .\ P \mid v[R] \mid y[x^{\uparrow}\{u\}\ .\ Q \mid s] & \longrightarrow & P \mid y[Q \mid s \mid u[R]] & \text{(move in)}
\end{array}
$$

Figure 9: Seal calculus reduction rules

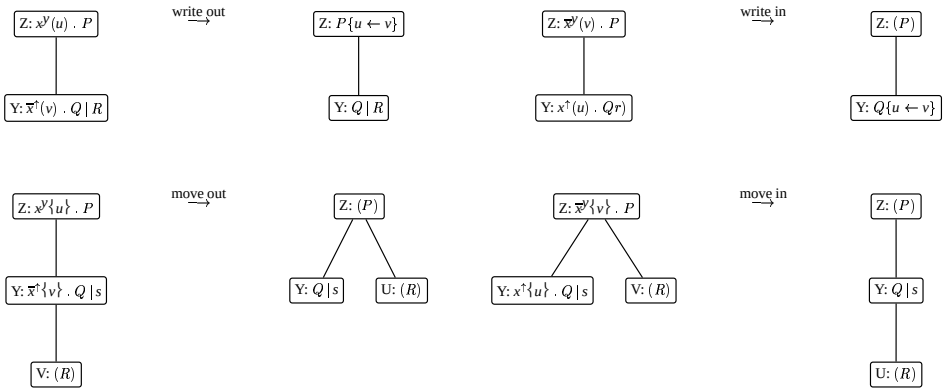


Figure 10: Seal calculus reductions as tree transformations

is followed by the reduction rules in figure 9 minus the structural congruence rules and lastly there are illustrations of the reductions as tree transformations. A point to note is that communication in the Seal calculus is synchronous and channel-based – so too is seal mobility. Furthermore it is important to know that communication and mobility are denoted differently, communication using the notation of parentheses familiar from the π-calculus while mobility uses $\{$ and $\}$. A channel is annotated with a superscript to show whether the communication/move is with the parent, indicated by up-arrow ($\uparrow$), local, indicated by star ($*$) or with a sub-seal, indicated by the name of the sub-seal.

4.3 Agent communication

We now turn back to one of the two problems we identified at the beginning of this section: communication. After considering a number of fixed designs (such as requiring that all scenes be siblings, that is, a flat structure), we concluded it was desirable to abstract

information about scene topology from the governors, which have to communicate with (i) staff agents (ii) other governors (iii) external agents on whose behalf they are acting and essentially borrow a standard solution from networking by having each scene provide a proxy, which we call the scene manager (SM), and whose purpose is to permit location transparent communication between the agents listed above and the governor. The very idea of location transparency seems to clash with the precept of communication being strictly local, but is necessary, both to simplify the logic of the governor – it should not have to know the position of a scene in the hierarchical structure of the institution in order to be able to send a message – and more importantly because the (external) agent should not know that information. There is also the issue of the secrecy of the messages themselves, both between governor and agent and between the agent and the outside, but in both AC and SC, this can readily be resolved by wrapping a message in a (mobile) location, although encryption alone may be enough. More sophisticated solutions are described in (Abadi and Gordon, 1999). We are now in a position to state an informal specification of the market.

The market as constituted consists of six immobile locations: the all-enclosing FISHMARKET, and five inner, sibling locations – BUYERS' ADMISSION, SELLERS' ADMISSION, AUCTION, BUYERS' SETTLEMENT and SELLERS' SETTLEMENT. The mobile agents comprise (i) external agents, which may only enter and exit the FISHMARKET (ii) governors, which may not exit the FISHMARKET, but may enter and exit the five inner locations (iii) the staff agents (the admitter, the auctioneer and the accountant), which may not exit the FISHMARKET and may only enter and exit their corresponding scenes. The governors and the staff agents only communicate using the illocutions defined for a particular scene. External agents may use any illocution, including those defined in the published market ontology (not specified here, but part of the institution definition language we are developing), at any time and in any order – it is the job of the governor to respond as best they can, but to ensure that the defined conversation structure is adhered to within each scene, while attempting to carry out the expressed intentions of the external agent.

Our consideration of communications begins with the following two assumptions: that the external agent will be immobile (or leave an immobile representative/proxy) within the market for the duration of all its transactions and that the market representative – the governor – will move between the scenes of the market, interacting with the staff agents and communicating with the external agent it represents. Thus, an external agent enters a location identified as the FISHMARKET, which creates a GOVERNOR for it, which then moves between the market scenes where the real action takes place, following the performative structure. Hence, at the highest level we could describe the market (independently of seal or ambient calculus) as:

$$\begin{aligned} \text{FISHMARKET} \triangleq\ & \text{BUYERS-ADMISSION} \\ & \mid \text{SELLERS-ADMISSION} \\ & \mid \text{AUCTION-SCENE} \\ & \mid \text{BUYERS-SETTLEMENT} \\ & \mid \text{BUYERS-SETTLEMENT} \end{aligned}$$

Then, within each scene, a scene manager (SM) will fulfill the following requirements:

1. mediation of local one-to-one and broadcast message traffic
2. management of agent transport (addressed in the following subsection)
3. relay of messages between governors and associated external agents

which can be written (and visualized), still independent of choice of calculus, as:

$\textsc{SceneManager}_{\textsc{Seal}} \triangleq$
`repeat {`
$\quad$ *inform*$^{a}(m, b)$
$\quad$. `if` $b \in$ *subseals*
$\quad\quad$ `then` $\overline{\textit{inform}}^{b}(m, a)$
$\quad\quad$ `else` route message m from a to b
$\quad$ | *enter*$^{\uparrow}\{a\}$. add a to routing tables
$\quad$ | *leave*$^{a}\{b\}$. add b to routing tables
$\quad$ | *exit*$^{a}(a)$
$\quad\quad$. delete a from routing tables
$\quad\quad$. $\overline{\textit{leave}}^{\uparrow}\{a\}$
$\quad$ | *broadcast*$^{a}(m)$
$\quad\quad$. $\forall b : (b \in \textit{subseals})\ \&\ (a \neq b), \overline{\textit{inform}}^{b}(m, a)$
`}`

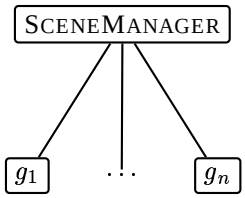
$\textsc{Governor}_{\textsc{Seal}} \triangleq$
`repeat {`
$\quad$ *inform*$^{\uparrow}(m, a)$. process inform
$\quad$ | other agent services
`}`

$\textsc{Governor}_{\textsc{Seal}} = \texttt{Id}^{\frown}[\textit{inform} : \textit{message} \times \textit{name}; \textit{exit} : \textit{name}; \textit{broadcast} : \textit{message} \times \textit{name}]$

$\textsc{Admission}_{\textsc{Seal}} = \texttt{Id}^{\vee}[\textit{enter} : \textsc{Governor}_{\textsc{Seal}}; \textit{leave} : \textsc{Governor}_{\textsc{Seal}}]$

Figure 11: Seal scene manager, governor and types

$\textsc{Scene} \triangleq \textsc{SceneManager}$
$\quad | \ g_1[\textsc{Governor}]$
$\quad | \ \ldots$
$\quad | \ g_n[\textsc{Governor}]$



where g_1 and g_n are the names of the seals/ambients containing governors created to act on behalf of the external agents. A complete specification of such a SM is too long and contains too much irrelevant detail for the space available, so instead we restrict ourselves to examining the interface provided in each case and a sketch of the specification, looking at the infrastructure for scenes and their interaction with governors.

4.3.1 A Seal calculus approach

We have taken some pseudo-language syntax liberties which we hope the reader will forgive – and understand – to provide some simple control structures. As was noted earlier, the seal calculus provides synchronous channels and this leads to a SM that operates over consistently named shared channels to communicate both with sub- and super-seals, offering the following interface (see Figure 11) from the governor's point of view:

- *inform*$^{\uparrow}(m, a)$ is a read operation receiving a tuple containing a message m and an agent identifier a from the scene manager agent.

$\textsc{SceneManager}_{\textsc{Amb}} \triangleq$
`repeat {`
open put
| (`inform`(m, a, b)) . `if` $b \in$ *subambients*
`then` *get*$[\langle$`inform`$(m, a, b)\rangle$. in $b]$
`else` route message m from a to b
| (`enter`(a)) . add a to routing tables
| (`exit`(a)) . delete a from routing tables . *get*$[\langle$`leave`$(out\ self)\rangle$. in $a]$
| (`broadcast`(m, a)) . $\forall b : (b \in subambients)\ \&\ (a \neq b), \langle$`inform`$(m, a, b)\rangle$
`}`

$\textsc{Governor}_{\textsc{Amb}} \triangleq$
`repeat {`
open get
| (`inform`(m, a)) . process inform
| (`leave`$(move)$) . prepare to move . *move*
| other agent services
`}`

$msgs = inform + enter + exit + leave + broadcast$

$put/get : messages\curvearrowright\varnothing[\curvearrowright\{governors, staff\}, {}^{\circ}\varnothing, msgs]$

$\textsc{Governor}_{\textsc{Amb}} : governors\curvearrowright\varnothing[\curvearrowright\{scenes\}, {}^{\circ}messages, inform + leave + \text{others}]$

$\textsc{Admission}_{\textsc{Amb}} : \textsc{Scene}\curvearrowright\varnothing[\curvearrowright\varnothing, {}^{\circ}messages, msgs]$

Figure 12: Ambient scene manager and governor

- $\overline{inform}^{\uparrow}(m, a)$ sends a tuple containing a message m and an agent identifier a from the governor to the SM, which will then forward it to the agent a.
- $\overline{exit}^{\uparrow}(a)$ is used by the governor to signal to the SM that it (agent a) wants to leave the scene.
- $\overline{broadcast}^{\uparrow}(m)$ sends a message m from the governor to the SM for broadcasting to all the other agents in the scene.

Following the typing conventions presented in (Ghelli et al., 2001), we can assert types for the governor and for the admission scene (see Figure 11). The format of these types specifies whether the agent is mobile ($\curvearrowright$) or immobile ($\underline{\vee}$), followed by the interface, listing the names and types of the channels over which it may communicate. Thus, the governor type means that it is mobile and communicates on the three named channels, and the admission scene type means that it is immobile and admits the transmission of seals with governor interfaces over the channels *enter* and *exit*. As we shall see later, the ambient group typing of (Cardelli et al., 2000) provides information about which kinds of locations the governor may enter, while that information is implicit (but provided in the admission scene type) in this particular Seal type system.

4.3.2 An Ambient calculus approach

Communication in AC is somewhat different, which is underlined if one tries to replicate the nature of the Seal calculus solution: it rapidly becomes very awkward, involving much opening of ambients and their reconstitution with the message unleashed from the open-

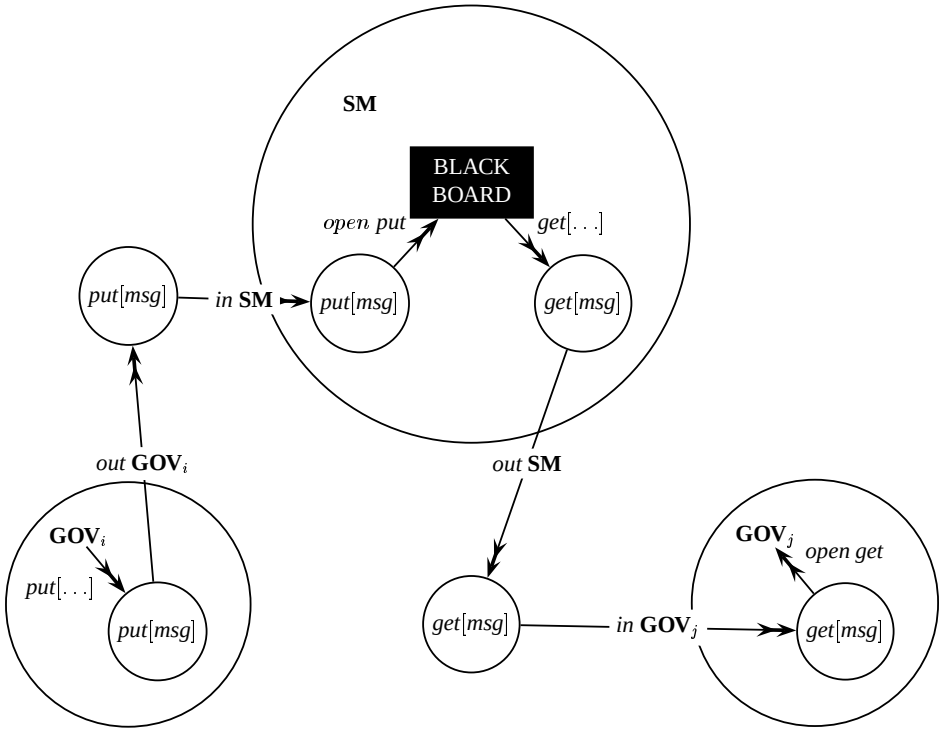


Figure 13: Communication between governors via the Scene Manager blackboard

ing. The real message is that trying to build a channel-like solution is the wrong metaphor in AC, because although each ambient does contain a single anonymous channel, communication is asynchronous and with the use of disjoint sum types, that single channel becomes, in effect, multiple named channels, or rather a pool (c.f. (Gelernter and Carriero, 1992)) which uses type information to match messages with read requests (a similar but more detailed observation on polymorphism and ambients appears in (Amtoft et al., 2001)) and starts to look rather similar to the ideas outlined in SecureSpaces (Bryce et al., 1999). In spirit however, it is much closer to a classical AI component: the blackboard.

Thus, our solution to communication in an AC specification takes that principle and implements a blackboard at scene ambient level by means of the single asynchronous channel in each ambient, and communication is achieved by creating ambients named *get* and *put* which move between the governor and SM ambients and are opened to release the message within and post it on the recipient's blackboard. In retrospect, it might have been convenient to use objective moves (the means in the ambient calculus for one ambient to move another), so that a governor might write $go\ out\ a.in\ sm.put[\langle m, a\rangle]$ where a is the governor ambient name and meaning "move out of a, in to SM and then become the ambient $put[\langle \texttt{inform}(m, a)\rangle]$, which when opened will release the message on to the SM blackboard". However, we decided to restrict ourselves for now to subjective operations, so it is a matter of the governor creating an ambient named *put* which will subsequently move out of the governor, and be dissolved by the SM, so unleashing the *inform* message for posting on the blackboard.

Meanwhile, the SM reads the messages posted on the blackboard. In the case of an *inform* message, it checks to see if the intended recipient is a sub-ambient (that is, another governor), and if so, constructs a *get* ambient containing the message, which

enters the target ambient. The target ambient dissolves the *get*, unleashing the *inform* message, which gets posted on the governor's internal blackboard and is subsequently processed. The passage of a message from one governor to another is shown in figure 13. If the target is not a sub-ambient, the message will somehow be routed to the recipient, assuming it is known within the institution. The *broadcast* message is simply broken into multiple *inform* messages and the *enter* and *exit* messages perform book-keeping for message routing.

Looking at this from the governor's point of view, it engages in the following activities:

- $put[\langle \texttt{inform}(m, \text{TO}, self)\rangle \ . \ out\ self]$ puts a message on to the blackboard for agent TO. The SM will forward the message by wrapping it in a *get* ambient.
- $open\ get$ is used by an agent to open ambients from the SM's blackboard in order subsequently to be able to receive `inform` and `exit` messages.
- $put[\langle \texttt{enter}(self)\rangle \ . \ out\ self]$ informs the SM of an agent's presence (since moves are subjective in AC, the SM must be informed explicitly of an agent's presence, otherwise it will not receive any messages, but see later in this section).
- $put[\langle \texttt{exit}(self)\rangle \ . \ out\ self]$ informs the SM that an agent would like to depart and the reply will include a capability to exit the scene.

Following the typing conventions presented in (Cardelli et al., 2000), where a type is a 4-tuple *proto-type* : $a\overset{\frown}{}b[\overset{\frown}{}c, {}^{\circ}d, e]$, where:

a	is this type
b	is the set of ambient types over which this may move (objective)
c	is the set of ambient types over which this may be moved (subjective)
d	is the set of ambient types this may open
e	is the set of message types exchangeable within this ambient

We can now assert types (see the bottom of figure 12) for (i) the blackboard messages (*put* and *get*), to say they do not undergo objective moves, but do cross governors and staff, open nothing and exchange messages of type *msgs*, where we have additionally defined *msgs* as the sum of *inform*, *enter*, *exit*, *leave* and *broadcast* (ii) for the governor to say they do not undergo objective moves, but do cross scenes, open messages and exchange messages of type *msgs* and other (unspecified) internal types and (iii) for the scenes to say they do not undergo objective moves, do not cross anything (that is, they are immobile), but do open messages and exchange messages of type *msgs*.

4.4 Agent mobility

The second aspect on which we focus, is the movement of governors between scenes. We put forward one view of our institution at the beginning of section 3, in which it was "just" the composition of conversation graphs. However, as we pointed out at the time, that ignores the semantics imposed by the synchronization nodes. In addition, it throws away any potential security measures that might be enabled through a strong separation of scenes based on mapping them to locations. Indeed, it is conceivable that the scenes within an institution could reside on distinct machines, using a location based model. Although it is tempting to see the merging of scenes as a simplification of the institutional structure and a means to remove an apparent need for mobile processes, we prefer to retain the distinction between the performative structure and conversations in the high-level institutional specification. Likewise, we consider it valuable to reflect that separation in the process algebraic modelling and only to consider its removal as a form of implemen-

$\textsc{Transport}_{\textsc{Amb}}(\mathit{route}\ \mathit{passenger}) \triangleq$
$\mathit{go}[\overline{\mathit{in}}\ \mathit{go}\ .\ \mathit{route}\ .\ \langle \texttt{enter}(\mathit{passenger})\rangle\ .\ \overline{\mathit{open}}\ \mathit{go}]$
$\textsc{Transport}_{\textsc{Amb}} : \mathit{transporters}^{\frown} \varnothing[^{\frown}\{\mathit{scenes}\}, {}^{\circ}\varnothing, \varnothing]$

$\textsc{Transport}_{\textsc{Seal}}(\mathit{route}) \triangleq$
$\mathit{load}^{\uparrow}\{\mathit{passenger}\}$
`.repeat {`
`  if` *nextdest* `then` $\overline{\mathit{go}}^{\uparrow}(\mathit{nextdest})$ `else break`
`  .` find next destination
`}`
$.\ \overline{\mathit{unload}}^{\uparrow}\{\mathit{passenger}\}$
$\textsc{Transport}_{\textsc{Seal}} = \texttt{Id}^{\frown}[\mathit{load} : \textsc{Governor}_{\textsc{Seal}}; \mathit{unload} : \textsc{Governor}_{\textsc{Seal}}; \mathit{go} : \mathit{name}]$

Figure 14: Transport agents

tation optimization, in the same way as modules do not have a physical representation in most compiled programs, and as in classical cases of strength reduction.

Having put forward a justification for our choice, we now proceed to look at the movement of agents between scenes as the movement of mobile locations between immobile locations.

Almost the only apparent concession to security in the preceding section on communication is that an agent is unable to leave without collaborating with the SM: in SC because a seal may only be moved by its parent and in AC, because that is the only way to obtain an exit capability. Of course, there is a qualitative difference here: all movement is mediated by the parent in SC, but it was a design decision to involve the SM in the AC solution. There are additional consequences too: once issued, that capability could be communicated and re-used. A development of the idea of affine types mentioned at the conclusion of (Cardelli and Gordon, 1999) might offer a solution here. Another solution might be to utilize the famous indirection principle of computing and handle all mobility via transport ambients, which are created on demand with fixed destinations, issue a capability for the agent that wants to move to enter it, and then dissolve the transport ambient upon arrival at its destination. This approach could be taken a step further and incorporate the registration on arrival action, rather than making it the responsibility of the governor. A similar approach could also be described in SC following that of the web-crawler specification that appears at the end of (Bugliesi and Castagna, 2001). Sketches of both appear in Figure 14.

The SC solution is fairly straightforward: the passenger is loaded into the transport seal via the parent and for each step of the path, the transport seal sends a message on the go channel to its parent until it has reached its destination; at this point the passenger is unloaded, again mediated by the parent of the transport seal. The *route* is a sequence of seal names over which the variable *nextdes* iterates. The type is read as comprising three channels: *load* and *unload* for the input and output of the seal to be transported, plus *go* which is the name of the channel to be used for general transport by all participating seals.

The AC solution requires a little more explanation (see figure 14), although its specification is briefer. The issue is that some local synchronization at each end of the journey is necessary to facilitate the ambient solution, in that the transporter should only start to move once the passenger ambient has entered it, furthermore, it should only be dissolved

once it has arrived at the destination. Thus it is that we can specify synchronization on the arrival of the passenger ($\overline{in}$ *go*) , then follow the path specified in the *route* capability, generate the message `enter`(*passenger*) to be received by the SM at the target scene and finally synchronize on the dissolving of the ambient on arrival at the destination ($\overline{open}$ *go*). It is important to note that we have used an ambient named *go* and this should not be confused with the objective move operator of the same name. As mentioned earlier, an objective move solution is also feasible.

The *open* operation of AC has been the target of some criticism lately (Castagna and Vitek, 1999; Ghelli et al., 2001), which is not unjustified in an insecure setting, but as can be seen from its restricted application here in the context of a fixed framework, it can, along with composite capabilities, lead to elegant and safe solutions.

4.5 Comparison

Most of the observations on the differences have been made in the preceding text, but to put forward a concrete conclusion to this evaluation process, we feel that despite the security objective behind the seal calculus, the ambient calculus, in the form extended by co-actions, offers a richer environment for the kind of modelling we wish to carry out, coupled with the increasing body of work on logics on top of ambient calculus which should enable us to express and verify a wide range of the norms that characterize institutions.

5 Related and future work

All the related work we are aware of is in the domain of type analysis and static analysis of various flavours of process algebra. The main body of work we have not referred to above is that of (Nielson and Nielson, 2000) on the use of Flow Logic to discover or establish properties of process algebraic specifications from static analysis. Our preliminary conclusion, however, is that the type and logic technologies appear more promising for the application we have described.

At the time of writing, we are still in the modelling stage, exploring appropriate metaphors for expressing our high-level structure in the entities provided by various process algebras. Although much of our current practical development is based on the Jade platform, it does not address the kinds of security issues that concern us, for which reason we are both interested, but cautious about, experimenting both with the implementation of Ambients on top of JoCaml (Fournet and Schmitt, 1999; Fournet et al., 2000) and the JavaSeal kernel (Bryce and Vitek, 1999).

We are also in the process of developing an institution specification language which has both a textual and graphical form – the latter corresponding to the designs in figures 3 and 4 – and tools for its analysis and transformation, including translating down to a stylized form of ambient calculus. However, what we really need are tools to analyze and eventually animate those ambient specifications, carry out type checking and type synthesis – prototype and evaluate the various process algebra type systems that are being proposed – and verify and analyze mobility patterns within the institution so that we can assert that the defining norms of the institution are upheld. Tools for type systems and logics for process algebra are few, largely due to the diversity of approaches currently under exploration, although the temporal logic system of (Ciancarini et al., 2000) seems promising.

Acknowledgements

Thanks to Marc Esteva and Carles Sierra for extended discussions on the FishMarket. Christian Queinnec's LiSP2TEX package was used to typeset the fragments of ambient and seal calculus. Further useful commentary on the conference version of this paper was supplied by Giuseppe Castagna, Silvano dal Zilio, Andrew Gordon and the AISB'01 referees. This work was partially supported by the Royal Society and the Consejo Superior de Investigaciones Cientificas.

References

Abadi, M. and Gordon, A. D. (1999). A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70.

