Klaus Fischer
Ingo J. Timm
Elisabeth André
Ning Zhong (Eds.)

LNAI 4196

Multiagent System Technologies

4th German Conference, MATES 2006
Erfurt, Germany, September 2006
Proceedings

Springer

Lecture Notes in Artificial Intelligence 4196

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Klaus Fischer Ingo J. Timm
Elisabeth André Ning Zhong (Eds.)

Multiagent System Technologies

4th German Conference, MATES 2006
Erfurt, Germany, September 19-20, 2006
Proceedings

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Klaus Fischer
DFKI GmbH
Deduction and Multiagent Systems
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
E-mail: Klaus.Fischer@dfki.de

Ingo J. Timm
Universität Bremen
Technologie-Zentrum Informatik
Am Fallturm 1, 28359 Bremen, Germany
E-mail: i.timm@tzi.de

Elisabeth André
Universität Augsburg
Institut für Informatik
Eichleitnerstr. 30, 86135 Augsburg, Germany
E-mail: andre@informatik.uni-augsburg.de

Ning Zhong
Knowledge Information Systems Lab.
Maebashi Institute of Technology
460-1 Kamisadori-Cho, Maebashi-City 371-0816, Japan
E-mail: zhong@maebashi-it.ac.jp

Library of Congress Control Number: 2006932513

CR Subject Classification (1998): I.2.11, I.2, C.2.4, D.2.12, D.1.3, J.1

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-45376-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-45376-5 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11872283 06/3142 5 4 3 2 1 0

Preface

After three successful MATES conferences in Erfurt in 2003 and 2004 and in Koblenz in 2005, the 4th German conference on Multiagent System Technologies (MATES 2006) took place again in Erfurt collocated with Net.ObjectDays 2006. Building on other agent-related events in Germany in the past, and organized by the GI German Special Interest Group on Distributed Artificial Intelligence, the MATES conference series aims at promoting the theory and applications of agents and multiagent systems.

As in the past years, MATES 2006 provided a distinguished, lively and interdisciplinary forum for researchers, users, and developers of agent technologies, to present and discuss the latest advances of research and development in the area of autonomous agents and multiagent systems. Accordingly, the topics of MATES 2006 covered the whole range from theory to applications of agent and multiagent technologies. The technical program included a total of 23 scientific talks, and demonstrations of selected running agent systems.

The international Program Committee for MATES 2006 selected carefully 15 out of 52 submissions from all over the world to be accepted as full papers. Additionally, eight papers were selected for short presentations which are, however, published in *International Transactions on Systems Science and Applications*. The program included four distinguished invited speakers: Frank Dignum, Joaquim Filipe, Omer Rana and Hong Zhu.

Finally, as General Co-chairs and PC Co-chairs, and in the name of all members of the Steering Committee, we would like to thank all authors of submitted papers and all invited speakers for their contributions, all members of the Program Committee as well as other reviewers for their careful, critical, and thoughtful reviews, and all local conference organizers and others involved in helping to make MATES 2006 a success.

We hope the attendees enjoyed MATES 2006 and the conference site in Erfurt both scientifically and socially and will continue to support MATES as a conference series with many more successful events to come in the future!

July 2006

Elisabeth Andre, Ning Zhong
General Co-chairs

Klaus Fischer, Ingo J. Timm
Program Co-chairs

Organization

General Co-chairs

Elisabeth Andre	University of Augsburg, Germany
Ning Zhong	Maebashi Institute of Technology, Japan

Program Co-chairs

Klaus Fischer	DFKI Saarbrücken, Germany
Ingo J. Timm	University of Bremen, Germany

Program Committee

Bernhard Bauer	University of Augsburg, Germany
Michael Beetz	Technische Universität München, Germany
Wolfgang Benn	Chemnitz University of Technology, Germany
Federico Bergenti	University of Parma, Italy
Hans-Dieter Burkhard	Humboldt-Unversität zu Berlin, Germany
Monique Calisti	Whitestein Technologies, Switzerland
Cristiano Castelfranchi	NRC Rome, Italy
Thomas Christaller	Fraunhofer AIS, Germany
Rosaria Conte	NRC Rome, Italy
Stephen Cranefield	University of Otago, New Zealand
Hans