Amtoft, T., Kfoury, A., and Pericas-Geertsen, S. (2001). What are polymorphically-typed ambients? In Sands, D., editor, *Programming Languages and Systems, 10th European Symposium on Programming, ESOP 2001*, volume 2028 of *Lecture Notes in Computer Science*, pages 206–220. Springer Verlag.

Bryce, C., Oriol, M., and Vitek, J. (1999). A Coordination Model for Agents Based on Secure Spaces. In Ciancarini, P. and Wolf, A., editors, *Proc. 3rd Int. Conf. on Coordination Models and Languages*, volume 1594 of *Lecture Notes in Computer Science*, pages 4–20, Amsterdam, Netherlands. Springer-Verlag, Berlin. revised into Coordinating Processes with Secure Spaces and to appear in Science of Computer Programming (Autumn 2001).

Bryce, C. and Vitek, J. (1999). The JavaSeal mobile agent kernel. In *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, Palm Springs, CA, USA.

Bugliesi, M. and Castagna, G. (2001). Secure safe ambients. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages*, London. ACM Press.

Cardelli, L. (1997). Mobile Ambient Synchronization. Technical Report SRC Tech Note 1997-013, Digital.

Cardelli, L., Ghelli, G., and Gordon, A. D. (2000). Ambient groups and mobility types. In van Leeuwen, J., Watanabe, O., Hagiya, M., Mosses, P. D., and Ito, T., editors, *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, volume 1872 of *LNCS*, pages 333–347. IFIP, Springer.

Cardelli, L. and Gordon, A. D. (1999). Types for mobile ambients. In ACM, editor, *POPL '99. Proceedings of the 26th ACM SIGPLAN-SIGACT on Principles of programming languages, January 20–22, 1999, San Antonio, TX*, ACM SIGPLAN Notices, pages 79–92, New York, NY, USA. ACM Press.

Cardelli, L. and Gordon, A. D. (2000). Anytime, anywhere: Modal logics for mobile ambients. In *Conference Record of POPL'00: The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 365–377, Boston, Massachusetts.

Castagna, G. and Vitek, J. (1999). Seal: A framework for secure mobile computations. In Bal, H., Belkhouche, B., and Cardelli, L., editors, *Internet Programming Languages*, number 1686 in LNCS, pages 47–77. Springer.

Ciancarini, P., Franzè, F., and Mascolo, C. (2000). Using a Coordination Language to Specify and Analyze Systems Containing Mobile Components. *ACM Transactions on Software Engineering and Methodology*, 9(2):167–198.

Esteva, M. and Padget, J. (2000). Auctions without auctioneers: distributed auction protocols. In Moukas, A., Sierra, C., and Ygge, F., editors, *Agent-mediated Electronic Commerce II*, volume 1788 of *Lecture Notes in Artificial Intelligence*, pages 20–38. Springer Verlag.

Esteva, M., Padget, J., and Sierra, C. (2001). Formalizing a language for institutions and norms. In Tambe, M. and Meyer, J.-J., editors, *Intelligent Agents VIII*, Lecture Notes in Artificial Intelligence. Springer Verlag. to appear.

Fournet, C., Lévy, J.-J., and Schmitt, A. (2000). An asynchronous distributed implementation fo mobile ambients. In van Leeuwen, J., Watanabe, O., Hagiya, M., Mosses, P. D., and Ito, T., editors, *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, volume 1872 of *LNCS*, pages 348–364. IFIP, Springer.

Fournet, C. and Schmitt, A. (1999). An implementation of ambients in JoCaml. In *Proceedings of the 5th ECOOP Workshop on Mobile Object Systems (MOS'99)*, Lisbon, Portugal.

Gelernter, D. and Carriero, N. (1992). Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107.

Genesereth, M. R. and Fikes, R. E. (1992). Knowldege interchange format version 3.0 reference manual. Technical Report Report Logic–92–1, Logic Group, Computer Science Department, Stanford University.

Ghelli, G., Castagna, G., and Zappa Nardelli, F. (2001). Typing mobility in the seal calculus. In *Proceedings of CONCUR 2001*, volume 2154 of *LNCS*, pages 82–101. Springer.

Levi, F. and Sangiorgi, D. (2000). Controlling interference in ambients. In *Conference Record of POPL'00: The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 352–364, Boston, Massachusetts.

Lomuscio, A. and Sergot, M. (2001). On multi-agent systems specification via deontic logic. In Tambe, M. and Meyer, J.-J., editors, *Intelligent Agents VIII*, Lecture Notes in Artificial Intelligence. Springer Verlag. to appear.

Merriam-Webster, Inc. (2001). *Merriam-Webster on-line Dictionary*. Merriam-Webster. http://www.m-w.com/.

Nielson, F. and Nielson, H. R. (2000). Shape analysis for mobile ambients. In *Conference Record of POPL'00: The 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Boston, Massachusetts. ACM, ACM Press.

Noriega, P. (1997). *Agent mediated auctions: The Fishmarket Metaphor*. PhD thesis, Universitat Autonoma de Barcelona.

North, D. C. (1991). *Institutions, Institutional Change and Economic Performance*. Cambridge University Press.

Padget, J. and Bradford, R. (1999). A π-calculus model of the spanish fishmarket. In *Proceedings of AMET'98*, volume 1571 of *Lecture Notes in Artificial Intelligence*, pages 166–188. Springer Verlag.

Rodríguez, J., Noriega, P., Sierra, C., and Padget, J. (1997). FM96.5 A Java-based Electronic Auction House. In *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, pages 207–224, London, UK.

Rodríguez, J.-A. (2001). *On the Design and Construction of Agent-mediated Institutions*. PhD thesis, Universitat Autonoma de Barcelona.

Sandhu, R., Ferraiolo, D., and Kuhn, R. (2000). The NIST model for role-based access control: Towards a unified standard. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC-00)*, pages 47–64, N.Y. ACM Press.

Searle, J. R. (1969). *Speech acts*. Cambridge University Press.

Tidswell, J. F. and Jaeger, T. (2000). Integrated constraints and inheritance in DTAC. In *Proceedings of the 5th ACM Workshop on Role-Based Access Control (RBAC-00)*, pages 93–102, N.Y. ACM Press.

Wojciechowski, P. T. and Sewell, P. (2000). Nomadic Pict: Language and infrastructure design for mobile agents. *IEEE Concurrency*, 8(2):42–52.

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Towards Agent-Based Service Composition Through Negotiation in Multiple Auctions

Christ Preist, Andrew Byde, Claudio Bartolini and Giacomo Piccinelli

Hewlett-Packard Laboratories
Filton Road, Stoke Gifford, Bristol, BS32 8QZ, United Kingdom
Chris_Preist@hp.com ; Andrew_Byde@hp.com ;
Claudio_Bartolini@hp.com ; Giacomo_Piccinelli@hp.com

Abstract

Service composition is the act of taking several component products or services, and bundling them together to meet the needs of a given customer. In the future, service composition will play an increasingly important role in e-commerce, and automation will be desirable to improve speed and efficiency of customer response. In this paper, we discuss the technical issues surrounding the automation of dynamic electronic service composition, using a fictitious company, FreightMixer, to demonstrate the process. We focus specifically on the issue of appropriate negotiation strategies for service composition, and present the specification of an algorithm to provide a robust solution to these problems in the context of multiple simultaneous auctions. We present a worked example to demonstrate the behaviour of the algorithm, and discuss related and future work.

1 Introduction

Over the past few years, Electronic Commerce has become an increasingly central part of the economy. An Internet presence is considered an essential part of doing business, rather than an exotic add-on to a company. More and more transactions, both from business to consumer and between businesses, are taking place online. Simple fixed cost business transactions are often automated at one or both ends, and auctions are overwhelmingly conducted by automated auctioneer software. Agent technology has been proposed as a means of automating some of the more sophisticated negotiations which businesses are involved in (e.g. (Jennings et al., 1996)). In this paper we look at a specific class of business process that will become increasingly important in the virtual economy: service composition. We consider the different technical issues that must be addressed if service composition is to be automated, and focus specifically on algorithms for the purchase of composite services from a group of auctions.

Over the last decade, companies have been encouraged by business consultants (Peters et al., 1984) to focus on their core competences. By trying to do everything – welding, graphic design, supply chain management, customer care, keeping the photocopiers running, producing good food in the office canteen – companies run the risk of being 'jack of all trades, but master of none'. As a result of this there is a danger that other smaller companies focused on the same core business will outperform them. To avoid that risk, and become more competitive, large companies are going through a process of 'disaggregation'. In some cases, this can mean splitting a large company into several parts, each

of which can focus on one core business (such as the recent move by Hewlett Packard to separate it's test and measurement business from its computing business, creating a new company, Agilent, from the former). In other cases, it can mean outsourcing more and more of a company's activities to other companies, maintaining only those activities that it truly excels in.

This trend is beginning to have an impact on many E-businesses, as well as traditional bricks-and-mortar companies. Companies would like to be able to outsource some of their activities over the Internet. Initially, this has focused on semi-permanent arrangements, with the web acting as an intermediary. (For example, career guidance information is provided to HP employees via a web-based third party). However, as this trend is becoming increasingly important, much research and development effort has been focusing on a new vision for the Internet – e-services. E-services are virtual entities that provide a service over the network through an open standard interface. The service may be information, such as the latest stock prices, or may be a virtual representation of some physical good or activity, such as a contract to transport a crate from one location to another. Because the service is offered through an open standard interface, any client familiar with this standard can use it. Furthermore, the output from one service can be fed directly into another service. This makes the creation of composite services and complex business processes which cross organizational boundaries possible. Potentially, this can be done automatically and dynamically, and agent technology will play a key role in this.

This leads to the emergence of an important role in the virtual economy – the service composer. As companies focus on their core competencies, other companies can focus on creating composite packages. This is not new – travel agents, among others, have done exactly that for years – but what is new is that it will be able to take place dynamically, automatically, over the Internet. In this paper, we discuss the technical issues that must be overcome if this is to come about, and focus specifically on negotiation algorithms. Firstly, in §2, we introduce the problem of service composition, and discuss which technical issues must be overcome if it is to be automated in e-commerce. In §3, we present an example service composition scenario involving a virtual company, "FreightMixer". In §4, we focus specifically on the problem of participating in multiple auctions to purchase service bundles. We present an algorithm specification, and give an example of the algorithm's behaviour. We then discuss related work, and finish by presenting conclusions and future work.

2 Issues in Service Composition

In an automated B2B transaction, the participants must go through three conceptually separate phases; matchmaking, negotiation and service delivery. (This lifecycle is an abstraction of that used in (Jennings et al., 1996)). We briefly describe these three phases, and then discuss how an enterprise involved in service composition participates in them. We conclude the section with an example scenario, taken from the freight services domain, that will illustrate the concepts discussed.

2.1 Matchmaking Phase

Matchmaking is the process of putting service providers and service consumers in contact with each other. For matchmaking to take place, services that wish to be dynamically located must publish details of themselves, and entities wishing to locate such services must search for these details. Some of the services advertising themselves for match-

making will be simple end-providers that can be negotiated with directly. Others may be brokers, auction houses and marketplaces that offer a locale for negotiating with and selecting among many potential providers offering similar services.

The services advertise a service description; a formal specification of the nature of the service they offer. Usually this information will be held in a central matchmaking directory. The facilitator agents of KQML (Finin and Fritzson, 1994) provide one approach to handling this.

When an entity wishes to locate a service of a certain type, it queries the matchmaker with a service request. This request takes a similar form to the service description, but may have certain fields unbound or constrained. The matchmaker returns a set of pointers to negotiations with appropriate service providers; in some cases, these negotiations are 1-1, while others may be auctions, exchanges, etc. Each pointer may also have a contract template associated with it, showing the associated terms and conditions of the negotiation. (Of course, some of these may be uninstantiated or semi-instantiated, and therefore open to negotiation). Standardization is essential to allow effective matchmaking. FIPA (Dale and Mamdani, 2001) is currently developing one approach. UDDI provides a less rich, but more widely supported, alternative.

2.2 Negotiation Phase

After matchmaking, a service consumer is faced with a variety of potential negotiations. Its aim is to procure the best service, taking into account factors such as price, speed of delivery, etc. To do this, it will participate in one or more of these negotiations. Different negotiations will have different market mechanisms – sets of rules determining how the negotiation should take place. The simplest such mechanism consists of a single service, offering itself at a fixed, non-negotiable, price. Other services may be willing to enter in to 1-1 bargaining with potential customers, or conduct auctions. Others may post their availability through exchanges, together with many other similar services. As a result of this negotiation, the different parties will agree a contract with terms and conditions that give each member certain rights (such as the right to use a certain service) and obligations (such as the obligation to pay a certain price), (Dignum and Weigand, 1995; Tan and Thoen, 1999; Norman and Reed, 2000).

2.3 Service Delivery Phase

Once the terms and conditions of service execution have been agreed by the participant to negotiation, service execution can start. That involves interaction between the service provider and the service consumer, to act according to the terms and conditions established during the negotiation process.

2.4 Service Composition

Service composition is the act of purchasing several *component services*, combining them, and selling them as a single composite service. The *service composer* responsible for the generation of the composite service must purchase the component services from a group of *suppliers* and will sell the composite service to one or more *customers*. In §3, we will give a detailed example of a company responsible for shipping freight. The company, FreightMixer, is the service composer. It is approached by a customer with a request to ship a crate from London to San Francisco. However, it does not own any cargo facilities of its own. Instead, it subcontracts, and arranges cargo space on a set of linked flights

from London to San Francisco. The airlines running these flights are the suppliers, and the individual flights are the component services.

Of course, the concepts of component and composite services are relative, based on the perspective of the service composer. A supplier may in turn be a service composer, and view the component service as a composite service from their perspective. Similarly, the buyer may be a service composer, using this service as a component in a larger composite service. For example, the shipping of the crate may be on behalf of a conference venue organiser, who is using the display materials in the crate to prepare a conference in San Francisco. The shipment is a component service, and the conference is the composite service.

The composite nature of the service affects the behaviour of the composer in all phases of the business transaction, and requires some modification to the standard, static view of these phases.

Matchmaking is traditionally viewed as a lookup process to find service providers able to meet a requester's needs, prior to the requester selecting and/or negotiating with them. Service providers simply advertise information about the service they offer in a database, and requesters use this database to make their selection. However, if the virtual economy is to encourage dynamic service composition, more flexibility will be necessary at this stage. A provider will have some idea of the general services it is interested in offering, but will not know the full details. At any given time, it can estimate these details based on the current state of markets. Hence, if it is to participate in matchmaking, the matchmaker must play a more active role. It must route potential service requests to service composers, which then respond with a dynamic service advertisement detailing the closest service to the request they can offer. This advertisement should not be treated as binding – it is simply an estimate based on the current market situation. Negotiation would be required to reach a binding contract.

In a context of dynamic service composition, additional requirements will be imposed on the service description to be advertised by the matchmaker and on the queries the system can deal with. The description should include abstract roles such as "insurance provider". The customer will know that these will be filled by subcontractors found by the service composer, but it will not know a priori who will take these roles. The service composer will dynamically negotiate with potential subcontractors to determine exactly who will perform these tasks. Service composers must take into account any restrictions that potential customers may wish to place over who plays the roles. For example, a customer may want to ensure that all subcontractors are members of appropriate trade bodies. For this reason, the service provider may need to give information to the customers during the matchmaking process about the names and/or details of potential subcontractors.

When the composer is either advertising a composite service or responding to a request through the matchmaker, then it needs models of how to decompose a service description into base service types that it can try to procure. Initially, this will be done at the service specification level. From a declarative description of the high level service, it will generate declarative descriptions of the sub-services which can be used during matchmaking to locate potential subcontractors.

The way in which a service request can be decomposed is not unique. The composer may generate many alternatives in advance, and then place requests for the base services. Alternatively, it can use the currently advertised services to inform the generation process. Which of these two strategies is appropriate will depend on whether the base services work in a "push" or "pull" advertising mode.

In the negotiation phase, the service composer will be involved in many interlinked negotiations. For any single bundle of base service types, the composer will be involved

in at least one, but more likely many negotiations to acquire instances of each service type. Furthermore, the service composer may simultaneously negotiate to purchase alternative bundles, in an effort to find which bundle is best. Ideally, it would like to do this in a non-committing way. However, some forms of negotiations (such as auctions) require participants to make a commitment when placing a bid, and provide no guarantees of success. When participating in negotiations of this kind, the provider of the composite service must take care to avoid buying incomplete or overly large bundles. Furthermore, the service composer may be simultaneously negotiating with potential clients. In this case, it must trade off its expectation of winning such negotiations against any commitments it makes in the negotiations to purchase base services.

As we have seen in the matchmaking phase, the composite service provider relies on the declarative description of the sub-services when generating potential bundles of base services. During negotiation, the service provider must ensure that the base services truly can inter-operate to provide the composite services. The declarative representation of the services can only guarantee this if the community as a whole defines standards of inter-operability. If this is not the case, the declarative description will have to be refined. In addition to the specification of the services, the parties will need to agree on the protocol that they will use to communicate during service delivery (e.g. what exchange of messages will take place to make a payment) and on the implementation of the interactions between the parties (e.g. what is the format of the messages that are exchanged). For more details on service composition, see (Piccinelli and Lynden, 2000; Piccinelli and Mokrushin, 2001)

At the end of the negotiation phase, the service agreement forms the basis of a contract. Usually contracts will be between two parties. There will be one contract between the requester and the service provider, and one contract between the service provider and each subcontractor. However, in some circumstances, multi-party contracts may be appropriate. They provide additional security to the service provider by offloading risk onto the subcontractors and onto the client.

After the contract has been formed, service delivery can commence. During service delivery, it is the responsibility of the service provider that the execution proceeds smoothly. Therefore the service provider will orchestrate the execution flow and ensure that each component service inter-operates appropriately. The orchestration relies upon a monitoring infrastructure that makes use of all the levels of service descriptions – declarative specification, procedural protocols and implementation of interactions. The subcontractors will play roles that appear in these descriptions. In order to fulfill these roles, each subcontractor will obtain an appropriate view over the original description which it will execute.

3 Example Scenario: FreightMixer

In order to illustrate these concepts, we now present a scenario to show the issues involved in service composition, and the impact of combined negotiation techniques. The scenario is taken from the freight domain. FreightMixer is an imaginary transport company that exploits cheap last-minute sales of excess hold space. While it may not be the quickest service, it aims to be the cheapest. Electronic marketplaces are both a source of resources (individual flight legs) and a channel for products (composite flights for a given customer).

FreightMixer does not own any transport infrastructure. Instead, it aims to dynamically design a cost-effective solution, using whatever third-party services are currently available cheaply. It composes these individual services together into a value-added solu-

tion which can be offered to customers at a premium. Its business model revolves around the dynamic acquisition of transport services at a competitive price, and the profitable sale of the composite service. Hence, effective negotiation techniques are crucial to the procurement as well as to the sales function of the company.

The knowledge that FreightMixer has of the freight market is the main asset of the company, and the very basis on which its competitive advantage is built. Crucial aspects of this knowledge are captured electronically, allowing algorithms to automatically design and implement end-to-end solutions. In particular, it must have domain knowledge about when two flight legs can be linked together, and how to do this. It must know how much time is required to get the crate from one plane to the other, how to contract with appropriate ground staff in different airports to arrange the hand-over, and what paperwork is necessary to enter different airports.

We now apply the three-stage model of business transactions to a typical deal generated by FreightMixer, and discuss what functionality a service composition company requires.

3.1 Matchmaking Phase

During the matchmaking phase, FreightMixer acts in two distinct sets of markets:
(i)In the markets for end-to-end cargo services, it acts as a potential seller. It observes the advertised requirements of potential customers in this market.
In the markets for hold space on flights (and possibly ships), it acts as a potential buyer. It observes the availability and cost of different options in these markets.

In its role as service composer, it must (a) understand requirements of the potential customers which are currently requesting services in the end-to-end cargo markets, and identify a service which could meet their needs (b) identify the alternative ways this service can be created from component services (i.e., hold space on specific flights) (c) identify potential sellers of these component services in the markets for hold space on flights.

As an outcome of the matchmaking phase, FreightMixer will have a list of negotiation options. Each option will consist of the following:

- A potential buyer, or set of buyers, who are currently requesting a service in the end-to-end cargo marketplaces.
- A service specification which meets the needs of these buyers.
- One or more alternative decompositions of this service into component services.
- A list of sellers in the markets for hold space who are offering to sell individual component services appearing in these decompositions.

For example, assume FreightMixer observes a Request For Quotes reverse auction for sending a 1 tonne crate from London to San Francisco, with the best offer currently at 210. Using its database of service models, FreightMixer identifies alternative combinations of flights which might potentially meet these needs. It identifies a direct route from London(LHR) to San Francisco(SFO), and also identifies alternative routes via Chicago(ORD), New York(JFK) and Boston(BOS). It then checks the auctions for excess hold space and finds that appropriate auctions exist for all legs except LHR to JFK. The remaining alternatives it has are shown in Figure 1.

Hence, FreightMixer has an option consisting of the buyer conducting the reverse auction, three alternative ways of generating the required service from component services

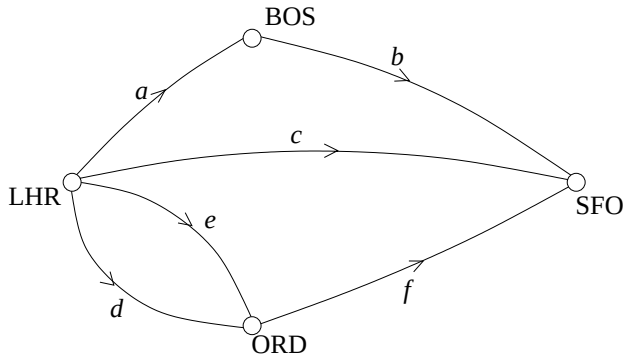


Figure 1: Graph of services

({LHR-SFO}, {LHR-ORD & ORD-SFO} or {LHR-BOS & BOS-SFO}) and potential sellers for each of the component services. The options would also include subsidiary services (e.g. insurance, re-packaging, temporary storage) which we shall not discuss.

FreightMixer may also choose to pro-actively advertise certain composite services during the matchmaking phase. If it expects demand for certain services, it can go out and provisionally negotiate for the individual components while waiting for clients to respond to the advertisement.

3.2 Negotiation Phase

During the negotiation phase, FreightMixer must again participate in two sets of markets. It must participate as a seller in the markets for end-to-end cargo services, negotiating over the terms and conditions of sale with the various buyers identified in its set of options. It must participate as a buyer in the markets for hold space on flights, negotiating with potential sellers of component services identified in the option set. Often, this will involve parallel negotiation in multiple marketplaces, and the use of different trading mechanisms (e.g. exchanges, auctions, RFQs). Furthermore, it may involve the negotiation of multiple complex parameters, for example: pricing policy, interaction processes, time constraints, and payment procedure.

One of the key problems FreightMixer faces is that of making commitments when negotiating simultaneously with customers and suppliers. It would like a scenario which avoids it making commitments to sell a service which it may not be able to deliver, or to buy a service it may not need. Hence it favours a scenario such as:

1. FreightMixer negotiates a price with a customer. The customer agrees to definitely buy the service, but FreightMixer doesn't commit to providing it.

2. Based on this known sale price, FreightMixer negotiates with several potential providers of component services. It agrees deals to maximise its profit, and commits to those deals.

3. FreightMixer returns to the customer, and commits to the original deal.

Notice that this scenario requires the customer to commit to an uncertain deal, to allow FreightMixer to avoid risk. A similar, dual, scenario where the component service providers commit to a deal without FreightMixer also committing will also give this.

However, neither of these scenarios can be relied upon. Firstly, buyers (resp. suppliers) may not want to use such a scenario as it places risk on them. Secondly, many market mechanisms (such as auctions) require commitment if FreightMixer is to negotiate in them.

Hence a more likely scenario, based on the example above, is:

1. FreightMixer observes the RFQ auction to ship a crate to SFO, and estimates (based on prior experience) what offer it would need to place to be likely to win, and the risk of losing associated with such an offer.

2. Based on the income it would receive, and taking into account the risk associated with the chance of losing, FreightMixer determines the maximum it is prepared to spend on procuring the composite service.

3. Using this as an upper bound, it places bids in the auctions for some of the component services. (e.g., space on the flights from LHR to BOS and BOS to SFO). It must do this in such a way to procure an appropriate set of components for the cheapest price. In §4, we discuss this problem in more detail, and present a solution.

4. If it wins these auctions, it returns to the RFQ auction and places an offer there.

The outcome of this second phase is a series of contracts with suppliers that make sure that FreightMixer can deliver a composite service to the customer, and a contract with the end customer.

3.3 Service Execution

When FreightMixer has bought appropriate components to meet a customers need, and successfully negotiated with the customer to agree a contract, service can be delivered. During the service delivery phase, FreightMixer must ensure that the hand-over of the good at each stage of the journey takes place smoothly and appropriate paperwork is carried out.

A detailed discussion of the problems deriving from this phase is beyond the scope of this paper. For more information see (Salle et al., 2001; Morciniec et al., 2001).

In this way, FreightMixer is able to provide the same functionality as a large company despite the fact that its only assets are its market knowledge and organization ability. Using this expertise, it can compete with established transport companies with large infrastructures. We believe that service composition will play an important role in the future of e-commerce.

4 A Negotiation Algorithm Specification to Purchase Service Bundles

We now turn our attention to one specific aspect of the service composition problem – that of negotiating to purchase composable services. For the purposes of this initial work, we make certain restrictions on the scenario discussed above. Firstly, we assume that the service composer is buying from a set of auctions only. Secondly, we assume the customer of the service composer is offering to pay a certain fixed price. Hence, initially, we ignore the issue of simultaneously negotiating with the customer. We hope to address this in future work.

We consider how an agent involved in service composition should behave when participating in a set of auctions. Its aim is to buy a set of services which can be composed to sell on as a bespoke composite service, possibly to a specific customer with special requirements. There may be several ways of creating this composite service out of individual services for sale in the auctions. The agent's task is to purchase one such set of services which can be composed, without accidentally purchasing additional, unnecessary services.

In this section, we propose a possible approach for doing this. We first present the decision problem the agent is faced with, and then present the specification of an algorithm to perform service composition in this environment.

4.1 Specification of the Decision Problem

We assume our agent is participating in a set of auctions, $\mathcal{A}$. These auctions all start at roughly the same time, but may finish at different times. Each auction is selling one good or service. The auctions are English auctions with a fixed closing time. Participants can place bids at any time, provided the new bid is a minimum increment, ϵ, above the last bid. We choose units in which this minimum increment is 1. At the closing time, the good or service for sale is sold to the highest bidder at the price they bid. We also assume that the bid increment of each auction is very small with respect to the value of the good or service for sale.

To each subset $A \subset \mathcal{A}$ we associate a number $v(A)$, the "value" to the agent of winning the auctions A. By structuring the valuation of the agent as a function $v : 2^{\mathcal{A}} \rightarrow \mathbb{R}$, we allow for complements and substitutes in the normal fashion. We define a *bid set* to be a pair $(A, \mathbf{p})$, where $A \subset \mathcal{A}$ and $\mathbf{p} : A \rightarrow \mathbb{Z}$ is a price function. The "utility" to the agent of winning a bid set $(A, \mathbf{p})$ is:

$$u(A, \mathbf{p}) = v(A) - \sum_{a \in A} \mathbf{p}(a).$$

Our agent maintains a probabilistic model of the expected outcomes of each auction, based on past performance of similar auctions. (Discussion of some possible ways of generating this model is provided in (Preist et al., 2001a)).

To each auction $a \in \mathcal{A}$ is associated a price distribution $P_a : \mathbb{Z} \rightarrow [0, 1]$ representing the belief that, with probability $P_a(p)$, auction a will close at price p. We set

$$F_a(p) = \sum_{p' \geq p} P_a(p')$$

as the agent's believed probability that auction a will close at or above price p. For subsets $A \subset \mathcal{A}$ we define $P_A(\mathbf{p})$ to be the believed probability that the auctions in A will close at the prices specified by a *price function* $\mathbf{p} : \mathcal{A} \rightarrow \mathbb{Z}$:

$$P_A(\mathbf{p}) = \prod_{a \in A} P_a(\mathbf{p}(a)), \tag{1}$$

and likewise F_A, the probability that that the auctions in A will close at or above the prices specified by $\mathbf{p}$:

$$F_A(\mathbf{p}) = \prod_{a \in A} F_a(\mathbf{p}(a)). \tag{2}$$

If the price in auction a is q, then the agent believes that the probability of a bid at price $p \geq q$ winning is:

$$P_{win}(a, p, q) := \frac{P_a(p)}{F_a(q)}. \tag{3}$$

Similarly, for a collection of auctions A with current prices $\mathbf{q} : A \to \mathbb{R}$, the probability of the auctions closing at prices $\mathbf{p}$ is:

$$P_{win}(A, \mathbf{p}, \mathbf{q}) = \frac{P_A(\mathbf{p})}{F_A(\mathbf{q})}. \tag{4}$$

4.2 Specification of the algorithm

We now consider how the agent can use these beliefs to calculate information about expected future utility of deals it may win. Firstly, we define the notion of the expected utility $E(B, A, \mathbf{q})$ of a set of auctions B, given a set of observed prices $\mathbf{q}$, and given that the agent holds active bids in auctions A:

$$E(B, A, \mathbf{q}) = v(B) - C(B \cap A, \mathbf{q}) - C(B \setminus A, \mathbf{q} + 1) \tag{5}$$

where the function $C(S, \mathbf{q}')$ is the expected cost of winning the auctions S at prices greater than or equal to $\mathbf{q}'$:

$$C(S, \mathbf{q}') = \sum_{\mathbf{p}' \geq \mathbf{q}'} \sum_{a \in S} P_{win}(a, \mathbf{p}'(a), \mathbf{q}'(a))\mathbf{p}'(a). \tag{6}$$

The expected utility of a set of auctions is thus the value of the bundle, minus the expected cost of winning each of the auctions. The latter is calculated by using the believed probability that the auction will finish at each given price, if our agent places a bid at that price. We restrict $p' > q$ for auctions $B \setminus A$, in (5) because we know that the agent does not hold bids in these auctions at prices $\mathbf{q}$, and so has no probability of winning at these prices.

The expression (5) gives us some idea of the intrinsic value of a bundle of goods B, but is not the expected return for placing a single bid in the auctions in B. In general, such a bid does not *have* an expected return: we must reason over complete strategies.

Consider the expected value (given that prices are currently $\mathbf{q}$, and the agent holds the active bids A) of the following strategy, which we call "commitment to B": The agent chooses a set of auctions B, and for all future time steps, will always bid on any elements of B in which it does not hold active bids. If the agent sticks to this commitment, then we know its future choices, and so precise formulae for expected return can be calculated.

Let S be a possible set of auctions that the agent may win using this strategy: $B \subset S \subset A \cup B$. The probability that the auctions $S \setminus B$ will not be outbid, while the auctions $A \setminus S$ are, is:

$$P_{ret}(S, A, \mathbf{q}) = \frac{F_{A \setminus S}(\mathbf{q} + 1) P_{S \setminus B}(\mathbf{q})}{F_{A \setminus B}(\mathbf{q})}$$

Given this eventuality, the expected utility is evaluated in the same way as (5), except that instead of $v(B)$, the value we obtain is $v(S)$, and we occur additional costs for each auction in $S \setminus B$ that we win.

It follows that the expected value for following the commitment to B is:

$$E_c(B, A, \mathbf{q}) = E(B, A, \mathbf{q}) + \sum_{B \subset S \subset A \cup B} P_{ret}(S, A, \mathbf{q}) \Big((v(S) - v(B)) - \sum_{a \in S \setminus B} \mathbf{q}(a) \Big) \tag{7}$$

The terms in this expression for which $S = B$ are the desired outcomes. The other terms correspond to obtaining some non-empty collection $S \setminus B$ of goods that do not contribute to our desired bundle B. Although they could still provide positive value, it is anticipated that in the service composition arena, where goods tend to complement one another, the slight increase of $v(S)$ with respect to $v(B)$ will not be large enough to compensate for the increase in costs $\sum_{a \in S \setminus B} \mathbf{q}(a)$, and each of these terms would have a negative impact on the expected value of the commitment.

The algorithm (`COMPOSER`) we propose is that at each time step the agent calculates the commitment B which has largest expected utility $E_c(B, A, \mathbf{q})$ given the currently held bids A and prices $\mathbf{q}$, and places the minimal bids required to take the lead in $B \setminus A$. In practice, this means it will bid initially in the auctions which have highest a-priori expected utility. It will continue to compete in these auctions, placing more bids when outbid. However, if sufficient competing bids are placed to reduce the expected utility of this set of auctions, then it may change to another set of auctions, for another bundle. It will do this if the expected gain from changing to this new bundle outweighs the expected cost of currently held bids which appear in the old bundle but not in the new bundle.

There are two obvious problems with this algorithm:

- By its very nature, our algorithm does not in fact commit, since it re-evaluates its options at each opportunity. However, the value $E_c(B, A, \mathbf{q})$, which is truly the expected value of committing to bid on B, and hence is *not* the expected value according to the specified algorithm, is none-the-less (we claim) a good indication of the optimal choice to make. The estimate we use is conservative, in that the agent chooses a single bundle that will give the best overall expected utility. Choosing a different bundle for each possible outcome can only improve on this. We have adopted this approach initially, as we believe that it will provide good performance in the majority of situations. Experimentation and further analysis will be necessary to test this hypothesis.

- In practice, if the number of auctions is large, it will be difficult to evaluate (7) given realistic computational resource bounds. Ideally, if we had perfect information and unlimited computation time, we would calculate this accurately. Finding appropriate simplifications which still give good results is a topic to which we will return in the future work section.

4.3 Worked Example

To illustrate how this analysis operates, we return to the FreightMixer scenario described in §3. Based on past histories of similar auctions to the ones that were selected during the matchmaking phase, FreightMixer creates beliefs about the expected distribution of closing prices of these auctions. We assume that the closing prices are uniformly distributed over the following sets:

$$\begin{array}{lcl} a & : & \{40, 45, \ldots, 135, 140\} \\ b & : & \{20, 25, \ldots, 95, 100\} \\ c & : & \{130, 135, 140, 145, 150\} \\ d & : & \{50, 55, \ldots, 105, 110\} \\ e & : & \{80, 85, \ldots, 115, 120\} \\ f & : & \{30, 35, \ldots, 65, 70\} \end{array} \tag{8}$$

Before bidding begins, the agent holds no bids. We assume that the current price function $\mathbf{q}_0$ lies just below all of the above prices. The expected utilities of committing to each of the bundles which we seriously consider are therefore the same as the expected values of the bundles:

$$\begin{array}{rcl} E(\{a, b\}, \emptyset, \mathbf{q}_0) & = & 50 \\ E(c, \emptyset, \mathbf{q}_0) & = & 60 \\ E(\{d, f\}, \emptyset, \mathbf{q}_0) & = & 70 \\ E(\{e, f\}, \emptyset, \mathbf{q}_0) & = & 50 \end{array} \tag{9}$$

The agent therefore chooses to bid in $\{d, f\}$, even though the bundle $\{a, b\}$ has greater initial value[1]. This can be seen as sensible, given that by bidding for $\{a, b\}$ the agent runs the risk of ending up committed to this bundle, even though it is non-optimal in expectation. It can be argued that the risk of such a commitment is low, since if the prices in a or b become prohibitively high, then the agent can simply wait, and with high probability will be outbid in these auctions, and so de-committed from them. It follows, however, that the agent also has a correspondingly low probability of winning $\{a, b\}$ at these low prices: it is precisely the payoff between the chance of a good deal and the chance of being committed to a bundle which our algorithm seeks to address.

This payoff between expected return and commitment explicitly comes into effect in this example if prices in d or f rise without the auctions closing.

Suppose that prices in c, d and f have risen to 140, 75, 40. If the agent holds no bids, it should bid in c, since the expected cost of c given these prices is 147.5, whereas the expected cost of $\{d, f\}$ is $97.5 + 57.5 = 155$. Even if the agent holds the active bid in auction f, then the expected loss from accidentally winning f, which is the cost of f, 40, multiplied by the probability that no other agent will bid in f, 0.16666, is less than the difference between the expected costs of c and $\{d, f\}$, and so it is still preferable to bid for c despite the risk of winning f.

If, on the other hand, the agent holds the leading bid in d, then the potential cost of winning d is too high (expected loss 13.333) to risk bidding for c, despite the fact that c is, by now, expected to do better than $\{d, f\}$.

The benefit of this algorithm over a greedy one is clear in the same situation: a greedy algorithm would continue to pursue $\{d, f\}$ in preference to c until its current aggregate price was as large as that in c. For example, if prices in c, d and f were 145, 95 and 45 (and the agent held no bids) then a greedy algorithm would bid in $\{d, f\}$ in preference to c. The reason why this is foolish is that the probability of winning one of these auctions but not the other (at these prices), is large: 40%. If the agent wins one, then it is committed to bidding for the other, despite its large expected cost.

[1] The bid-price for the bundle $\{a, b\}$ is 60, much lower than that of $\{d, f\}$, at 80.

5 Related Work

Research into automated negotiation has long been an important part of distributed AI and multi-agent systems. Initially it focused primarily on negotiation in collaborative problem solving, as a means towards improving coordination of multiple agents working together on a common task. (Laasri et al., 1992) provide an overview of the pioneering work in this area. As electronic commerce became increasingly important, the work expanded to encompass situations with agents representing individuals or businesses with potentially conflicting interests. The Contract Net (Smith, 1980) provides an early architecture for the distribution of contracts and subcontracts to suppliers. It uses a form of distributed request-for-proposals. However, it does not discuss algorithms for determining what price to ask in a proposal. (Jennings et al., 1996) use a more sophisticated negotiation protocol to allow the subcontracting of aspects of a business process to third parties. This is primarily treated as a one-to-one negotiation problem, and various heuristic algorithms for negotiation in this context are discussed in (Faratin et al., 1998). (Vulkan and Jennings, 1998) recast the problem as a one-to-many negotiation, and provide an appropriate negotiation protocol to handle this.

Other relevant work in one-to-one negotiation includes the game-theoretic approach of (Rosenschein and Zlotkin, 1994) and the logic-based argumentation approach of (Parsons et al., 1998). As much electronic commerce involves one-to-many or many-to-many negotiation, the work in the agent community has broadened to explore these cases too. The Michigan AuctionBot (Wurman et al., 1998) provides an automated auction house for experimentation with bidding algorithms. The Spanish Fishmarket developed by (Rodriquez-Aguilar et al., 1997) provides a sophisticated platform and problem specifications for comparison of different bidding strategies in a Dutch auction, where a variety of lots are offered sequentially. The Kasbah system (Chavez et al., 1997) featured agents involved in many-to-many negotiations to make purchases on behalf of their users. However, the algorithm used by the agents (a simple version of those in (Faratin et al., 1998)) was more appropriate in one-to-one negotiation, and so gave rise to some counter-intuitive behaviours by the agents. (Preist and van Tol, 1998) and (Cliff and Bruten, 1998) present adaptive agents able to effectively bid in many-to-many marketplaces, and are the first examples of work which borrow techniques from experimental economics to analyze the dynamics of agent-based systems. (Preist, 1999) demonstrates how these can be used to produce a market mechanism with desirable properties. (Park et al., 1998) and (Park et al., 1999) present a stochastic-based algorithm for use in the University of Michigan Digital Library, another many-to-many market.

(Gjerstad and Dickhaut, 1998) use a belief-based modeling approach to generating appropriate bids in a double auction. Their work is close in spirit to ours, in that it combines belief-based learning of individual agents bidding strategies with utility analysis. However, it is applied to a single double auction marketplace, and does not allow agents to bid in a variety of auctions. (Vulkan and Preist, 1999) use a more sophisticated learning mechanism that combines belief-based learning with reinforcement learning. Again, the context for this is a single double auction marketplace. Unlike Gjerstad's approach, this focuses on learning the distribution of the equilibrium price. The work of (Garcia et al., 1998) is clearly relevant. They consider the development of bidding strategies in the context of the Spanish Fishmarket tournament. Agents compete in a sequence of Dutch auctions, and use a combination of utility modeling and fuzzy heuristics to generate their bidding strategy. Their work focuses on Dutch rather than English auctions, and on a sequence of auctions run by a single auction house rather than parallel auctions run by multiple auction houses.

In our previous work, (Preist et al., 2001a), we have presented algorithms which allow agents to participate simultaneously in multiple auctions for the purchase of a number of similar goods. In (Preist et al., 2001b), we show how agents using these algorithms in multiple auctions can create a more efficient and stable market. It is interesting to contrast our analysis with that of (Greenwald and Kephart, 1999). They demonstrate that the use of dynamic price-setting agents by sellers, to adjust their price in response to other sellers, can lead to an unstable market with cyclical price wars occurring. We, however, show that (in a very different context) the use of agents improves the dynamics and stability of the market. From this, we can conclude that agent technology is not a-priori 'good' or 'bad' for market dynamics, but that each potential role must be studied to determine its appropriateness.

In this paper, we have extended our earlier work to develop algorithms to purchase heterogeneous bundles of goods from multiple auctions. An alternative approach is to attempt to provide the right market mechanism in the first place, providing a centralized point of contact for all buyers and sellers to trade. (Sandholm, 2000) proposes a sophisticated marketplace able to handle combinatorial bidding, and able to provide guidance to buyers and sellers as to which market mechanism to adopt for a particular negotiation. In the long term, as the different auction houses merge or fold and only a few remain, this approach will be ideal. In the short term, we expect improved market dynamics will occur through autonomous agents in multiple auctions.

6 Conclusions and Future Work

In the future, service composition will play an essential role in e-commerce. Composite service products will be created on the fly in response to customer requests. However, if this is to happen, several technical problems must be overcome.

We have focused on the key problem of effective negotiation for service composition, and presented the specification of an algorithm to perform this task. The algorithm competes in multiple simultaneous auctions, placing appropriate bids to create service bundles. In future work, we will focus on the two areas mentioned as problematic at the end of §4.2.

Firstly, we will pursue formal methods to calculate the expected value of a specified bidding algorithm. We already have a closed form (7) for the value of the scale of strategies described as "committed", but not (for example) for `COMPOSER`. By finding expressions which bound, or better still equal (in expectation) the value that can be extracted from a given algorithm, we can be sure that a given algorithm has theoretical performance properties which otherwise we would have to guess at, or simulate.

On the other hand, theoretical models for algorithm effectiveness are useless if those algorithms are impractical, and so another main focus of our future work in this area will be finding algorithms which are of low complexity. There is likely to be a payoff between complexity and optimality; it is our goal to develop practical algorithms which have high, well understood value.

Throughout the paper, we have made assumptions regarding risk which are often not appropriate in practice. In particular we have assumed risk-neutrality, whereas in reality many potential users of service aggregation algorithms are risk-averse. In future work we will extend the algorithmic analysis we have just described to cases involving alternative risk profiles. We also plan to address the problem of simultaneous negotiation not only with the suppliers, but also with the purchasers. We would also like to generalize the work beyond the English auction, and to allow auctions to have staggered opening times.

References

Chavez, A., Dreilinger, D., Guttman, R., and Maes, P. (1997). A real-life experiment in creating an agent marketplace. In *Proc. 2^{nd} Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Systems*.

Cliff, D. and Bruten, J. (1998). Less than human: Simple adaptive trading agents for CDA markets. In *Proc. of the 1998 Symposium on Computation in Economics, Finance, and Engineering: Economic Systems*.

Dale, J. and Mamdani, E. (2001). Open standards for interoperating agent based systems. forthcoming.

Dignum, F. and Weigand, H. (1995). Modelling communication between cooperative systems. In *Proc. 7^{th} Conf. on Advanced Information Systems Engineering (CAiSE'95)*, pages 140–153.

Faratin, P., Sierra, C., and Jennings, N. (1998). Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 3-4(24):159–182.

Finin, T. and Fritzson, R. (1994). KQML as an agent communication language. In *Proc. Third International Conference on Information and Knowledge Management (CIKM'94)*. ACM Press.

Garcia, P., Giminez, E., Godo, L., and Rodriguez-Aguilar, J. (1998). Possibilistic-based design of bidding strategies in electronic auctions. In *Proc. 13th Biennial European Conference on Artificial Intelligence*.

Gjerstad, S. and Dickhaut, J. (1998). Price formation in double auctions. *Games and Economic Behaviour*, 22(1):1–29.

Greenwald, A. R. and Kephart, J. O. (1999). Shopbots and pricebots. In *Proc. 16^{th} Int. Joint Conf. on Artificial Intelligence*.

Jennings, N. R., Faratin, P., Johnson, M. J., O'Brien, P. O., and Wiegand, M. E. (1996). Using intelligent agents to manage business processes. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, pages 345–360.

Laasri, B., Laasri, H., Lander, S., and Lesser, V. (1992). Generic model for intelligent negotiating agents. *International Journal of Intelligent and Cooperative Information Systems*, 2(1):291–317.

Morciniec, M., Salle, M., and Moynihan, B. (2001). Towards regulating electronic communities with contracts. In *Proc. 2^{nd} Workshop on Norms and Institutions in MAS, at Autonomous Agents 2001*.

Norman, T. J. and Reed, C. (2000). Delegation and responsibility. In *Proc. 3^{rd} UK Workshop on Multi-Agent Systems*.

Park, S., Durfee, E., and Birmingham, W. (1998). Emergent properties of a market-based digital library with strategic agents. In *Proc. 2^{nd} Int. Conf. on Multi-Agent Systems*.

Park, S., Durfee, E., and Birmingham, W. (1999). An adaptive agent bidding strategy based on stochastic modelling. In *Proc. 3^{rd} Int. Conf. on Autonomous Agents*.

Parsons, S., Sierra, C., and Jennings, N. R. (1998). Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292.

Peters, T., Austin, K., and Peters, T. J. (1984). *A Passion for Excellence : The Leadership Difference*. Warner Books.

Piccinelli, G. and Lynden, S. (2000). Concepts and tools for e-service development. In *Proc.* 7^{th} *Int. Workshop HP OVUA*.

Piccinelli, G. and Mokrushin, L. (2001). Dynamic e-service composition in DySCo. part of the 21^{st} Int. Conf. on Distributed Computing Systems (ICDCS-21).

Preist, C. (1999). Commodity trading using an agent-based iterated double auction. In *Proc.* 3^{rd} *Int. Conf. on Autonomous Agents*.

Preist, C., Bartolini, C., and Philips, I. (2001a). Algorithm design for agents which participate in multiple simultaneous auctions. In Dignum, F. and Cortes, U., editors, *Agent Mediated Electronic Commerce III*, Lecture Notes in AI. Springer Verlag.

Preist, C., Byde, A., and Bartolini, C. (2001b). Economic dynamics of agents in multiple auctions. In *Proc.* 5^{th} *Int. Conf. on Autonomous Agents*.

Preist, C. and van Tol, M. (1998). Adaptive agents in a persistent shout double auction. In *Proc.* 1^{st} *Int. Conf. on the Internet, Computing and Economics*. ACM Press.

Rodriquez-Aguilar, J., Noriega, P., Sierra, C., and Padget, J. (1997). A Java-based electronic auction house. In *Proc. Second International Conference on the Practical Application if Intelligent Agents and Multi-Agent Systems*, pages 207–224.

Rosenschein, J. and Zlotkin, G. (1994). *Rules of Encounter*. MIT Press.

Salle, M., Morciniec, M., and Moynihan, B. (2001). A conceptual architecture for market governance in trusted electronic marketplaces. In *Proc.* 2^{nd} *Workshop on Norms and Institutions in MAS, at Autonomous Agents 2001*.

Sandholm, T. (2000). eMediator: A next generation electronic commerce server. In *Proc.* 4^{th} *Int. Conf. on Autonomous Agents*.

Smith, R. G. (1980). The Contract Net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computing*, 12(29):1104–1113.

Tan, Y. and Thoen, W. (1999). A logical model of directed obligations and permissions to support electronic contracting. *International Journal of Electronic Commerce*, 3(2):87–104.

Vulkan, N. and Jennings, N. (1998). Efficient mechanisms for the supply of services in multi-agent environments. In *Proc.* 1^{st} *Int. Conf. on the Internet, Computing and Economics*. ACM Press.

Vulkan, N. and Preist, C. (1999). Automated trading in agents-based markets for communication bandwidth. In *Proc.* 2^{nd} *UK Workshop on Multi-Agent Systems*.

Wurman, P., Wellman, M., and Walsh, W. (1998). The Michigan internet AuctionBot: A configurable auction server for human and software agents. In *Proc.* 2^{nd} *Int. Conf. on Autonomous Agents*.

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Autonomous Reflectors over Active Networks: Towards Seamless Group Communication

Lidia Yamamoto and Guy Leduc

Research Unit in Networking, University of Liège
Institut Montefiore, B28, B-4000 Liège, Belgium
yamamoto@run.montefiore.ulg.ac.be ; leduc@run.montefiore.ulg.ac.be

Abstract

We present a reflector service that seeks to maintain application-level connectivity in the presence of network-level multicast failures. The service is based on the dynamic deployment of autonomous reflectors modelled as mobile agents on top of an active network infrastructure. It is able to repair multicast tree failures by building a self-organising tree of reflectors, which will be connected to each other via unicast. The scheme is decentralised and takes into account node and link resources to find agent locations that lead to low cost tree configurations. We focus on the basic decision mechanisms related to code mobility during the tree construction and destruction phases, namely: cloning, migration, merging and termination. We show some preliminary simulation results that confirm the viability of the approach and discuss directions for further research.

1 Introduction

The demand for multimedia group communication is growing, and multicast is widely recognised as an important service to enable efficient group communication. However, multicast protocols still face deployment obstacles, and the quality experienced by many users is still unsatisfactory.

One of the fallback solutions used is to establish multicast reflectors to serve users that have no multicast access. A reflector is a user-level gateway application that acts as a proxy between a multicast-enabled network and a set of unicast users. It forwards packets from the multicast group to all unicast clients, and from every unicast client to the multicast group and to all other unicast clients. This guarantees that connectivity is maintained within the session, in spite of the fact that some participants have no access to IP multicast, or in the presence of failures in the multicast tree.

Existing reflector software must typically be installed manually, which is an extra burden for the session organisers and users. Besides that, even the session organisers seldom have enough knowledge about the current network conditions in order to be able to choose an optimum location for a reflector. What happens then is that reflectors are typically placed close to the main session source, and all multicast disabled participants must connect to it as clients. This generates an amount of redundant traffic which is proportional to the number of reflector clients, and therefore obviously does not scale to large sessions where potentially large portions of the network might need the reflector service.

It would be interesting to be able to dynamically install reflectors when there are connectivity failures or administrative restrictions to multicast traffic. The location of reflectors should be automatically determined according to the network conditions observed during the session.

We have designed an autonomous reflector based on mobile code that runs on top of an active network execution environment. The candidate locations for such reflector agents are active network (AN) (Tennenhouse et al., 1997) or active server (AS) nodes (Amir, 1998; Fry and Ghosh, 1999). These nodes run an execution environment (EE) capable of downloading and executing the reflector's code, and of discarding it when no longer used.

Our reflectors are autonomous and decide when to migrate to other nodes, clone in order to cope with increasing demand, merge with other reflectors, or disappear when no longer needed. The decisions are based solely on local knowledge available at the terminals or active nodes where they run. This guarantees that reflector code is deployed only where needed and when needed, and that after the session finishes all the reflectors will be automatically eliminated. The idea is that reflectors progressively move from the affected session members towards a well-known main centre of interest, until failure points are successfully bypassed, such that data coming from the main centre of interest can reach such members.

Additionally, reflectors that do not receive sufficient demand die out, and those which are overloaded spawn others to less loaded nodes. Using such a scheme, a tree of reflectors emerges as a result of failure detection, and disappears by itself when the failure is repaired. It should be noticed that the reflectors are not able to diagnose nor repair failures by themselves. Their objective is only to maintain application-level connectivity in the presence of network-level multicast failures. Network management mechanisms to detect and repair such failures are orthogonal and outside the scope of this work.

The paper is structured as follows: Section 2 gives a brief overview of the relevant concepts in our context as well as related work in the area. Section 3 describes the autonomous reflector scheme to build reflector trees. Section 4 explains the decision model that each agent adopts while building the tree. Section 5 shows some simulation results. Section 6 describes the current state of our Java implementation. Section 7 concludes the paper.

This article is an extended version of an earlier work (Yamamoto and Leduc, 2001a). Sections 2 and 3 have been reorganised and enhanced to clarify some ambiguities detected in the initial article. The merge procedure has been revised and is briefly described in Section 4.5.

2 Background

2.1 Multicast over the Internet

Multicast communication models for the Internet have received considerable attention since the early 1990s. However, multicast protocols are still not widely available on the global Internet, and the experimental Multicast Backbone has been slow to take off. Wide-area MBone sessions still fail due to multicast problems in some sections of the network. It is very difficult for the session participants to diagnose a failure and eliminate it during the lifetime of the session. The network conditions are unstable, it is difficult to monitor traffic and to detect points of failure, etc. The result is poor quality for the users.

There are several reasons for such a situation. One of them is the design of the pro-

tocols that usually requires modifications in the network routers and little support for incremental deployment. For a new protocol to be deployed over the Internet, it needs to be agreed upon, standardised, and manufacturers must implement compatible versions. Incremental deployment is difficult in this context, due to different paces of development and upgrade in different parts of the global network.

In the case of multicast, security concerns are also an obstacle to deployment, since multicast reinforces the risk that attackers easily flood the network with unwanted data. Therefore, many providers are reluctant to allow IP Multicast in their networks, and firewalls can block multicast traffic. Some instabilities come from the fact that multicast is still considered as an experimental service in many places, and therefore it is given low priority over the operational tasks.

New network-layer solutions such as REUNITE (Stoica et al., 2000) and HBH (Costa et al., 2001) propose the use of the standard unicast addressing model to build multicast distribution trees, such that unicast-only regions can be supported in a transparent way, and therefore facilitate incremental deployment. However, such solutions still require compatible peers, and would need to be agreed upon and standardised as other protocols, before being deployed. While these new protocols are being discussed, application-layer solutions such as Narada (Chu et al., 2000) offer the users an alternative to waiting for a larger scale availability of network-layer solutions. However, due to the lack of network support, application-layer solutions are difficult to implement and often lead to inefficient overlay topologies.

2.2 Active Networks and Active Servers

Many of the deployment difficulties described for multicast are also shared by other Internet protocols such as IPv6, Mobile IP, etc. Research on Active Networking (AN) (Tennenhouse et al., 1997) came as a response to such deployment difficulties, among other motivations. Active networking enables the dynamic deployment of protocols and services over a set of programmable routers. The nodes of an active network are capable not only of forwarding packets as usual but also of loading and executing mobile code. The code can come in the form of active extensions or capsules. Active extensions are complete modules that implement a given service, while capsules contain small pieces of code (or a reference to the code) that are executed in every AN node they visit.

Since AN raises security issues which are still being studied and debated upon, some researchers have proposed the Active Server (AS) approach (Amir, 1998; Fry and Ghosh, 1999) as a shorter term alternative to AN. AS nodes are end systems that allow the secure downloading of mobile code such that new services can be deployed on-demand.

2.3 Existing Reflector Systems

Several commercial and non-commercial reflector systems are available, e.g. (Highfield, 1998; Live Networks, Inc., 2000; Kirstein and Bennett, 2000) (with additional references on page 27 of (Kon et al., 2000)). These systems are typically software packages that must be manually installed at the sites that will provide the reflector service. Therefore, the location of reflector sites must be decided beforehand, and cannot be easily changed during the lifetime of the multimedia session. Changes in reflector location or configuration generally lead to temporary service disruption.

In (Baldi et al., 1998) mobile reflectors that can clone or migrate appear as part of a videoconference architecture for active networks. The authors focus on software design issues, and the actual algorithms and criteria for placing such reflectors are not covered.

A multicast reflector has been mentioned as an example application over the AS environment ALAN (Fry and Ghosh, 1999). It has been used to transmit MBone sessions via unicast to sites not connected to the MBone, and has the potential to move across ALAN nodes. However, dynamic reflector placement algorithms seem not to have been proposed in this context yet.

In (Kon et al., 2000) a distributed framework to manage a network of reflectors is proposed, based on dynamic code distribution and (re)configuration. Reflectors are used to support network and terminal heterogeneity. Within this framework, it is possible to manage networks containing a large number of reflectors. Each reflector has a limited degree of autonomy, such as reconfiguring the neighbour nodes to bypass a reflector that failed. For most other operations, however, the reflector elements need to be managed by a privileged user, the reflector administrator, who decides where to install new reflectors or to remove reflectors from the network. To take such decisions, the administrator needs to have a global view of the network topology and characteristics, which is not trivial to obtain from the wide-area Internet or the MBone. Another difficulty is to cope with participants dynamically joining and leaving the session, and the corresponding reflector tree reconfiguration in order to maintain a tree which always tracks the optimum. (Kon et al., 2000) report that in one experiment they were forced to deny approximately one million connection requests due to lack of bandwidth on the reflector sites. This could have been avoided if reflectors were able to automatically clone themselves to other sites in order to cope with the additional demand.

2.4 Market-based resource control

A considerable amount of research results in the area of market-based control are available mainly in the agents field (Clearwater, 1996). This work provides algorithms inspired by optimisation and economy theories for distributed control of resource usage, with many applications to computer and telecommunication networks. In (Gibney et al., 1999) a market-based mechanism to set up circuit switching paths with resource reservation is described. Closer to the AN perspective, in (Tschudin, 1997) an open resource allocation scheme based on market models is applied to the case of memory allocation for mobile code.

In (Najafi, 2001) a cost model for active networks is proposed, which takes into account the cost of processing a flow in the active nodes as well as the transmission costs. The author includes an algorithm that converges to the minimum cost for a flow that may be transformed in several active nodes before reaching its destination. He has also proposed a single agent positioning algorithm in which an agent can decide to reposition itself in the network in order to reduce the session cost.

2.5 Placing functionality and routing in active networks

The problem of choosing active nodes to place a given programmable functionality has been identified as a routing problem (Najafi, 2001; Choi et al., 2001). The potential of active routing is broadly discussed in (Maxemchuck and Low, 2001). The extensive simulation results in (Kiwior and Zabele, 2001) show that the performance of a reliable multicast protocol that makes use of active network nodes heavily depends on the location of the active nodes.

Several proposals in this area such as (Akamine et al., 2000; Duysburgh et al., 2000; Safaei et al., 2001; Choi et al., 2001; Partridge et al., 2001) require knowledge of the whole network graph in order to compute the active paths. Solutions of this kind are

feasible to place generic services to be used by a wide range of active applications within a domain. However, they generally do not scale beyond a single domain, and are not feasible when the applications themselves need to find optimum locations for their active elements, depending on their own specific characteristics and constraints.

In (Wen et al., 2001) a framework is proposed for composing customised multicast protocols for active networks out of elementary building blocks. Our approach could benefit from such a framework to build the reflector service using similar blocks.

2.6 Self-deploying services

In the context of agents and active networking, a number of proposals for self-deploying services have been made. In (Shehory et al., 1998) a framework is proposed in which agents deal with overload by cloning, passing tasks to others, merging, or dying. Agents decide when to clone according to the loads of the different resources they use, such as memory, processing and communication resources. The possible decisions that the agent can take are described by a decision tree, and the optimum decision is calculated via dynamic programming. In (Tschudin, 1999a) an election service based on active packets is developed, that deploys itself to every reachable node. In a later work (Tschudin, 1999b) the same author addresses the security issues involved with a necessary self-destruction mechanism for such kind of services.

In (Roadknight and Marshall, 2000), the issue of quality of service differentiation is addressed by using a distributed genetic algorithm inspired by the behaviour of bacteria. The authors show that the amount of servers and their location in the network evolve according to the user demand for a given type of service and a requested trade-off between latency and packet loss. The potential of genetic techniques such as the ones proposed by (Roadknight and Marshall, 2000) is the increased variability to find new solutions and adapt to new situations not envisaged at the beginning. However, in an environment where nodes and links are heterogeneous, propagating successful rules (“genes”) to neighbouring nodes might not necessarily be a good idea, since a rule that is successful in one node might fail completely in another node due to different resource constraints. In the context of self-organising systems, biologically-inspired and market-based techniques seem complementary, and an interesting research challenge would be to combine the best of both worlds to obtain new adaptation mechanisms.

3 Autonomous Reflectors

In this section, we describe our autonomous reflector scheme. We start with some definitions and assumptions, and then describe the basic mechanisms for building and destroying trees of reflectors.

3.1 Definitions and terminology

- Reflector, or reflector agent: Software package that implements the reflector functionality and can be loaded on the active nodes on-demand. Each reflector is uniquely identified within the session. Each active node may hold only one instance of a reflector for a given session (although several reflector instances for different sessions running on the same node may share the same code).
- Client reflector: Reflector A is said to be a client of reflector B when A is a child of B on the tree.

- Server reflector: Reflector B is said to be a server for reflector A when B is A's parent on the tree.
- Connection: A logical client-server connection between two reflectors A and B. Packets are exchanged over this logical connection using direct unicast addressing.
- Terminal reflector: A reflector that is a leaf of the tree.
- Intermediate reflector: A non-leaf reflector.
- Root reflector: The root of a reflector tree. It is a reflector agent that is either located at the RP or receiving data from the RP via native multicast.
- RP or rendez-vous point: Predefined target node towards which the reflector trees will be built.
- Downstream: The flow direction from the RP towards the root reflector, or from the root reflector towards the leaves of the reflector tree.
- Upstream: The direction from the tree leaves towards the root reflector or from the root reflector towards the RP.
- Clone: To send a copy of a reflector to another node; the clone will initially have its original reflector as a client.
- Migrate: To move to a given destination node, by sending a clone there, transferring the agent state to the clone (mainly its client list) and terminating.
- Merge: To combine two reflectors into a single one, by merging their client lists and terminating one of the reflectors. A merge operation is often used only as an abstraction, as two reflectors that intend to merge may negotiate a different configuration before the merging actually takes place (see Section 4.5).
- Terminate: To terminate the execution of a reflector at a given node (its code may remain cached for a certain amount of time until it is garbage collected).

3.2 Assumptions and Limitations

We assume that some basic default unicast routing service interconnecting all active nodes is available, such that at any moment it is possible for an active application to obtain the next hop to a given destination. This service can be either provided by the Execution Environment itself, or installed as active extension code with a well-defined interface exported to the active applications that need it. By default it can simply map directly to IP routing, but more sophisticated techniques such as application-layer routing (Ghosh et al., 2000) could also be available to provide optimised paths according to specific criteria. We assume that unicast routing is robust: rerouting around failed unicast links is out of the scope of this work.

The current failure detection mechanism is very simple: a reflector simply attempts to join the multicast group, listen for a while, and if no multicast packets arrive then it assumes that there is a failure. It also assumes a failure when there is an error while attempting to join the group at a given network interface, meaning that no multicast support is available for this interface.

Another assumption is that there is only one main centre of interest that generates content for the session (e.g. the lecturer's site). This centre of interest or main source

will also be called the rendez-vous point (RP). All other session participants are also allowed to generate content to the session, as it is generally the case with RTP sessions (Schulzrinne et al., 1996), that nowadays are widespread on the MBone. So the system is not constrained to Single Source Multicast. However, these other sources of data will be considered as secondary from the point of view of the mobile reflectors, which means that when there is a multicast failure affecting only the reception of secondary sources, the system of reflectors will not attempt to repair it.

It is assumed that all session members learn the RP address in advance, together with other group information, that is generally advertised by standard session announcement mechanisms such as SIP or SAP (Handley et al., 1999; Handley et al., 2000), which are outside the scope of the paper.

One might argue that assuming a single centre of interest is not a realistic approach, but in practice it is often the case that a single centre of interest exists or can be defined close to where most of the "action" occurs in the session. Besides that, if multicast is down for a particular member, it is likely to be down for other members too, but detecting it for each member individually would be too costly for large sessions. Therefore, the repair tree of reflectors is bidirectional, so that all participants affected by a failure share a single tree to distribute and receive content to/from the rest of the session.

The reflectors we propose are only able to repair multicast trees using unicast: multicast tree failures will cause a tree of reflectors to be formed, which will be connected to each other via unicast. Since multicast routing and active network unicast routing are independent, the unicast tree of reflectors might not coincide with the corresponding multicast subtree for a given set of session members. One can imagine that it would be interesting to use unicast only to bypass "broken" segments of a tree, using multicast everywhere else. This would result in a significantly smaller amount of reflectors being deployed. While this is an interesting possibility, it raises many new difficulties related to the self-organisation of disjoint subgroups within a session, with corresponding allocation of multicast group addresses, underlying topology discovery to make sure subgroups don't overlap, etc. Therefore we leave this possibility open for future study.

3.3 Tree Construction

The reflectors start at leaf nodes co-located or close to the session members, and then progressively move, clone and merge with other reflectors along their respective unicast paths towards a rendez-vous point (RP). As discussed in the previous paragraph, the RP role is typically assigned to the main source of content in the session. This leaf-initiated approach is similar to filter placement schemes based on RSVP, such as the AMNet prototype described by (Wittmann et al., 1998).

We distinguish two types of reflectors: terminal and intermediate. Terminal reflectors are located as close as possible to the end systems and do not move, while intermediate reflectors are dynamically placed on other active nodes in the network, and might move from one node to another according to the network conditions.

A terminal reflector ideally serves one local client, which is the user application that generates and/or treats session content (e.g. MBone tools such as `vic` or `rat`). A terminal reflector must be installed at each end system that wishes to make use of autonomous reflectors, or as close as possible to the end system (or set of end systems) to be served. The terminal reflector works as a proxy between the actual multicast group and the user application, so that the direct use of multicast or the use of reflectors is hidden from the application. This allows the use of reflectors based on mobile code without requiring any change to existing applications. Terminal reflectors are intermediate reflectors that have

their migration rules disabled. They must be fixed because the existing tools are not able to detect moving peers.

The tree of reflectors organises itself in a client-server hierarchy. Each intermediate reflector serves a number of clients that are located downstream from it and directly connected to it via unicast. All intermediate reflectors are both servers for a number of clients and clients of an upstream reflector, except the root of the tree which only acts as a server.

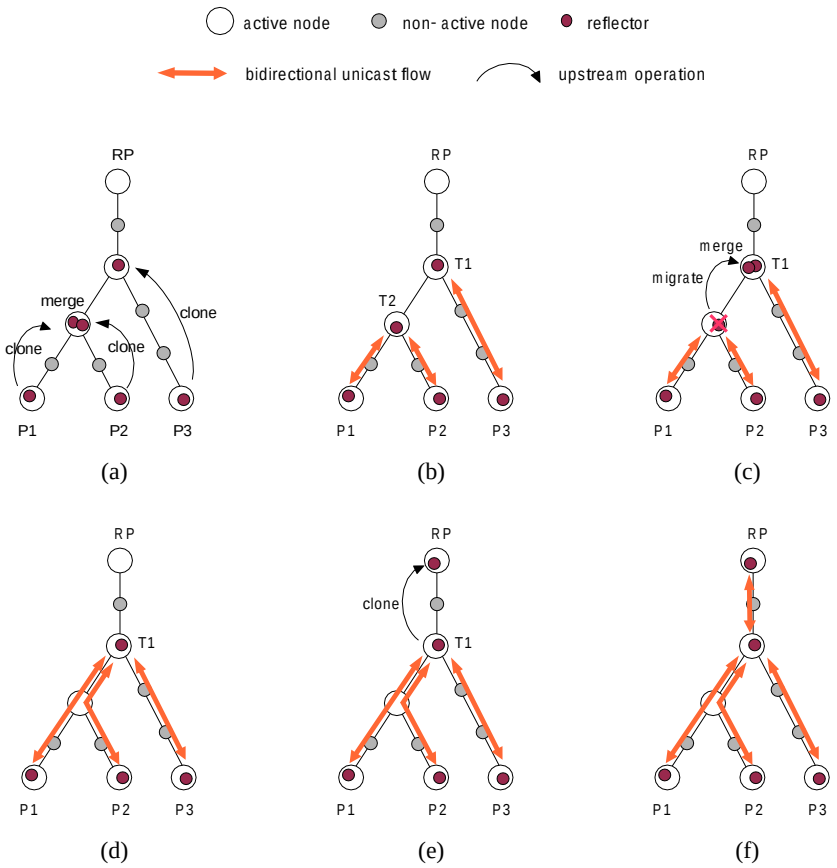


Figure 1: Tree construction example.

Figure 1 shows a tree construction example for a session consisting of an RP and three session members P1, P2, and P3. When a participant detects the absence of native multicast connectivity to the RP (by trying to listen to the multicast group as described in Section 3.2), its terminal reflector sends another reflector to the next AN hop towards the RP. This operation is called *upstream cloning* (Fig. 1(a)). The clone reflector spawned in this way is not an exact copy of the original reflector, since it is an intermediate reflector, but it serves the same session and has the same goals. In our case, the goal is to maintain application connectivity when native multicast fails or is absent, such that at least the data coming from the main session source reaches all session members.

When two reflectors belonging to the same session meet at the same active node, they merge into a single reflector (Fig. 1(a)(c)). Two reflector trees, T1 and T2 (Fig. 1(b)), result from the operations shown in Fig. 1(a). If native multicast traffic from the RP is detected by an intermediate reflector, it becomes a root reflector and stays in the node.

Otherwise, it either clones or migrates to the next active hop towards the RP. Figure 1(c) shows an *upstream migration*. A hierarchy of reflectors results from this process (Fig. 1(d)(e)(f)).

Since each agent reflects every packet received, the result is a bidirectional shared tree such as CBT (Ballardie, 1997). For example, in Figure 1(d), if T1 is receiving multicast from RP, then the flow coming from RP will be sent in unicast to P1, P2, and P3; and the flow coming from P1 will be copied to P2, P3, and to the multicast group.

While network-layer protocols typically deal with outgoing interfaces, the reflectors deal with unicast client connections directly: a copy of each packet is sent to each client, even when several clients share the same outgoing interface. A packet is never looped back to a client, even in the presence of route and interface asymmetry such that the incoming and outgoing interfaces to a given client are different. This strict hierarchy must be observed at all times in order to ensure that loops in the bidirectional tree do not occur.

Each reflector includes a local selector that selects data from either multicast or unicast channels, to ensure that no duplicate packets are forwarded downstream, and that packets coming from downstream are forwarded only to the selected channel (either multicast or parent reflector).

Applying this bottom-up tree construction algorithm, a reflector that succeeds bypassing the failure point becomes the root of its reflector tree. This method that not only one tree but several ones might arise in response to an absence of native multicast, in case multiple reflectors cross a failure point at different nodes. This can happen, for example, if the failure "point" is not a single link but a whole network with multicast capabilities disabled for some reason.

The repair trees will be located as close as possible to the concerned participants, and will not interfere with the rest of the session running in native multicast. The drawback is that the tree is built following the reverse path to the RP, which might lead to suboptimal downstream paths when routes are asymmetric.

3.4 Termination

A reflector that runs out of clients automatically terminates itself. If native multicast connectivity is somehow restored, reflectors start receiving data from the main source via the multicast channel, and disconnect from their upstream reflectors. The latter will eventually die out due to lack of clients. Each reflector contains a local selector module that is responsible for discarding duplicate packets, and for ensuring that packets going upstream are forwarded to the selected channel (either native multicast or parent reflector).

If a reflector terminates abnormally, its children will detect the absence of traffic coming from their parent and will restart the tree construction process to rebuild the affected portion of the tree.

3.5 Communication among reflectors

AN Capsules are used to implement a signalling mechanism among neighbouring reflectors so that a given intermediate reflector can inform its downstream clients of its current location, and to keep the reflector tree alive. Capsules are also used to prospect the state of an upstream node before making a decision to clone or migrate, and to enable two agents to take merge decisions jointly, as will be explained in Sections 4.4 and 4.5.

4 Reflector decisions

Reflectors can use resource control based on market mechanisms (Clearwater, 1996) to make decisions to either clone, merge, migrate or terminate themselves. Such mechanisms can also be used to dynamically decide on the maximum number of clients to accept at a given machine, in order not to cause link or CPU overload. Another usage is to make downstream cloning decisions: for example, in the case of an overloaded reflector, a number of clones can be sent downstream to handle part of the clients.

In this paper, we concentrate on the basic decision mechanisms related to code mobility during tree construction and destruction, that is, cloning, migrating, merging and terminating. We are currently working on the additional mechanisms related to tree reshaping and load control.

Each reflector has costs associated with its consumption of node and network resources. The tree of reflectors is organised in a client-server hierarchy, such that server reflectors sell session data to their clients, and buy data from their server reflector. A reflector uses the revenues that come from its clients to pay for resource usage in the active nodes and for the services of the upstream reflector.

4.1 Resource usage costs

Each reflector has associated fixed costs and variable costs for the use of node resources. The fixed costs do not vary with the number of clients that a reflector has, and correspond to the costs of using the mobile code platform. They represent the minimum processing plus storage cost that the mobile agent incurs, even when no clients are connected. Note, however, that the fixed costs are not constant in general, as they may vary as a function of the load level of the resource in question (CPU, memory).

The variable costs increase with the number of clients, and correspond to the link transmission costs to all clients, plus the processing costs for all packets. These costs may also vary according to the total load of the corresponding resource (link bandwidth or processing).

From the cost point of view, having many clients is good for a reflector because the fixed costs are shared among all the clients, but if the number of clients becomes too large, the demand for one or more resources might exceed the supply (congestion situation), leading to an increase in processing and link prices, with possible packet losses and consequent degradation in quality for the end user.

Every time a new reflector is added to the tree, there is an increase in costs corresponding to the resources that the new reflector needs. However, this increase might be compensated by a decrease in costs for other reflectors, e.g. because their load is alleviated.

The cost of processing at an active node is also related to the delay penalty imposed to the end user due to the use of a tree of reflectors. The delay penalty is the ratio between the actual delay experienced by an end user and the delay that would be experienced if the user could connect directly to the multicast session without the help of reflectors. If the processing power were infinite, the extra delay imposed by a reflector would be null. On the other hand, a very low processing power would incur a high additional delay. The same is valid for a machine with high processing power but which is overloaded, such that the processing time available to a reflector is very low. Therefore, if a reflector tries to choose nodes that have low processing costs, it is likely to be moving towards a lower delay penalty for its users.

4.2 Definitions

We begin by providing some definitions of terms that will be used later in this section:

n_i: a reflector that runs at a given node i.
nc: number of clients of reflector n_i.
$r_{i,j}$: reflector j, the j-th client of reflector n_i, for $j = 1, \ldots, nc$.
$R_{i,j}$: data sending rate of reflector $r_{i,j}$ to n_i (upstream direction).
S: data sending rate of the main source.
nk: number of terminal reflectors in session.

SR: total rate of the session (session bandwidth)

$$SR = S + \sum_{1 \leq k \leq nk} R_k \tag{1}$$

for all terminal reflectors r_k with sending rate R_k.

cf_i: fixed costs at node n_i (do not vary with nc).
$cv_i(nc)$: variable costs at node n_i (vary with nc).
$cvp_i(nc)$: processing costs at n_i; depend on the amount of data treated per second.
$cvl_i(nc)$: total link costs at n_i: represent the costs associated with the total amount of bandwidth emitted by reflector n_i to each link that leads to clients of n_i, and to the parent reflector if any.

$ct_i(nc)$: total cost at node n_i when nc clients are present: the sum of fixed and variable costs, as follows:

$$\begin{aligned} ct_i(nc) &= cf_i + cv_i(nc) \\ &= cf_i + cvp_i(nc) + cvl_i(nc) \end{aligned} \tag{2}$$

4.3 Estimating costs

In order to make a decision to either clone or to migrate, a reflector first needs to estimate the costs that would result from choosing either option. A simple decision strategy would then be just to choose the configuration with the lowest cost. However, there are a number of difficulties in obtaining such estimation. Actually, this is a typical problem of making decisions in the presence of risks, and decision analysis could be applied here, as in (Shehory et al., 1998). In this section, we present a first simplified approach to the problem. Further research is necessary in order to extend it to a more general case.

One of the main difficulties is that, at the beginning, when the reflector still hasn't reached the main source (directly via multicast or via another server reflector), it is not able to measure the actual resource consumption that will result when it reaches it. When that happens, it goes into full operation mode, but at this moment, it is too late to revise its previous decisions concerning cloning or migrating. Especially, if the reflectors underestimate the aggregate sending rates of all the session members beyond the multicast failure point while building the tree, several points of congestion might appear as soon as the tree becomes fully operational.

A solution to this problem would be to rely on an estimation of the total rate of the session (session bandwidth), that must be available somehow before the session starts. In practice it is possible to obtain such information by looking at the media types in the

SDR session announcements. Additionally, if RTCP is used (Schulzrinne et al., 1996), and assuming that only a limited number of session members send significant amounts of data to the group, the session bandwidth grows very little with the total session size.

Using such an upper bound, resources could be reserved at the active nodes along the path in order to guarantee that enough resources are available when the reflectors reach the main source. However, resource reservation might not be available at all nodes, and most of the nodes might not even be active. Besides that, if the session bandwidth is overestimated, too many costs might incur with little extra benefit for the end user. We adopt a simple solution that relies on an upper bound on the session bandwidth to simplify the cost calculations, but does not reserve resources on the nodes.

Next, we quantify each cost component in our context. We begin with the processing costs.

4.3.1 Processing costs

Network packets constitute the bulk of the data treated by a reflector. Therefore, the processing costs during a given interval increase with the number and size of the packets treated. Every packet received is reflected to everyone else. Thus every packet from the upstream channel (reflector or multicast) is copied to every client, and every packet from a client is copied to the upstream channel plus all the other clients except itself. For a reflector that has already reached the main source (directly via multicast or via another server reflector), the total number of bits per second treated at n_i is:

$$st_i(nc) = sp_i(nc) + sc_i(nc) \tag{3}$$

where:
$sp_i(nc)$ is the data rate sent from the parent to all child reflectors of n_i
$sc_i(nc)$ is the data rate sent from all child reflectors of n_i to all others and to the parent.

$$sp_i(nc) = nc \cdot (SR - \sum_{1 \leq j \leq nc} R_j) \tag{4}$$

$$\begin{aligned} sc_i(nc) &= \sum_{1 \leq j \leq nc} ((nc-1) \cdot R_j + 1 \cdot R_j) \\ &= nc \cdot \sum_{1 \leq j \leq nc} R_j \end{aligned} \tag{5}$$

Substituting equations 4 and 5 in 3, we have:

$$st_i(nc) = nc \cdot SR \tag{6}$$

Assuming constant prices, and processing costs that increase linearly with the data rate treated, we have:

$$\begin{aligned} cvp_i(nc) &= pp_i \cdot st_i(nc) \\ &= pp_i \cdot nc \cdot SR \end{aligned} \tag{7}$$

where:
pp_i is the (constant) processing price per bit per second at node n_i.

4.3.2 Link costs

The link usage costs include the costs for bandwidth and queuing. Here we consider only the bandwidth costs for simplification. There are only costs associated with the transmission of packets, not with the reception of packets. Thus the link costs are the sum of the costs to reflect a packet from the parent reflector to all child reflectors, and from each child to every other child plus the parent.

Assuming constant link prices, the total link cost for n_i can be written as:

$$cvl_i(nc) = pl_{i,p} \cdot \sum_{1 \le j \le nc} R_j + \sum_{1 \le j \le nc} (pl_{i,j} \cdot (SR - R_j)) \tag{8}$$

where:
$pl_{i,j}$ is the price per unit of bandwidth on the link in n_i that leads to the client $r_{i,j}$.
$pl_{i,p}$ is the price per unit of bandwidth on the link in n_i that leads to the parent reflector of n_i (or the candidate parent in case a decision to clone or to migrate is about to be made).

If the link price is the same for all clients and equal to $pl_{i,l}$, or when all clients of n_i share the same link l, we can rewrite the link cost as:

$$cvl_i(nc) = pl_{i,p} \cdot \sum_j R_j + pl_{i,l} \cdot (nc \cdot SR - \sum_j R_j) \tag{9}$$

4.3.3 Cost of the cloning configuration

If we are going to send a clone from the origin node n_i to an upstream destination node n_d, the cost of the resulting clone configuration can be calculated as:

$$\begin{aligned} ctc_{i,d} &= ct_i(nc) + ct_d(1) \\ &= cf_i + cvp_i(nc) + cvl_i(nc) + cf_d + cvp_d(1) + cvl_d(1) \end{aligned} \tag{10}$$

Here $ct_i(nc)$ represents the total cost of running the agent at the current node when the agent is fully operational, while $ct_d(1)$ is the cost of a new agent running at the upstream node n_d with a single node (n_i) as a client.

4.3.4 Cost of the migration configuration

When migrating to an upstream destination n_d, a reflector n_i carries its client list along with it. Assuming symmetric unicast routing paths, the traffic will continue to go through node i, therefore consuming the same amount of bandwidth resources at the links leading to each client reflector. Since the reflector itself will disappear from node n_i, there are neither fixed nor processing costs associated with it anymore at this node. Therefore, the cost of the resulting configuration after migration can be calculated as:

$$\begin{aligned} ctm_{i,d} &= cvl_i(nc) + ct_d(nc) \\ &= cvl_i(nc) + cf_d + cvp_d(nc) + cvl_d(nc) \end{aligned} \tag{11}$$

4.4 Making a decision

We would like to make a decision to either clone or migrate, based on the total costs of resources for each configuration.

A simple decision strategy is to choose the configuration with the lowest cost:

$$\text{if } ctm_{i,d} > ctc_{i,d} \text{ then clone else migrate.} \tag{12}$$

In order to simplify the calculations, we rewrite the above rule as:

$$\text{if } ctm_{i,d} - ctc_{i,d} > 0 \text{ then clone else migrate.} \tag{13}$$

The costs of each configuration are given by equations 11 and 10, therefore we have:

$$\begin{aligned} ctm_{i,d} - ctc_{i,d} = {} & cvp_d(nc) - cvp_d(1) - cvp_i(nc) \\ & + cvl_d(nc) - cvl_d(1) - cf_i \end{aligned} \tag{14}$$

Note that the fixed costs at n_d, as well as the link costs at n_i have disappeared since they are the same in both configurations. With symmetric unicast paths and no multipath, all the traffic from n_d to n_i will go through a single link. Thus, assuming linear costs for link resources, we can use equation 9 to calculate the terms $cvl_d(nc)$ and $cvl_d(1)$. Assuming linear costs also for processing resources, equation 7 can be used to calculate $cvp_d(nc)$, $cvp_d(1)$, and $cvp_i(nc)$. After these operations we obtain:

$$ctm_{i,d} - ctc_{i,d} = SR \cdot ((nc-1) \cdot (pp_d + pl_{d,i}) - nc \cdot pp_i) - cf_i \tag{15}$$

which can also be written as:

$$\begin{aligned} ctm_{i,d} - ctc_{i,d} = {} & SR \cdot nc \cdot (pp_d + pl_{d,i} - pp_i) - cf_i \\ & - SR \cdot (pp_d + pl_{d,i}) \end{aligned} \tag{16}$$

The first line of the right side of equation 16 represents the increase in costs associated with the migration configuration, while the last line corresponds to the increase associated with the cloning configuration. When nc is high, the migration configuration tends to become more expensive than the cloning configuration. Therefore, the cloning configuration will generally be preferred for high nc, unless the fixed or processing costs at n_i are prohibitive.

With this result, we obtain an easy way to make a decision, by using equation 15 to choose the cheapest configuration. As discussed earlier, an estimation on the upper bound of SR is considered available before the session starts. The number of clients, nc, is known at n_i, as well as the local cost cf_i and price pp_i. Consequently, before making a decision, the agent needs to obtain the following information from its neighbour n_d:
pp_d: processing price per unit
$pl_{d,i}$: link price per unit for the outgoing interface from n_d to n_i.

The information above is collected by a Prospecting capsule that is sent to the destination node before the cloning or migration action actually takes place. The costs for the intermediate links on the direct and reverse paths between nodes i and d have to be taken into account as well. The Prospecting capsule partially does this by accumulating into $pl_{d,i}$ the sum of the link costs for the active nodes on the path from node d to node i. When some nodes are inactive, an approach similar to the equivalent link abstraction (Sivakumar et al., 2000) could be used to estimate the transmission costs of a non-active network cloud.

This is the strategy adopted to obtain the simulation results shown in this paper. Although it seems a little simplistic, this strategy already takes into account an important criterion which is the delay penalty for the user which is imposed by the use of the reflectors instead of native IP multicast. As discussed earlier, this delay penalty is implied within the processing costs.

In classical multicast algorithms such a decision dilemma usually doesn't apply, because only link resources are typically taken into account. In this case we can make $pp_d = pp_i = cf_i = 0$, and our calculations reduce to:

$$ctm_{i,d} - ctc_{i,d} = SR \cdot pl_{d,i} \cdot (nc - 1) \quad (17)$$

In the above we have:
For $nc > 1 : ctm_{i,d} - ctc_{i,d} > 0$ and we choose to clone.
For $nc \leq 1 : ctm_{i,d} - ctc_{i,d} \leq 0$ and we choose to migrate.

These observations confirm that when bandwidth is the only scarce resource, cloning is the default choice except in the trivial case ($nc \leq 1$), since migration always implies duplicating packets on the link from n_d to n_i when $nc > 1$, and therefore causes the total costs to increase.

4.5 Merging

When two reflectors belonging to the same session meet at the same active node, they merge into a single reflector. This operation involves the union of both client lists and any other necessary information. Since this might result in resource overload, a preliminary negotiation between both agents is desirable to achieve favourable configurations. For instance, when sending the Prospecting capsule to an upstream neighbour to find out about costs, the capsule could also be programmed to look for the presence of another reflector for the same session, and check its current resource consumption. An outcome of the negotiation could be that the server delegates some of its own clients to the prospecting reflector, in order to balance the load and reduce costs.

It is possible to show that assuming linear costs and symmetric paths, the same rule (Rule 13) with equation 15 can still be applied to take a joint merge decision involving two reflectors. Due to lack of space, the analysis is not shown here but can be found in a separate report (Yamamoto and Leduc, 2001b). The resulting merge procedure is to consider as if all clients of the reflector at node n_d that share the outgoing interface towards n_i ($nc_{d,i}$) were attached to node n_i so that we can make $nc \leftarrow nc + nc_{d,i}$ in Equation 15; and then apply Rule 13. If the rule says "migrate" then the reflector at node n_i migrates to n_d. Otherwise ("clone"), and n_i attaches itself to n_d as a client, and n_d transfers its $nc_{d,i}$ clients to n_i.

To implement this mechanism, a Prospecting capsule is first sent from node n_i to n_d. The capsule goes back to n_i with the values of pp_d and $pl_{d,i}$ and $nc_{d,i}$ (which is zero when no reflector for the group is running at n_d). The reflector at n_i then uses these values in Rule 13 to take a local decision to clone or to migrate. If the rule advises a clone decision and $nc_{d,i} > 0$, then the reflector at d will take action to transfer its $nc_{d,i}$ clients to n_i, after the cloning operation has been successfully completed. Otherwise everything occurs as described in Section 4.4.

4.6 Terminating

Since reflectors must "pay" for resource usage in the active nodes, and their clients are their sole source of income, they will be automatically eliminated by the active platform when there are no further clients. However, there is a risk that sudden changes in load can make prices increase in unpredictable ways, causing fully operational reflectors to die out prematurely.

In our current implementation, this problem is still not solved, and in our view it can only be solved with the help of load control operating at shorter time scales than the ones

in which the reflectors operate. This requires adaptive (elastic) flows or transcoding, and here we are assuming that the reflectors merely repair connectivity failures, and don't interfere with the session data contents.

5 Simulation results

We have performed some simulation experiments using ns-2, in order to visualise the tree construction and destruction mechanisms. The topology for the simulations is illustrated in Figure 2 (Left). All links have a fixed capacity of 10Mbit/s and a propagation delay of 10ms. The main session source is located at node S. The leaves of the tree contain terminal reflectors that join the session at random times from t=0s to t=10s. All nodes are active and have the same prices for resources: $pp = 1$, $pl = 1$, and $cf = 1 \cdot 8 \cdot C$ where C is the size of the mobile code in bytes, and is currently set to 50000, which is the current approximate size of the bytecode in our Java prototype.

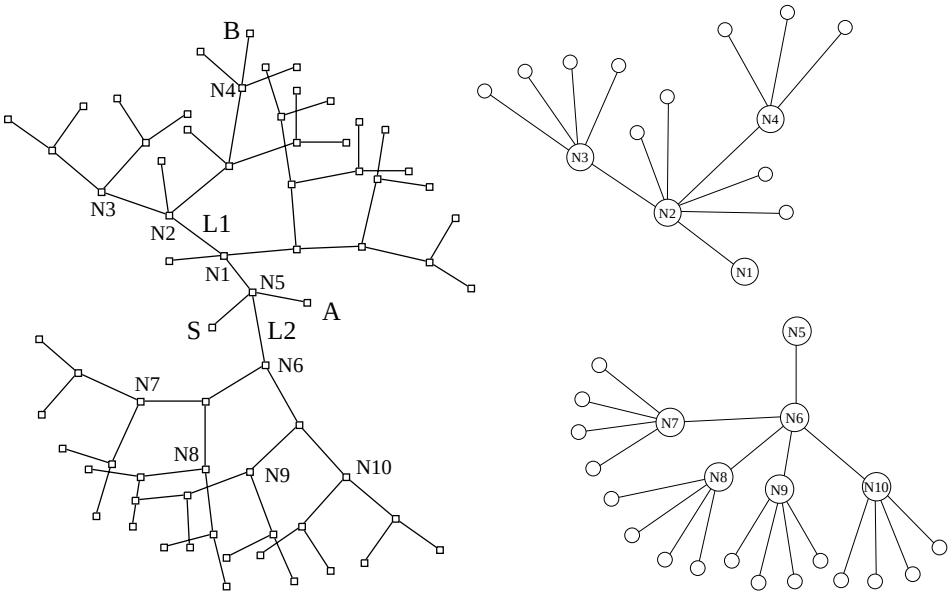


Figure 2: Left: Topology used in the simulations. Right: Two sample reflector trees over the topology on the left. Top Right: tree that results from the failure of L1, rooted at N1. Bottom Right: tree that results from the failure of L2, rooted at N5.

The multicast communication via links L1 and L2 is interrupted at t=20s. As a result, two reflector trees appear. Both trees starts at around t=24s, but the tree on the upper side of the topology is ready at t=29s, while the second one is only ready at t=37s. After this construction phase all terminal reflectors affected by failures are served by an intermediate reflector. The resulting trees are shown in Figure 2 (Right).

At t=70s the multicast communication via L1 and L2 is restored. Most reflectors detect this a couple of seconds later, and disconnect from their parent reflectors, which die out between t=77s and t=80s.

Figures 3(A), (B), and (C) show the aggregate session data rate received by three sample session participants: A, B, and C, respectively, whose location on the tree can be

observed in Figure 2 (Left). The main source rate is 500kbps while all the other session members send around 10kbps each.

Participant A happened to join the group at around t=10s, while B joined at the beginning, t=0s. During the failure period, although the multicast feed to node A is up and running, it receives less aggregate traffic until the reflector tree is fully operational, since during this period it doesn't receive the multicast packets coming from the participants that have stayed on the other side of the failure point. Participants B and C suffer from the failures until about t=30s and t=38s, respectively. After that, their respective level of reception becomes about the same as the one of A, as if they were also unaffected by the failure. When the multicast feed is restored, sudden peaks of traffic arrive at B and C, due to duplicate packets sent once via multicast and again via the reflector. These packets are eliminated by the terminal reflectors before being sent to the applications. We can verify this by looking at the sequence numbers received by the decoder connected to C, on Figure 3.

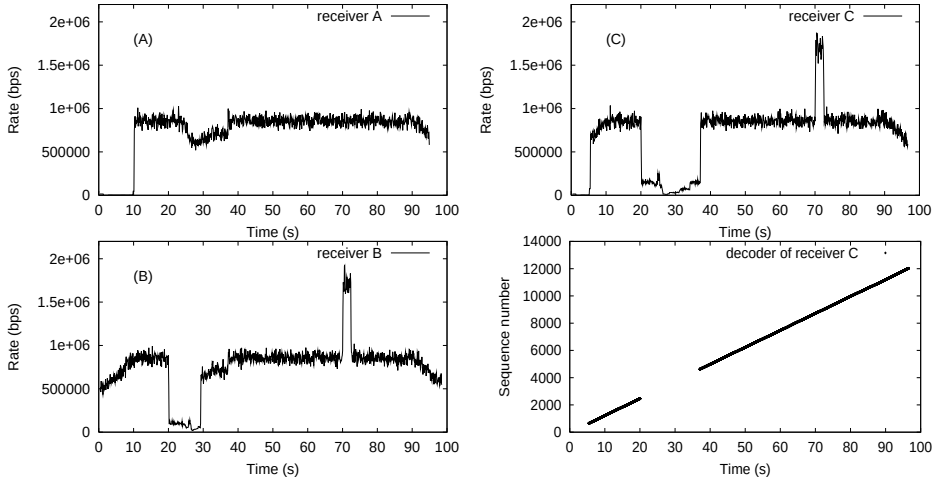


Figure 3: (A), (B), and (C): Data rate of three sample participants. (A): unaffected by multicast failure. (B) and (C): affected by failures of L1 and L2 respectively. (D): Sequence numbers received by the decoder of participant C.

In order to visualise the dynamics of the mobile code operations of migrating, moving, merging and terminating, we describe them in Figure 4 for the upper reflector tree on the topology. For compactness, we number events in time with integer numbers starting from 1, followed by the code of the operation performed. From t=24s to t=26s, all terminal reflectors spawn upstream clones. Since the order in which each terminal reflector sends a clone won't have any influence on the subsequent operations, we assign event number 1 to all. This is indicated as "1C" in the figure. The next event is event 2, and it's a cloning operation from node N4 towards its upstream neighbour. This is indicated as "2C". By following the sequence of events in this way, it is possible to track the main actions that lead to the tree configuration shown in Figure 2 (Top Right), and to its subsequent destruction ("T" operation). Although the merging operations are not indicated, they can be deduced as well, since they occur whenever a reflector arrives at a node where another one is already present.

The results above are intended to illustrate the basic behaviour of our autonomous reflectors in ideal conditions. They are not intended to show a realistic picture of a real

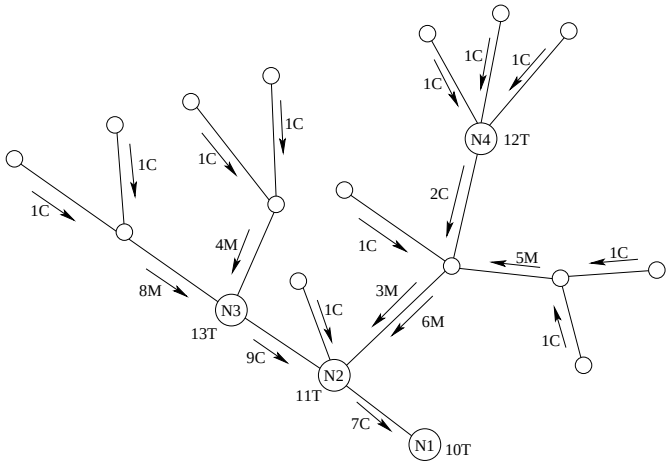


Figure 4: Dynamics of mobile code operations. The event numbers are shown beside the arrows or the node names, followed by the code of the operation: C (clone), M (migrate), T (terminate).

network. The topology is regular, the network is unloaded, all nodes are active, the delays are short and the paths for unicast and multicast traffic coincide.

In our simulations, we have noticed little impact of increased network latencies or moderate load on the results, even when the propagation delay on each link is increased to the order hundreds of milliseconds. However, we have often observed much larger latencies for joining live MBone sessions, as well as variable loss patterns. Thus we can expect higher reaction times for our reflectors in such a situation.

Further results, including the revised merge configuration, the impact of varying node and link prices, and of the amount of non-active nodes can be found in (Yamamoto and Leduc, 2001b). We are currently working on random topologies with concurrent sessions to assess the loading sharing capabilities and the distance to the global optimum.

6 Implementation

We are currently implementing a prototype of the mobile reflectors in Java using an architecture that allows the code to be easily ported to any EE that supports active extensions with minor modifications. The architecture is organised in three planes: data plane, monitoring plane, and control plane. This structure roughly follows the one suggested by (Blair et al., 1999).

The data plane is responsible for the blind forwarding of multicast and unicast data. Its core is inspired by the Mug reflector (Highfield, 1998): in Mug, a node that sends a RTP/UDP packet to the reflector is added to its client list; a client that stays idle (i.e. sends no more packets) for some time is removed from the client list. A selector is attached to the Mug-like core in order to switch between the multicast and the unicast upstream channels. The selector is controlled by the control plane. The data plane treats packets seamlessly whether they come from another reflector or from a multicast application such as the MBone media tools `vic` or `rat`.

The monitoring plane keeps track of the current resource usage and performance pa-

rameters of the data plane. To monitor CPU usage, we are currently integrating the CPU accounting facilities provided by J-Seal2 portable resource control framework (Binder et al., 2001). This requires some adaptations to the calculations in Section 4 to take into account measured resource consumption values besides estimated ones.

The control plane uses the data available in the monitoring plane to make decisions. Its core is a state machine with transitions triggered by events generated at the monitoring plane.

This architecture allows new strategies to be easily added to the control plane without affecting the other planes. It also maps naturally to the Bond platform (Bölöni and Marinescu, 1999), which opens up future possibilities for dynamic updates to the state machine through "agent surgery" (Bölöni and Marinescu, 1999). The communication mechanism among neighbouring reflectors takes the form of capsules such as in ANTS (Wetherall et al., 1998). Alternatively, the communication could be made via an existing agent message passing mechanism (e.g. Bond, see (Bölöni and Marinescu, 1999)). Both offer extra flexibility for enhancements and preclude the need to specify application-specific message formats and develop the corresponding parsers.

7 Conclusions and Future Work

We have described a decentralised scheme based on mobile code, to build a loosely connected network of autonomous reflectors that seeks to maintain session connectivity in the presence of multicast failures. The self-organising nature of the scheme ensures its robustness, scalability and autonomy properties, which make it suitable for sessions of any size, while minimising the necessary amount of human intervention.

For the moment each reflector treats only one media stream (e.g. either audio, or video, or whiteboard). In order to deal with several media, we plan to group multiple physical reflectors (each treating one media type) into a single logical reflector for cloning and migration purposes. In the near future, experiments over the MBone can be envisaged in the framework of the European COST Action 264, and with the help of existing active network overlays such as the ABone.

We plan to integrate the work presented in this paper with previous work on congestion control (Yamamoto and Leduc, 2000b; Yamamoto and Leduc, 2000a) such that reflectors also perform application-oriented filtering and/or transcoding of data in the presence of congestion, in a network which is likely to be only sparsely populated by active nodes. Other possible extensions include: exploring alternative paths, supporting strong route asymmetries and non-linear costs, multiple or changing centres of interest, QoS guarantees. It would also be interesting to generalise the technique for other group applications that require self-organisation.

Acknowledgements

This work has been carried out within the TINTIN project funded by the Walloon region in the framework of the programme *"Du numérique au multimédia"*. Part of this work was performed while the main author was a visiting researcher at Lancaster University. We would like to thank David Hutchison, Steven Simpson, Mark Banfield, Laurent Mathy, Stefan Schmid, and Katia Saikoski for their helpful support. We would also like to thank Allex Villazón (University of Geneva), Sandrine Calomme (University of Liège), and the anonymous reviewers for their insightful comments.

References

Akamine, H. et al. (2000). An Approach for Heterogeneous Video Multicast Using Active Networking. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 157–170, Tokyo, Japan.

Amir, E. (1998). *An Agent-based Approach to Real-time Multimedia Transmission over Heterogeneous Environments*. Ph.D. dissertation, University of California at Berkeley.

Baldi, M., Picco, G. P., and Risso, F. (1998). Designing a Videoconference System for Active Networks. In *Mobile Agents'98*.

Ballardie, A. (1997). Core Based Trees (CBT) Multicast Routing Architecture. Internet rfc 2201 (experimental), IETF.

Binder, W., Hulaas, J. G., Villazón, A., and Vidal, R. (2001). Portable Resource Control in Java: The J-SEAL2 Approach. In *ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA-2001)*, Tampa Bay, Florida, USA.

Blair, G. S., Andersen, A., Blair, L., and Coulson, G. (1999). The Role of Reflection in Supporting Dynamic QoS Management Functions. In *IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, London, UK.

Bölöni, L. and Marinescu, D. C. (1999). A Multi-Plane State Machine Agent Model. Technical Report CSD-TR 99-027, Purdue University. Also a poster at the Fourth International Conference on AUTONOMOUS AGENTS (Agents 2000) Barcelona, Spain, June 2000.

Choi, S. Y., Turner, J., and Wolf, T. (2001). Configuring Sessions in Programmable Networks. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, Alaska.

Chu, Y., Rao, S. G., and Zhang, H. (2000). A Case for End System Multicast. In *Proceedings of ACM Sigmetrics*, pages 1–12, Santa Clara, CA, USA.

Clearwater, S. H., editor (1996). *Market-based Control: A Paradigm for Distributed Resource Allocation*. World Scientific Publishing.

Costa, L. H. M. K., Fdida, S., and Duarte, O. C. M. B. (2001). Hop by Hop Multicast Routing Protocol. In *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, USA.

Duysburgh, B. et al. (2000). Data Transcoding in Multicast Sessions in Active Networks. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 130–144, Tokyo, Japan.

Fry, M. and Ghosh, A. (1999). Application level active networking. *Computer Networks*, 31(7):655–667.

Ghosh, A., Fry, M., and Crowcroft, J. (2000). An Architecture for Application Layer Routing. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 71–86, Tokyo, Japan.

Gibney, M., Jennings, N., Vriend, N., and Griffiths, J. (1999). Market-based call routing in telecommunications networks using adaptive pricing and real bidding. In *Proceedings of the IATA'99 Workshop*, Springer LNAI 1699, Stockholm, Sweden.

Handley, M., Perkins, C., and Whelan, E. (2000). Session Announcement Protocol. Internet rfc 2974 (experimental), IETF.

Handley, M., Schulzrinne, H., Schooler, E., and Rosenberg, J. (1999). SIP: Session Initiation Protocol. Internet rfc 2543 (standards track), IETF.

Highfield, J. (1998). Mug multicast packet reflector. URL http://www.stile.lboro.ac.uk/ cojch/mug/mug.html.

Kirstein, P. T. and Bennett, R. (2000). RE 4007 MECCANO Project Final Report. URL http://www-mice.cs.ucl.ac.uk/multimedia/projects/meccano/ deliverables/.

Kiwior, D. and Zabele, S. (2001). Active Resource Allocation in Active Networks. *IEEE JSAC*, 19(3):452–459.

Kon, F., Campbell, R., and Nahrsted, K. (2000). Using Dynamic Configuration to Manage A Scalable Multimedia Distribution System. *Computer Communication Journal*. Elsevier Science, Fall 2000.

Live Networks, Inc. (2000). URL http://www.live.com/.

Maxemchuck, N. F. and Low, S. H. (2001). Active Routing. *IEEE JSAC*, 19(3):552–565.

Najafi, K. (2001). *Modelling, Routing and Architecture in Active Networks*. Ph.D. dissertation, University of Toronto, Canada.

Partridge, C., Snoeren, A. C., Strayer, W. T., et al. (2001). FIRE: Flexible Intra-AS Routing Environment. *IEEE JSAC*, 19(3):410–425.

Roadknight, C. and Marshall, I. W. (2000). Differentiated Quality of Service in Application Layer Active Networks. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 358–370, Tokyo, Japan.

Safaei, F., Ouveysi, I., Zukerman, M., and Pattie, R. (2001). Carrier-Scale Programmable Networks: Wholesaler Platform and Resource Optimization. *IEEE JSAC*, 19(3):566–573.

Schulzrinne, H., Casner, S. L., Frederick, R., and Jacobson, V. (1996). RTP: A Transport Protocol for Real-Time Applications. Internet RFC 1889 (update in progress).

Shehory, O., Sycara, K., Chalasani, P., and Jha, S. (1998). Agent Cloning: An Approach to Agent Mobility and Resource Allocation. *IEEE Communications Magazine*, pages 58–67.

Sivakumar, R., Han, S., and Bharghavan, V. (2000). A Scalable Architecture for Active Networks. In *Proceedings of IEEE OPENARCH 2000*, Tel-Aviv, Israel.

Stoica, I., Ng, T. S. E., and Zhang, H. (2000). REUNITE: A Recursive Unicast Approach to Multicast. In *Proceedings of IEEE INFOCOM 2000*, Tel-Aviv, Israel.

Tennenhouse, D. L. et al. (1997). A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86.

Tschudin, C. (1997). Open resource allocation for mobile code. In *Proceedings of the Mobile Agent'97 Workshop*, Berlin, Germany.

Tschudin, C. F. (1999a). A Self-Deploying Election Service for Active Networks. In *Proc. 3rd International Conference on Coordination Models and Languages (COORDINATION'99)*, Springer LNCS 1594, pages 183–195, Amsterdam, The Netherlands.

Tschudin, C. F. (1999b). Apoptosis - The Programmed Death of Distributed Services. In Vitek, J. and Jensen, C., editors, *Secure Internet Programming - Security Issues for Mobile and Distributed Objects*, Springer LNCS 1603, pages 253–260.

Wen, S., Griffioen, J., and Calvert, K. L. (2001). Building Multicast Services from Unicast Forwarding and Ephemeral State. In *Proceedings of IEEE OPENARCH 2001*, Anchorage, Alaska, USA.

Wetherall, D. J., Guttag, J. V., and Tennenhouse, D. L. (1998). ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *Proceedings of IEEE OPENARCH'98*, San Francisco, CA, USA.

Wittmann, R., Krasnodembski, K., and Zitterbart, M. (1998). Heterogeneous Multicasting based on RSVP and QoS Filters. In *SYBEN'98*, Zürich, Switzerland.

Yamamoto, L. and Leduc, G. (2000a). An Active Layered Multicast Adaptation Protocol. In *Proceedings of IWAN 2000*, Springer LNCS 1942, pages 180–194, Tokyo, Japan.

Yamamoto, L. and Leduc, G. (2000b). An Agent-Inspired Active Network Resource Trading Model Applied to Congestion Control. In *Proceedings of the MATA 2000 Workshop*, Springer LNCS 1931, pages 151–169, Paris, France.

Yamamoto, L. and Leduc, G. (2001a). Autonomous Multicast Reflectors over Active Networks. In *AISB'01 Symposium on Software Mobility and Adaptive Behaviour*, pages 40–49, York, UK.

Yamamoto, L. and Leduc, G. (2001b). Autonomous Reflectors: Note on the Merge Operation. Technical report, University of Liège.

AISB Journal **1**(1), © 2001
http://www.aisb.org.uk

Market Diversity and Market Efficiency: The Approach Based on Genetic Programming

Chia-Hsuan Yeh* and Shu-Heng Chen†

* Department of Information Management, Yuan Ze University, Chungli, Taoyuan 320, Taiwan, *imcyeh@saturn.yzu.edu.tw*

† Department of Economics, National Chengchi University, Taipei 11623, Taiwan, *chchen@nccu.edu.tw*

Abstract

The relation between market diversity and market efficiency has been studied. Economic heterogeneity is a fundamental driving force and an essential property in the economic systems. People who have different perspectives, technologies, or endowments may benefit from their trading behaviour which constitutes economic activities. In this paper, economic simulation based on the growing field of *artificial stock markets* is employed to study this issue. Market size and different learning styles are used to discuss the influence of heterogeneity. Simulation results have demonstrated that more participants and individual learning cause higher degree of traders' diversity, which, in turn, enhances market efficiency.

1 Introduction

The relationship between *competitiveness* and market performance has been discussed for a long time. In a competitive economic environment, each firm or individual is unable to influence the market. It has been mentioned in economics courses that the competitive market is more efficient and has higher social welfare. Therefore, it is the desirable picture that economists intend to draw. The concept of competitiveness is related to market size, i.e., the number of market participants. The idea here is that the larger economy contributes to microeconomic heterogeneity, for example, behaviour and strategies, profitability and market shares, production technology and efficiency. People having different perspectives about the future implies that there exists room for the economic activity and they may benefit from their trading behaviour. In other words, the higher degree of heterogeneity may provide more opportunities for trading. It is also an important seed of innovation.

However, few papers have addressed the influence of the number of agents in the economic simulation literature. Usually, the number of agents in the simulated economies is determined arbitrarily or under the consideration of computational load. In (Arifovic et al. (1997)), they mentioned that the minimal number of strings (agents) for effective search is usually taken to be 30, according to the artificial intelligence literature. However, the point of view from optimization is quite different from the agent-based simulated economies (and/or the real world). (Aoki (1999)) pointed out that there exists different statistical properties when the number of agents (N) goes to infinity. (Egenter et al. (1999)) have shown that the behaviour of the simulated market becomes quite smooth or

periodic and thus predictable if $N \to \infty$. (Den Haan (2001)) also described the importance of the number of *different* agents. In the framework of asset-pricing models, several properties depend on the number of types of agents. For example, the dynamics of interest rate, investment behaviour, and agent's welfare. In this paper, we try to restudy its relation to the market efficiency. This issue is very important in the financial market because it is well known that the thin and thick markets may have different financial properties and phenomena.

Economic simulations have been widely used in the study of economics. In order to model the interaction between many *heterogeneous* agents, the techniques of evolutionary computation were employed, for example, artificial neural nets, genetic algorithms (GAs) and genetic programming (GP). Different representations for agents' behaviour will influence what they can learn, which may explain different scenarios. Genetic programming serves this purpose better than other techniques. Therefore, GP is applied to modeling the evolution of traders' behaviour in this paper.

However, there are two ways to implement GP, namely, single-population GP and multi-population GP. The former is a way to model social learning, whereas the latter focuses on individual learning (Chen and Yeh (2001)). In this paper, we also attempt to figure out how differently the relationship between market size and efficiency may emerge under these two different versions of GP. All these issues are studied within the context of *agent-based artificial stock markets*, which is probably the most powerful tool to study behavioural finance with.

The organization of this paper is as follows. In section 2, we present the framework of the agent-based artificial stock market. The implementation of social learning and individual learning is described in section 3. Experimental designs and simulation results are shown in sections 4 and 5 respectively. Section 5 concludes.

2 Model Description

The basic framework of the artificial stock market considered in this paper is the standard asset pricing model employed in (Grossman and Stiglitz (1980)). The dynamics of the market is determined by an interaction of many heterogeneous agents. Each of them, based on his forecast of the future, maximizes his expected utility.

2.1 Traders

For simplicity, we assume that all traders share the same *constant absolute risk aversion* (CARA) utility function:

$$U(W_{i,t}) = -exp(-\lambda W_{i,t}), \tag{1}$$

where $W_{i,t}$ is the wealth of trader i at period t, and λ is the degree of relative risk aversion. Traders can accumulate their wealth by making investments. There are two assets available for traders to invest. One is the riskless interest-bearing asset called *money*, and the other is the risky asset known as the *stock*. In other words, at each period, each trader has two ways to keep his wealth, i.e.,

$$W_{i,t} = M_{i,t} + P_t h_{i,t} \tag{2}$$

where $M_{i,t}$ and $h_{i,t}$ denote the money and shares of the stock held by trader i at period t respectively, and P_t is the price of the stock at period t. Given this portfolio ($M_{i,t}$,$h_{i,t}$), a

trader's total wealth $W_{i,t+1}$ is thus:

$$W_{i,t+1} = (1+r)M_{i,t} + h_{i,t}(P_{t+1} + D_{t+1}) \tag{3}$$

where D_t is per-share *cash dividends* paid by the companies issuing the stocks and r is the riskless interest rate. D_t can follow *a stochastic process* not known to traders. Given this wealth dynamics, the goal of each trader is to myopically maximize the one-period expected utility function:

$$E_{i,t}(U(W_{i,t+1})) = E(-exp(-\lambda W_{i,t+1}) \mid I_{i,t}) \tag{4}$$

subject to Equation (3), where $E_{i,t}(.)$ is trader i's conditional expectations of W_{t+1} given his information up to t (the information set $I_{i,t}$).

It is well known that under $CARA$ utility and Gaussian distribution for forecasts, trader i's desire demand, $h^*_{i,t+1}$ for holding shares of risky asset is linear in the expected *excess return*:

$$h^*_{i,t} = \frac{E_{i,t}(P_{t+1} + D_{t+1}) - (1+r)P_t}{\lambda \sigma^2_{i,t}}, \tag{5}$$

where $\sigma^2_{i,t}$ is the conditional variance of $(P_{t+1} + D_{t+1})$ given $I_{i,t}$.

The key point in the agent-based artificial stock market is the formation of $E_{i,t}(.)$. In this paper, the expectation is modeled by genetic programming. The detail is described in the next section.

2.2 Price Determination

Given $h^*_{i,t}$, the market mechanism is described as follows. Let $b_{i,t}$ be the number of shares trader i would like to submit a bid to buy at period t, and let $o_{i,t}$ be the number of shares trader i would like to offer to sell at period t. It is clear that:

$$b_{i,t} = \begin{cases} h^*_{i,t} - h_{i,t-1}, & h^*_{i,t} \geq h_{i,t-1}, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

and

$$o_{i,t} = \begin{cases} h_{i,t-1} - h^*_{i,t}, & h^*_{i,t} < h_{i,t-1}, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Furthermore, let

$$B_t = \sum_{i=1}^{N} b_{i,t}, \text{ and } O_t = \sum_{i=1}^{N} o_{i,t} \tag{8}$$

be the totals of the bids and offers for the stock at period t, where N is the number of traders. Following (Palmer et al. (1994)), we use the following simple rationing scheme:

$$h_{i,t} = \begin{cases} h_{i,t-1} + b_{i,t} - o_{i,t}, & \text{if } B_t = O_t, \\ h_{i,t-1} + \frac{O_t}{B_t} b_{i,t} - o_{i,t}, & \text{if } B_t > O_t, \\ h_{i,t-1} + b_{i,t} - \frac{B_t}{O_t} o_{i,t}, & \text{if } B_t < O_t. \end{cases} \tag{9}$$

All these cases can be subsumed into

$$h_{i,t} = h_{i,t-1} + \frac{V_t}{B_t} b_{i,t} - \frac{V_t}{O_t} o_{i,t} \tag{10}$$

where $V_t \equiv \min(B_t, O_t)$ is the volume of trade in the stock.

According to Palmer et al.'s *rationing scheme*, we can have a very simple price adjustment scheme, based solely on the *excess demand* $B_t - O_t$:

$$P_{t+1} = P_t(1 + \beta(B_t - O_t)) \tag{11}$$

where β is a function of the difference between B_t and O_t. β can be interpreted as the speed of adjustment of prices. The β function we consider is:

$$\beta(B_t - O_t) = \begin{cases} \texttt{tanh}(\beta_1(B_t - O_t)) & \text{if } B_t \geq O_t, \\ \texttt{tanh}(\beta_2(B_t - O_t)) & \text{if } B_t < O_t \end{cases} \tag{12}$$

where `tanh` is the *hyperbolic tangent function*:

$$\texttt{tanh}(x) \equiv \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{13}$$

The price adjustment process introduced above implicitly assumes that the total number of shares of the stock circulated in the market is fixed, i.e.,

$$H_t = \sum_i h_{i,t} = H. \tag{14}$$

In addition, we assume that dividends and interests are all paid by cash, therefore:

$$M_{t+1} = \sum_i M_{i,t+1} = M_t(1 + r) + H_t D_{t+1}. \tag{15}$$

2.3 Formation of Expectations

As to the formation of traders' expectations, $E_{i,t}(P_{t+1} + D_{t+1})$, we assume the following functional form for $E_{i,t}(.)$.[1]

$$E_{i,t}(P_{t+1} + D_{t+1}) = \begin{cases} (P_t + D_t)(1 + \theta_1 f_{i,t} * 10^{-4}), & \text{if } -10^4 \leq f_{i,t} \leq 10^4, \\ (P_t + D_t)(1 + \theta_1), & \text{if } f_{i,t} > 10^4, \\ (P_t + D_t)(1 - \theta_1), & \text{if } f_{i,t} < -10^4 \end{cases} \tag{16}$$

The population of $f_{i,t}$ (i=1,...,N) is formed by genetic programming. That means, the value of $f_{i,t}$ is decoded from its GP tree $gp_{i,t}$.

As to the subjective risk equation, we modified the equation originally used by (Arthur et al. (1997)):

$$\sigma^2_{i,t} = (1 - \theta_2)\sigma^2_{t-1|n_1} + \theta_2[(P_t + D_t - E_{i,t-1}(P_t + D_t))^2]. \tag{17}$$

where

$$\sigma^2_{t-1|n_1} = \frac{\sum_{j=0}^{n_1-1}(P_{t-j} - \overline{P}_{t|n_1})^2}{n_1 - 1} \tag{18}$$

and

$$\overline{P}_{t|n_1} = \frac{\sum_{j=0}^{n_1-1} P_{t-j}}{n_1} \tag{19}$$

[1]There are several alternatives to model traders' expectations. The interested reader is referred to (Chen et al. (2001)).

In other words, $\sigma^2_{t-1|n_1}$ is simply the *historical volatility* based on the past n_1 observations.

Given each trader's expectations, $E_{i,t}(P_{t+1} + D_{t+1})$, according to Equation (5) and his own subjective risk equation, we can obtain each trader's desire demand, $h^*_{i,t+1}$ shares of the stock, and then how many shares of stock each trader intends to bid or offer based on Equation (6) or (7).

3 The Framework of Social and Individual Learning

In this paper, genetic programming is employed to model the formation of traders' expectations. In agent-based computational economics, there are two different styles to implement GP, namely, single-population GP and multi-population GP. While, from the optimization point of view, the choice of the two styles (architectures) may have an effect on the search efficiency, it brings different implications for the agent-based economic modeling. In the framework of single-population GP (SGP), each agent is represented by a *tree* (forecasting model, computer program). The evolution of this population of agents is performed through natural selection and genetic operations (reproduction, crossover, and mutation) on all of these trees (agents). In this case, agents can learn from other agents' experiences, and hence we call this *social learning*. On the other hand, in the framework of multi-population GP (MGP), each agent is endowed with a population of trees. Evolution is respectively manipulated in terms of the population of trees for each agent. In this case, agents learn only from their own experiences and reasoning, and hence we call this *individual learning*.

According to (Vriend (2000)) and (Vriend (2001)), there is an essential difference between individual and social learning, and the underlying cause for this is the so-called *spite effect*. The spite effect may occur in a social learning GA, but can never occur in an individual learning GA. To see how the spite effect can influence the outcome of the evolutionary process, Vriend uses the two different GAs to simulate the learning process of an oligopoly game. The simulation results show that while the individual learning GA moves close to the Cournot-Nash output level, the social learning GA converges to the competitive Walrasian output level.

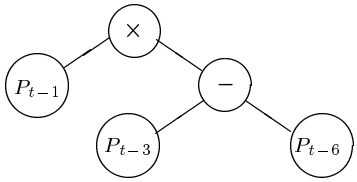
Vriend's finding pushes us to think harder of the choice of the two styles of learning. But, SGP (social learning) has another issue which is generally overlooked. (Harrald (1998)) criticized the traditional implementation of social learning. He mentioned the traditional distinction between the *phenotype* and *genotype* in biology and doubted whether the adaptation can be directly operated on the genotype via the phenotype in social processes. In particular, it is not easy to justify why we can learn or imitate other agents' strategies (genotype) by means of their actions (phenotype).

Motivated by this criticism, (Chen and Yeh (2001)) proposed a modified version of social learning. The idea is to include a mechanism, called the "*business school*". The business school serves as a faculty of researchers or a library. Knowledge in the business school or library is *open for everyone*. Traders can consult the researchers when they are under great peer pressure or economic loss. More precisely, the business school can be viewed as a collection of forecasting models. SGP (social learning) can then be applied to model the evolution of the business school without inviting Harrald's criticism. [2]

Each researcher (forecasting model) is represented by a GP parse tree. Initially, the forecasting models ($GP_{t=1}$) are randomly generated based on the function set and termi-

[2] For more details, please see (Chen and Yeh (2001)).

nal set given in Table 1. For example,



Each tree in the initial population of GP trees (forecasting models) is randomly generated by either the growth method or the full method based on a toss of fair coin.[3] In the initialization stage, the maximum depth of a tree is restricted to 6. The performance of each forecasting model is determined by the pre-assigned fitness function, i.e., *mean absolute percentage error* (MAPE).

In the business school, the forecasting models will be evaluated with a pre-specified schedule, say once for every m_1 trading days. The evaluation procedure proceeds as follows. At the evaluation date t, each old model (model at period $t-1$, GP_{t-1}) shall compete with a *new model* which is generated from GP_{t-1} itself by using one of the following four genetic operators: reproduction, mutation, crossover, and immigration, each with a probability p_r, p_m, p_c, and p_I (Table 1). The winner, the one with lower MAPE, will be selected as a model of the next generation, GP_t. When the evaluation is done over all the old models, a new generation of forecasting models (GP_t) is then born. The four genetic operators are detailed as below.

- Reproduction:
  Two forecasting models are randomly selected from GP_{t-1}. The one with lower MAPE over the last m_2 days' forecasts is chosen as the *new model*.

- Mutation:
  Two forecasting models are randomly selected from GP_{t-1}. The one with lower MAPE over the last m_2 days is chosen as a candidate, and has a probability of p_M (Table 1) being mutated. There are several ways to implement mutation. What is implemented here is known as *tree mutation*. If a tree is chosen to be mutated, one of its nodes is randomly selected, and the subtree originating from the node is replaced by a new subtree which is randomly generated. For example, consider the forecasting model *A*, whose tree structure is shown in figure 3. Suppose that the point M is chosen as the mutation point. Then the subtree originated from point M is removed and is replaced by a subtree *B*, which is randomly generated. And the result is the *new model C*.[4]

- Crossover:
  Two pairs of forecasting models are randomly chosen, ($gp_{j_1,t-1}$, $gp_{j_2,t-1}$) and ($gp_{k_1,t-1}$, $gp_{k_2,t-1}$). The one with lower MAPE in each pair is chosen as a *parent*. The crossover operation is a sexual operation that starts with two parental models. One of the nodes for each parent is selected randomly, and the subtrees originating from the nodes are exchanged. Two offspring are then produced. One of them is randomly chosen as the *new model*. For example, consider Tree *A* and *B* shown above as the parents, and points M and N are the crossover points. After crossover by exchanging the subtrees originating from M and N, the two offspring are Tree A' and Tree B' as shown in figure 3.

[3] For more details, please refer to (Koza (1992)).

[4] But, if Tree *A* is not mutated (with a probability $1 - P_M$), then *A* will automatically be the *new model*.

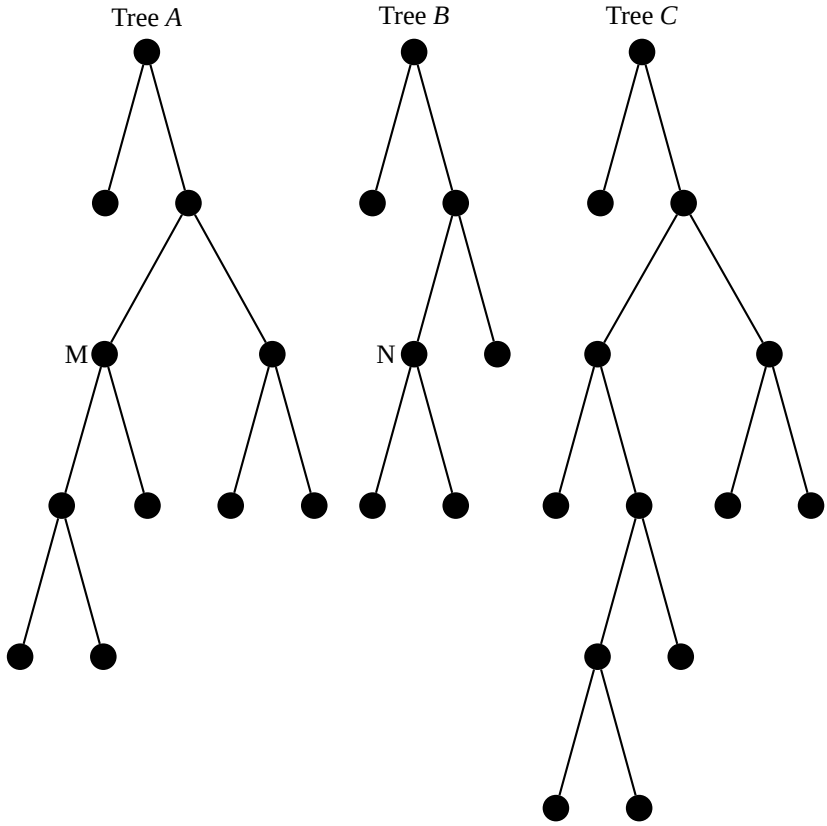


Figure 1: Tree structure for model A

- Immigration:
  A forecasting model is randomly created as the *new model*. This operator is used to approximate the concept of *imagination*.

As to the evolution of traders' behaviour, each trader is endowed with N_I forecasting models. These forecasting models are used to predict $E_{i,t}(U(W_{i,t+1}))$. In the beginning, they are also randomly generated. The performance of these forecasting models is measured by *profits*, not MAPE, under a *validation* process. Traders' adaptation proceeds as follows. At the evaluation date t, each trader has to make a decision. Should he change his forecasting model used in the previous period? This decision is affected by two psychological factors, namely, *peer pressure* and *a sense of self-realization*. Generally speaking, if a trader's profits earned in the last period is inferior to many other traders, or his profits come to a historical low, then he will have a higher motivation, and hence a higher probability, to change.

The probabilities that traders consider to change a model can be mathematically described as follows. First, suppose that traders are ranked by *the net change of wealth* over the last n_2 trading days. Let $\Delta W_{i,t}^{n_2}$ be this net change of wealth of trader i at time period t, i.e.,

$$\Delta W_{i,t}^{n_2} \equiv W_{i,t} - W_{i,t-n_2}, \tag{20}$$

and, let $R_{i,t}$ be her rank. Then, the probability that trader i will change a model at the end

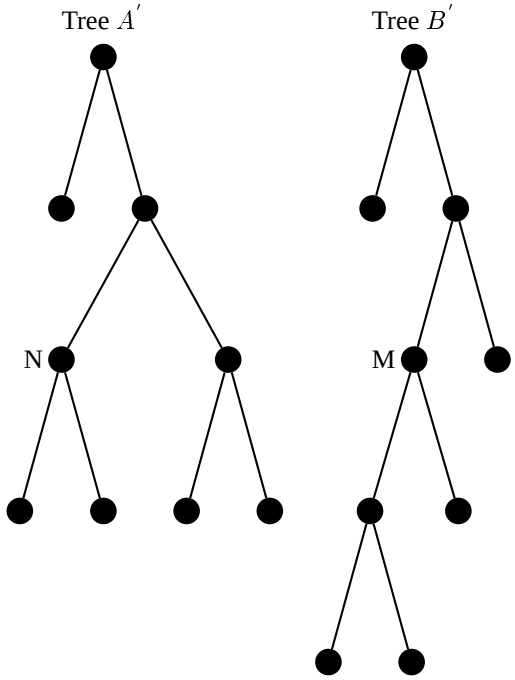


Figure 2: Tree structure after crossover

of period t is assumed to be determined by:

$$p_{i,t} = \frac{R_{i,t}}{N}. \tag{21}$$

The choice of the function $p_{i,t}$ is quite intuitive. It simply means that:

$$p_{i,t} < p_{j,t}, \text{ if } R_{i,t} < R_{j,t}. \tag{22}$$

In other words, the traders who come out top shall suffer less peer pressure, and hence have less motivation to change models than those who are ranked at the bottom.

In addition to peer pressure, a trader may also decide to change a model out of a sense of *self-realization*. Let the growth rate of wealth over the last n_2 days be:

$$\delta_{i,t}^{n_2} = \frac{W_{i,t} - W_{i,t-n_2}}{\mid W_{i,t-n_2} \mid}, \tag{23}$$

and let $q_{i,t}$ be the probability that trader i will change a model at the end of the tth trading day, then it is assumed that:

$$q_{i,t} = \frac{1}{1 + exp^{\delta_{i,t}^{n_2}}}. \tag{24}$$

The choice of this density function is also straightforward. Notice that

$$\lim_{\delta_{i,t}^{n_2} \to \infty} q_{i,t} = 0, \tag{25}$$

Table 1: Parameters of the Stock Market

The Stock Market	
Shares of the stock (H)	100 (500)
Initial money supply (M_1)	100 (500)
Interest rate (r)	0.1
Stochastic process (D_t)	Uniform distribution, U(5.01,14.99)
Price adjustment function	`tanh`
Price adjustment (β_1)	0.2×10^{-4}
Price adjustment (β_2)	0.2×10^{-4}
Parameters of Genetic Programming	
Function set	$\{+, -, \times, \%$, `sin,cos,exp,Rlog, Abs,sqrt`$\}$
Terminal set	$\{P_t, P_{t-1}, \cdots, P_{t-10}, P_t + D_t, \cdots, P_{t-10} + D_{t-10}\}$
Selection scheme	Tournament selection
Tournament size	2
Probability of creating a tree by reproduction (p_r)	0.10
Probability of creating a tree by immigration (p_I)	0.20
Probability of creating a tree by crossover (p_c)	0.35
Probability of creating a tree by mutation (p_m)	0.35
Probability of mutation (P_M)	0.30
Probability of leaf selection under crossover	0.5
Mutation scheme	Tree mutation
Replacement scheme	(1+1) Strategy
Maximum depth of tree	17
Maximum number in the domain of Exp	1700
Number of generations	4000
Business School	
Number of faculty members (F)	500
Criterion of fitness (Faculty members)	MAPE
Evaluation cycle (m_1)	20
Sample size (MAPE) (m_2)	10
Search intensity in Business School (I_s^*)	5
Traders	
Number of traders (N)	100
Number of ideas for each trader (N_I)	1 (A), 10 (B), 25 (C)
Degree of RRA (λ)	0.5
Criterion of fitness (Traders)	Increments in wealth (Income)
Evaluation cycle (n_2)	1
Search intensity by trader himself (I_h^*)	5
θ_1	0.5
θ_2	0.0133

and

$$\lim_{\delta_{i,t}^{n}{}^{2} \to -\infty} q_{i,t} = 1. \tag{26}$$

Therefore, the traders who have made great progress will naturally be more confident and hence have little need for changing ideas, whereas those who suffer devastating regression will have a strong desire for changing ideas.

In sum, for trader i, the decision to change a model can be considered as a result of *a two-stage independent Bernoulli experiment*. The success probability of the first experiment is $p_{i,t}$. If the outcome of the first experiment is success, the trader will change a model. If, however, the outcome of the first experiment is failure, the trader will continue to carry out the second experiment with the success probability $q_{i,t}$. If the outcome of the second experiment is success, then the trader will also change a model. Otherwise, the trader will keep her model unchanged. Based on the description above, the probability of changing the idea ($r_{i,t}$) for trader i at period t is:

$$r_{i,t} = p_{i,t} + (1 - p_{i,t})q_{i,t} = \frac{R_{i,t}}{N} + \frac{N - R_{i,t}}{N} \frac{1}{1 + exp^{\delta_{i,t}^{k}}} \tag{27}$$

Once the trader decides to change, how he changes depends on the two styles of learning. In the case of social learning, he will register to the business school, and randomly catch a model there. In the case of individual learning, he will search a new model on his own by running genetic programming on the population of his N_I existing models (think tank).

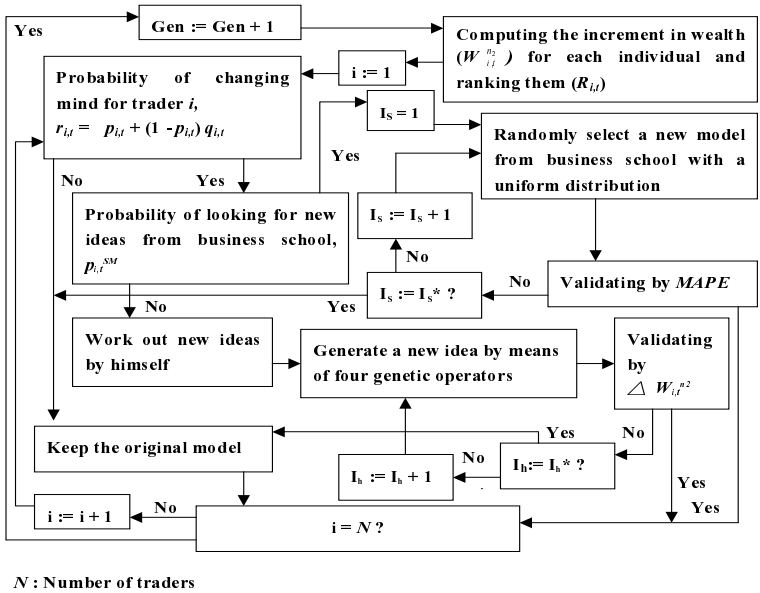
Once he gets a new model, he will test the new model with the historical data and see whether the new model will bring him more profits than the old one. If it does, the old model will be replaced by the new model. Otherwise, he will keep on searching in the business school (in the case of social learning) or his think tank (in the case of individual learning) until either he succeeds or he fails for a pre-specified time, I_s^* or I_h^* (Table 1). The process of traders' evolution is also shown in Flowchart 1.

4 Experimental Designs

In order to examine the effect of market size on market efficiency, we consider experiments associated with two different *market sizes* (numbers of participants). The small market, M_1, has 100 traders, whereas the large market, M_2 has 500 traders. Furthermore, to see the effect of social and individual learning on market efficiency, three scenarios corresponding to different styles of learning are proposed. Traders in Market A follow the social learning scheme (driven by SGP), and traders in Markets B and C follow the individual learning scheme (driven by MGP).

Markets B and C are distinguished by the number of models (N_I) (the size of think tank) assigned to each trader. In SGP, *market size* (number of traders) and *population size* (number of GP trees or models) refer to the same thing, because each trader is represented by one model in SGP. However, in MGP, market size and population size refer to two different things. Therefore, it is important to distinguish these two different size effects on market efficiency. In this paper, traders in Market B are endowed with 10 models, whereas traders in Market C are equipped with 25 models.

Therefore, in total, we conduct six distinct experiments, namely, (M_1, A), (M_1, B), (M_1, C), (M_2, A), (M_2, B), (M_2, C). Each experiment consists of 10 trials of 4000 trading periods. In terms of genetic programming, the number of *generations* is not necessarily the same as the number of *trading periods*. In fact, the former is the latter divided



N : Number of traders

Flowchart 1: Traders' Search Process

by evaluation cycle. In the case of Markets B and C, since GP is applied to traders, and the evaluation cycle (n_2) is 1 (Table 1), the number of generations is 4000, i.e., the same as the number of trading periods. Nevertheless, in Market A, GP is not applied to traders but to the business school, and the evaluation cycle (m_1) for the business school is 20 (Table 1); therefore the number of generations is only 200.

Each simulation will generate a time series data with 4000 observations of price. To avoid the effect of *initialization*, we shall drop the first 1000 observations, and base our analysis only on the last 3000 observations.

5 Simulation Results

5.1 The Efficiency Measure

The purpose of our experiments is to examine the effect of market size and learning styles on market efficiency. Before proceeding further, we have to give *market efficiency* a technical notion so that our our numerical results can be presented and analyzed accordingly. In financial econometrics, the market is said to be efficient if the return series $\{r_t\}$ is *unpredictable*, where:

$$r_t = \ln(P_t) - \ln(P_{t-1}). \tag{28}$$

Technically speaking, a series is unpredictable if there exists no linear and nonlinear structures in it, or the series is *independent*. To test whether a series is independent, we followed the procedure of (Chen et al. (2001)). This procedure is composed of two steps, namely, *the PSC filtering* and *the BDS testing*. We first applied the Rissanen predictive stochastic complexity (PSC) to filter the linear process. The PSC criterion is frequently used as a model selection tool in time series analysis. By using the criterion, we can

determine the *best* linear structure, i.e., the best ARMA (*AutoRegressive and Moving Average*), of the time series in question.[5] Once the linear signals are filtered, any signals left must be *nonlinear*. The next step is then to apply the BDS test (Brock et al. (1996)), one of the most frequently used tests for nonlinearity, to the residual series, i.e., the series after PSC filtering. The null hypothesis of the BDS test is that the series in question is identically and independently distributed. If we fail to reject the null hypothesis, then the series is said to be independent. Since the BDS test statistic under a large sample follows a standardized normal distribution, it is quite easy to have an eyeball check on the results.

5.2 Results: Price and Market Efficiency

The two-stage econometric procedure as outlined above gives three statistics for each market experiment, namely, PSC, R^2, and BDS. They are all shown in Table 2. The (p, q) under the column "PSC" is the orders p and q selected based on the Rissanen PSC criterion. The column "R^2" reports the coefficient of determination derived by running the PSC-selected $ARMA(p, q)$ regression. The column "BDS" gives the BDS test statistic. If the return series is independent, there shall be completely no structure, be it linear or nonlinear, found in the series. In our statistical language, p, q, and R^2 should all be zero; the BDS test should be as low as possible, but not greater than a critical value, say, 1.96.[6] Using these reference numbers, we can get a quick grasp of our simulation results.

First of all, we would like to draw readers' attention to the experiment (M_2, C). As detailed in the previous section, the market in the experiment has three characteristics, large number of traders, large population size, and individual learning. Based on the reference numbers given above, one can easily see that the return series generated by this experiment are all independent. The only exception is Run 2, which has a linear structure being detected ($p = 1$). But the associated R^2 is so small (0.009) that one can hardly be interested in this linear signal. So, one can generally conclude that markets with large number of participants, each learning with the evolution of his own large number of forecasting models, is highly efficient in the sense that the efficient market hypothesis is sustained (return series is unpredictable).

From (M_2, C), we have three directions to move in. We would first like to know what will happen when *the number of forecasting models* assigned to each trader is reduced. To answer this question, we move from (M_2, C) to (M_2, B). In contrast, markets under (M_2, B) have stronger linear structures: the $AR(1)$ signal is detected in seven out of the ten markets, while their associated R^2s are generally very low. Furthermore, in one case (Run 6), nonlinear structure is also found (BDS =2.107). All this evidence suggests that

[5]The ARMA process is a canonical linear time series model. Given a time series $\{X_t\}$, we say that $\{X_t\}$ follows a AR(p) process if

$$X_t = a_0 + \Sigma_{i=1}^{p} a_i X_{t-i} + \varepsilon_t, \tag{29}$$

and $\{X_t\}$ follows a MA(q) process if

$$X_t = b_0 + \Sigma_{j=1}^{q} b_j \varepsilon_{t-j} + \varepsilon_t, \tag{30}$$

where ε_t is a Gaussian white noise, i.e.,

$$\varepsilon_t \sim N(0, \sigma^2), \quad \text{and} \quad E(\varepsilon_s, \varepsilon_t) = 0 \quad \text{if} \quad s \neq t. \tag{31}$$

$\{X_t\}$ is said to follow a ARMA(p,q) process, if

$$X_t = c_0 + \Sigma_{i=1}^{p} a_i X_{t-i} + \Sigma_{j=1}^{q} b_j \varepsilon_{t-j} + \varepsilon_t \tag{32}$$

[6]The exact number depends on the chosen significance level of the test.

Table 2: PSC Filtering and BDS Test

Market Size: M_1									
	Market A			Market B			Market C		
Run	PSC	R^2	BDS	PSC	R^2	BDS	PSC	R^2	BDS
1	(2,2)	0.104	7.860	(2,3)	0.057	5.479	(2,3)	0.045	6.349
2	(3,3)	0.089	7.749	(0,2)	0.049	7.104	(3,0)	0.046	6.561
3	(2,2)	0.072	6.845	(2,2)	0.047	6.369	(0,0)	None	4.948
4	(1,2)	0.086	8.462	(2,2)	0.066	6.988	(2,2)	0.085	8.153
5	(3,3)	0.082	7.750	(0,2)	0.029	6.802	(4,5)	0.086	6.274
6	(2,2)	0.072	6.909	(0,2)	0.033	7.052	(2,2)	0.056	6.099
7	(2,3)	0.102	6.958	(1,2)	0.091	10.831	(0,2)	0.047	7.726
8	(1,0)	0.058	6.890	(1,0)	0.051	7.012	(3,4)	0.085	6.413
9	(4,5)	0.133	7.431	(0,2)	0.061	7.777	(1,2)	0.044	6.073
10	(0,3)	0.058	6.956	(2,2)	0.057	5.727	(0,2)	0.031	5.833
Average		0.086	7.381		0.054	7.114		0.058	6.443
Market Size: M_2									
	Market A			Market B			Market C		
Run	PSC	R^2	BDS	PSC	R^2	BDS	PSC	R^2	BDS
1	(1,0)	0.012	3.947	(1,0)	0.006	0.385	(0,0)	None	-0.009
2	(1,0)	0.011	3.096	(1,0)	0.007	1.479	(1,0)	0.009	0.788
3	(1,0)	0.018	2.971	(0,0)	None	0.801	(0,0)	None	0.455
4	(1,0)	0.015	2.076	(1,0)	0.007	0.373	(0,0)	None	-0.163
5	(1,0)	0.009	3.452	(0,0)	None	0.319	(0,0)	None	0.591
6	(1,0)	0.011	3.221	(1,0)	0.005	2.107	(0,0)	None	0.180
7	(1,0)	0.029	2.635	(0,0)	None	0.003	(0,0)	None	0.678
8	(2,3)	0.027	3.529	(1,0)	0.009	1.292	(0,0)	None	1.565
9	(1,0)	0.011	3.866	(1,0)	0.010	0.945	(0,0)	None	1.154
10	(0,2)	0.025	3.944	(1,0)	0.013	0.034	(0,0)	None	1.318
Average		0.0168	3.274		0.008	0.774		0.009	0.656

The BDS test statistic is asymptotically normal with mean 0 and standard deviation 1. The significance level of the test is set at 0.95. In BDS test, the distance parameter (standard deviations) is set to be 1, and the embedding dimension is set to be 5.

the number of forecasting models can have a positive effect on the market efficiency. The larger the number, the higher the efficiency.

The second direction to move in is to consider the effect of *the number of market participants* (*market size*), i.e., to move from (M_2, C) to (M_1, C). From PSC, we can see that the linear structure is even stronger than the case (M_2, C). Linear signals are pervasively found in all markets, with Run 3 as the only exception. Some of the linear signals are even highly structured, say, Run 5. Looking at BDS, none of these markets fail to reject the null hypothesis. As a result, all markets are not efficient. The striking fact is that in experiment (M_2, C), we have all markets efficient, but then all inefficient in (M_1, C). The only change is the number of participants. Therefore, *the market size can have a dramatic impact on the market efficiency*. The more the participants, the higher the efficiency.

The third direction to move in is to examine the effect of *social learning*. Here, we move from (M_2, C) to (M_2, A). From all the three statistics, one can easily see that all markets under (M_2, A) are relatively inefficient. Hence, the social learning scheme represented by SGP with the business school can have a negative impact upon market efficiency.

5.3 Results: Trading Volume and Market Diversity

In addition to price dynamics, another important observation in stock markets is trading volume. The trading volume serves as an important indicator of *the diversity of traders' expectations*. Consider an extreme case where all traders hold the same expectation (forecast). With the same expectation, traders always stand in the same position, either all to buy or all to sell at any given price. In this case, no trade can possibly happen in the market, and this corresponds to the famous *no-trade theorem* in neo-classical economics (Tirole (1982)). On the other hand, if traders' expectations are diversified (heterogeneous), then optimistic traders can easily be matched to pessimistic traders at many different prices, and one can expect a large trading volume appearing in the market.

To have a slice of the idea that the trading volume is very different among the experiments, Graphs 1-3 plot the time series of the trading volume observed in a typical run of the three market experiments (M_1, A), (M_1, B), and (M_1, C). By presenting these figures together, one can immediately see that the market (M_1, A) is rather quiet as opposed to the other two. Its trading volume is almost nil during many of the trading days. Therefore, except for a few cases, traders in market (M_1, A) share very similar forecasts.

Of course, three individual cases may tell us little about what causes the difference in the diversity of traders' expectations. To be more systematic, Table 3 gives us two summary statistics of all experiments. One is the average daily trading volume (the "Mean" column), and the other is the volatility of the daily trading volume (the "Std. Dev." column). Based on this table, one can articulate the main factors which contribute to the diversity of traders' expectations, and there are two factors, *learning styles* and *market size*.

First, *learning styles*. To see how learning styles can change the diversity of traders' expectations, let us compare the experiment of social learning ((M_1, A)) to the experiment of individual learning ((M_1, B), (M_1, C)). The ten markets in the experiment (M_1, A) are all very quiet. On average, the mean trading volume over these ten markets is only 0.15 units, whereas the same figure in the other two experiments are 15 (M_1, B) and 14 (M_1, C) units respectively, i.e. almost 100 times higher. This sharp contrast also appears in the comparison between the experiment (M_2, A) and the experiments (M_2, B) and (M_2, C). For the former, the mean trading volume is only 1 unit, but for the latter, the figure jumps to 81 and 79.

These figures clearly show that social learning and individual learning can have a dramatic impact upon the diversity on traders' expectations. This result may not come to us as a surprise. If traders learn and adapt via *a pool of common knowledge* (business school, library,...), then the diversity of them is constrained by the diversity of the pool. Since direct imitation (reproduction) in the business school is feasible, competitive forecasting models can be disseminated to a large number of faculty members, which reduces the diversity of the pool, and the diversity of traders' expectations. On the other hand, traders who learn and adapt on their own would not allow other traders to imitate their best forecasting models which are kept as a *business secret*. Therefore, dissemination of knowledge is more difficult to proceed in the context of individual learning, and hence it tends to maintain a greater diversity of traders' expectation.

Second, *market size*. To see the effect of market size on market diversity, the experiment (M_1, A) is compared to the experiment (M_2, A), i.e., the one with one hundred traders to the one with five hundred traders. From Table 3, one can see that when the number of traders increases by a factor of 5, the mean daily trading volume also increases by about 9 to 10 times, from 0.15 units to 1 units. But, caution should be exercised on interpreting these numbers. The thing is that when market size increases by 5 times, the

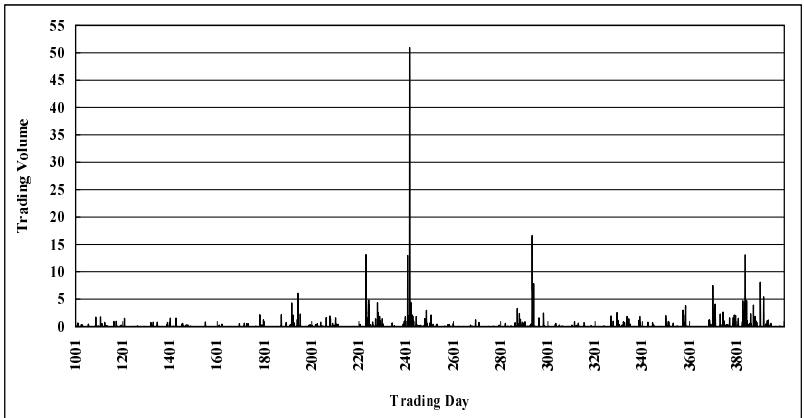


Figure 1 : Time Series Plot of the Trading Volume (Market A, First Run)

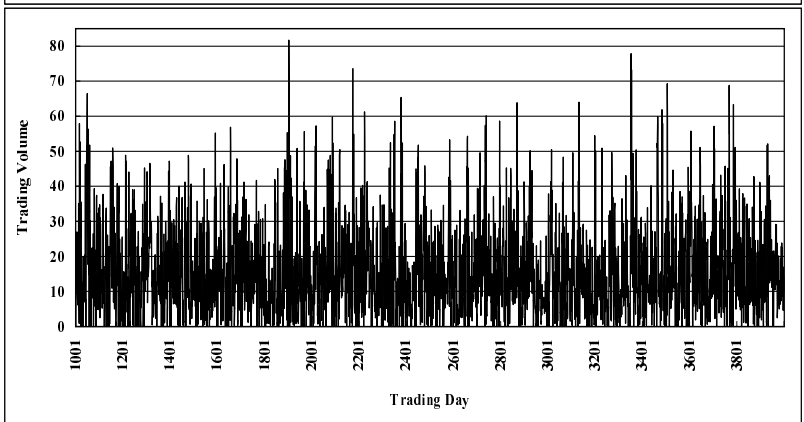


Figure 2 : Time Series Plot of the Trading Volume (Market B, First Run)

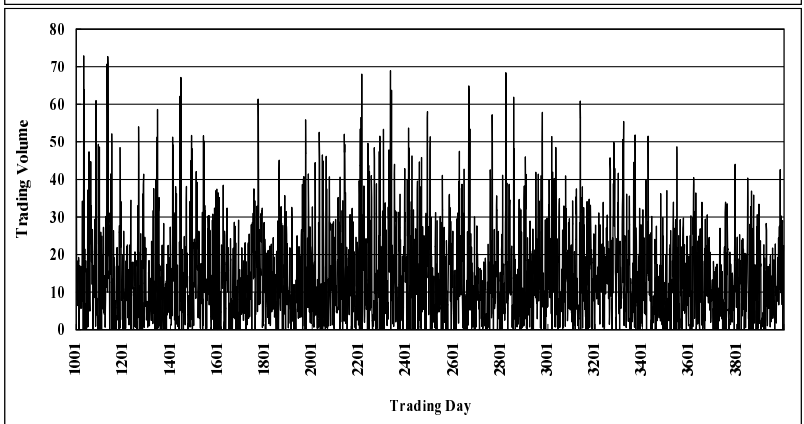


Figure 3 : Time Series Plot of the Trading Volume (Market C, Sixth Run)

total number of shares also increases by that amount (so that share per capita remains unchanged). Therefore, it would not surprise us if trading volume *only* increases in proportion to the increase in market size. However, here we see that the trading volume increases by more than 5 times, in fact to double that figure; therefore, the effect of market size on market diversity is not superficial. One can confirm this finding by the other two sets of experiments under individual learning, i.e., $\{(M_1, B), (M_2, B)\}$, and $\{(M_1, C), (M_2, C)\}$. By comparing the same figure pairwisely, one can see once again that when market size increases by five times, the trading volume increases by more than five times.

There is a simple explanation for the impact of market size upon market diversity. In the case of social learning, even though there is only a single pool of common knowledge, a larger number of traders (students) implies a larger sampling of the pool, and hence a greater diversity. In the case of individual learning, since there is no channel for direct dissemination of knowledge, a larger number of traders implies more secrets kept by individuals, and hence also a greater diversity.

But, this explanation is only one-way and may oversimplify the situation. The markets which we study are typical examples of *co-evolving* systems. What has not been mentioned in the explanation above is the *interaction* among different components of the market. For example, in the case of social learning, a greater diversity of traders' expectations may result in more complex aggregate (price) phenomena, which in turn also nurse the diversity of the business school. Furthermore, a pool with a greater diversity may make traders' expectations even more diversified. This *reinforcing mechanism* can go on and on, which eventually increases the trading volume by far more than 5 times.

As we have discussed earlier, in the context of individual learning, there is another key parameter which has an effect on market efficiency, i.e., *population size*. However, it is interesting to note that population size does not have a positive effect on trading volume. If we compare the experiment (M_1, B) to the experiment (M_1, C), we see the mean daily trading volume decreases from 15 units to 14 units when population size increases from 10 to 25. A similar result is also observed in the other pair of experiments, $\{(M_2, B), (M_2, C)\}$.

Why does population size have a *negative effect* on the diversity of traders' expectations? The reason is also intuitive. In the case of social learning, each trader can base their final decision only on *one model*, no matter how many models, be it 10 or 25, they are able to process. As a result, the diversity of traders' expectations is not directly affected by population size. Nonetheless, a larger population size equips each trader with a better capability to process information. Hence, if there is a best model at a point in time, the chance of discovering this model improves for all traders. Even though they do not come up with the same model, it is still likely that they find similar ones. That explains why the diversity of traders' expectations and the trading volume may actually drop in the presence of a larger population size.

5.4 Results: Portfolio and Market Diversity

In the previous section, market diversity was studied from the perspective of the trading volume. There is another way to observe market diversity, i.e, *trader's portfolios*. A portfolio is the distribution of traders' wealth into *money* and *shares of the stock*. If at any point in time, traders' are homogeneous in their portfolio decision, then they must hold the same shares of stock, and the variance (diversity) of shares held among all traders is *zero*. Therefore, *the variance of shares held* provides us another measure with which to examine market diversity.

Corresponding to graphs 1-3, graphs 4-6 present the time series plots of the variance

Table 3: Trading Volume

Market Size: M_1						
	Market A		Market B		Market C	
Run	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
1	0.150	1.384	16.216	13.034	15.189	12.805
2	0.277	1.328	17.665	14.517	15.363	13.067
3	0.410	1.998	15.373	13.785	15.613	13.035
4	0.150	0.928	16.713	13.898	15.843	13.323
5	0.049	0.285	15.122	13.405	13.702	12.439
6	0.080	0.480	14.342	12.345	14.613	12.380
7	0.044	0.328	13.955	11.980	12.892	12.168
8	0.172	1.242	15.174	13.116	12.841	11.371
9	0.059	0.329	14.365	12.535	14.820	12.369
10	0.105	1.134	14.153	12.447	14.838	12.503
Average	0.150	0.944	15.308	13.106	14.571	12.546
Market Size: M_2						
	Market A		Market B		Market C	
Run	Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
1	0.728	5.172	73.330	47.525	71.622	49.105
2	0.935	4.042	82.657	53.155	82.150	53.602
3	0.718	7.004	84.541	50.787	82.813	51.563
4	0.778	5.195	72.865	50.061	76.502	49.392
5	1.112	8.127	95.009	60.465	86.131	57.656
6	1.319	6.277	85.942	52.902	79.125	52.582
7	0.409	1.807	74.807	48.631	90.215	55.193
8	0.877	7.791	78.962	50.327	70.133	45.129
9	1.944	10.873	81.123	53.567	78.114	49.935
10	1.453	7.110	83.132	51.648	78.295	52.167
Average	1.0273	6.340	81.237	51.907	79.510	51.632

of shares held in the three different market experiments. A large degree of homogeneity of portfolios is observed in the experiment (M_1, A). In many of the trading days, the variance is almost zero, which suggests that traders hold the same amount of shares. But (M_1, B) and (M_1, C) give quite different pictures.

Table 4 gives the mean variance of all experiments. We shall not give detailed accounts of the results here, because the findings are the same as those in Table 3. Briefly speaking, a greater diversity of portfolios is observed under a larger market size and individual learning. However, a larger population size slightly reduced the diversity.

5.5 Market Diversity and Market Efficiency

The experimental results obtained above can be summarized as two effects on market efficiency (price predictability), namely, the *size effect* and the *learning effect*. The size effect says that the market will become efficient when the number of traders (market size) and/or the number of models (GP trees) processed by each trader (population size) increases. The learning effect says that the price will become more efficient if traders'

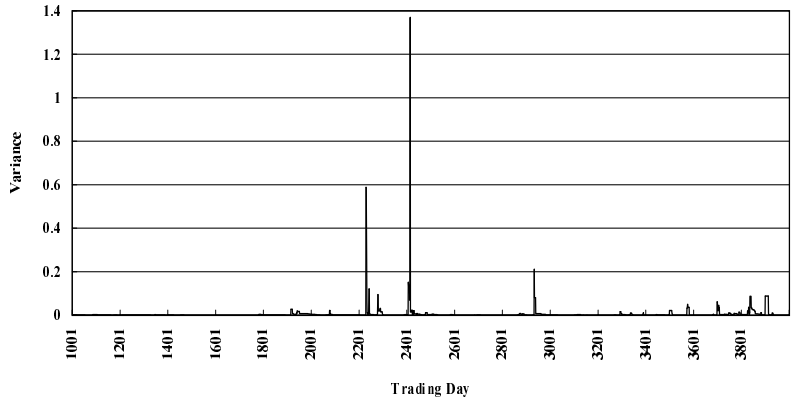


Figure 4 : The Variance of Stock Shares between Traders (Market A, First Run)

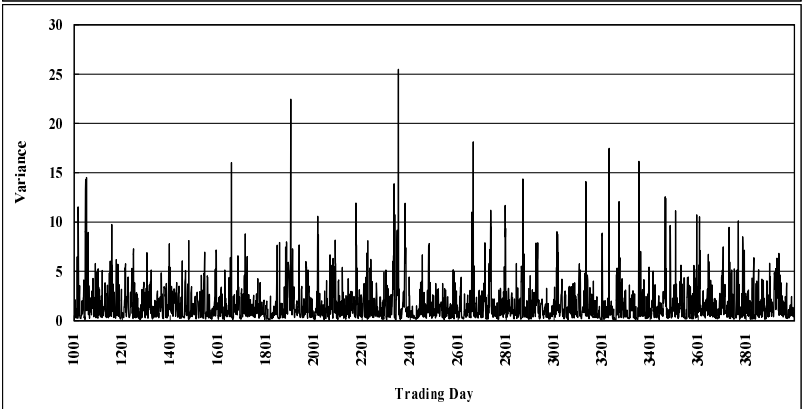


Figure 5 : The Variance of Stock Shares between Traders (Market B, First Run)

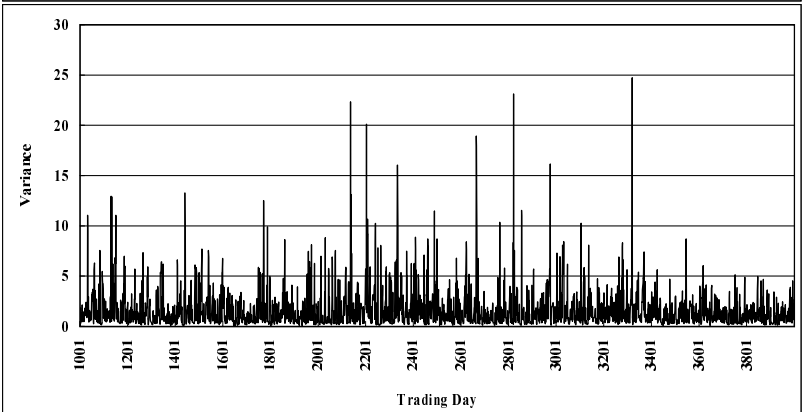


Figure 6 : The Variance of Stock Shares between Traders (Market C, Sixth Run)

Table 4: The Mean Variance of Stock Shares between Traders

Run	Market Size: M_1			Market Size: M_2		
	Market A	Market B	Market C	Market A	Market B	Market C
1	0.004	1.743	1.652	0.003	1.733	1.722
2	0.008	2.034	1.757	0.004	1.961	1.937
3	0.012	1.837	1.768	0.005	1.916	1.870
4	0.007	1.957	1.780	0.005	1.732	1.866
5	0.001	1.690	1.601	0.005	2.299	2.047
6	0.002	1.556	1.688	0.006	1.967	1.966
7	0.001	1.584	1.451	0.002	1.782	2.110
8	0.006	1.700	1.433	0.007	1.826	1.612
9	0.001	1.642	1.618	0.017	1.982	1.794
10	0.003	1.576	1.679	0.010	1.825	1.918
Average	0.005	1.732	1.643	0.006	1.902	1.884

adaptive behaviours become more independent and private. Coming to market diversity, we observe very similar effects except with population size: market diversity does not go up with population size.

These findings motivate us to search for a link between *market diversity* and *market efficiency*. A suggested argument is: *a larger market size, and a more isolated learning style will increase the diversity of traders' expectations, which in turn make the market become more active (high trading volumes), and hence more efficient (less predictable).* While the statement is somewhat plausible, we cannot give it a formal proof. In fact, in complex adaptive systems, the route from cause to effect are sometimes so complicated that no one can follow every step of it. Nevertheless, since the argument is empirically relevant, it can be taken as a hypothesis to test with real data.

Consider the Taiwan stock market as an example. A large proportion of market participants are individual investors who have little control of the market. Based on our "theorem" that a large market size implies efficiency, the Taiwan stock market should be efficient. However, empirical evidence has shown that this market is not that efficient (Chen and Tan (1996)). What is missing here? By our "theorem", the answer rests on *social learning*. Actually, in this market, individual traders usually consult the professionals from companies, institutions and even mass media., which is exactly a kind of social learning. Also, take the United States as another example. In the U.S. stock market, a large proportion of market participants are *institutional investors*, and each has their own research department. This is more similar to *individual learning*. By our "theorem", this market should be very efficient. Empirical studies have shown that it is indeed the case (Chen and Tan (1996)).

6 Concluding Remarks

In this paper, the relation between market diversity and market efficiency is investigated. The simulation results reveal that the important driving force which makes market efficiently is market diversity. The increase of market size contributes to the market efficiency

by means of introducing greater diversity into the market. Moreover, individual learning further reinforces the effect, whereas population size plays a mixing role. These three factors co-influence the market dynamics.

Of course, there are some other determinants for market efficiency. For example, the degree of traders' *prudence*. This behavioural parameter concerns the number of periods (time horizon) which the traders look back at while making their forecasts. A prudent trader cares about long-term profits, which, in our framework, corresponds to a long evaluation cycle (n_2). Also, in a more flexible design, traders should be able to adaptively switch between individual learning and social learning, rather than getting stuck with only one learning style. The effects of these determinants are left for further research.

Acknowledgements

Research support from NSC grants No. 89-2415-H-214-002 is gratefully acknowledged. This paper was revised from its original version by taking the advice and comments of four anonymous referees. The authors are very grateful for their painstaking review of this paper. Of course, all remaining errors are the authors' sole responsibility.

References

Aoki, M (1999). Aggregate Dynamics and Agent Interactions in Economic Model: A Stochastic View. Concluding lecture given at the *Fourth Workshop on Economics with Heterogeneous Interacting Agents* (WEHIA'99), Genoa, Italy.

Arifovic, J., Bullard, J. and Duffy, J. (1997). The Transition from Stagnation to Growth: An Adaptive Learning Approach. *Journal of Economic Growth*, Vol. 2, pp. 185-209.

Arthur, W. B., Holland, J. H., LeBaron, B., Palmer, R. and Taylor, P. (1997). Asset pricing Under Endogenous Expectations in an Artificial Stock Market. In *The Economy as an Evolving Complex System*, Vol. II, W. B. Arthur, S. Durlauf, and D. Lanl (eds.), Santa Fe Institute Studies in the Sciences of Complexity, Proceedings Volume XXVII, Reading, MA: Addison-Wesley. pp. 15-44.

Brock, W., Dechert, D. Scheinkman, J. and LeBaron, B. (1996). Test for Independence Based on the Correlation Dimension. *Econometric Reviews*, Vol. 15, pp. 197-235.

Chen, S.-H. and Tan, C.-W. (1996). Measuring Randomness by Rissanen's Stochastic Complexity: Applications to the Financial Data. In *Information, Statistics and Induction in Science*, D. L. Dowe, K. B. Korb and J. J. Oliver (eds.), Singapore:World Scientific, pp.200-211.

Chen, S.-H., Lux, T. and Marchesi, M. (2001). Testing for Non-Linear Structure in an Artificial Financial Market. *Journal of Economic Behaviour and Organization*, Vol. 46, Issue 3, pp. 327-342.

Chen, S.-H. and Yeh, C.-H. (2001). Evolving Traders and the Business School with Genetic Programming: A New Architecture of the Agent-Based Artificial Stock Market. *Journal of Economic Dynamics and Control*, Vol. 25, Issue 3-4, pp. 363-393.

Chen, S.-H., Yeh, C.-H. and Liao, C.-C. (2001). On AIE-ASM:Software to Simulate Artificial Stock Markets with Genetic Programming. In *Evolutionary Computation in Economics and Finance*, S.-H. Chen (ed.), Heidelberg:Physica-Verlag. pp. 107-122.

Den Haan, W. J. (2001). The Importance of the Number of Different Agents in a Heterogeneous Asset-Pricing Model. *Journal of Economic Dynamic and Control*, Vol. 25, Issue 5, pp. 721-746.

Egenter, E., Lux, T. and Stauffer, D. (1999). Finite-Size Effects in Monte Carlo Simulations of Two Stock Market Models. *Physica A*, Vol. 268, pp 250 - 256.

Grossman, S. J. and Stiglitz J. (1980). On the Impossibility of Informationally Efficiency Markets. *American Economic Review*, 70, pp. 393-408.

Harrald, P. (1998). Economics and Evolution. The panel paper given at the *Seventh International Conference on Evolutionary Programming*, March 25-27, San Diego, U.S.A.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press.

Palmer, R. G., Arthur, W. B., Holland, J. H., LeBaron, B. and Taylor, P. (1994). Artificial Economic Life: A Simple Model of a Stockmarket. *Physica D*, 75, pp. 264-274.

Tirole, J. (1982) On the Possibility of Speculation under Rational Expectations. Econometrica, 50, pp. 1163-1181.

Vriend, N. (2000). An Illustration of the Essential Difference between Individual and Social Learning, and Its Consequence for Computational Analysis. *Journal of Economic Dynamics and Control*, Vol. 24, Issue 1, pp. 1-19.

Vriend, N. (2001). On Two Types of GA-Learning. In *Evolutionary Computation in Economics and Finance*, S.-H. Chen (ed.), Heidelberg:Physica-Verlag. pp. 233-243.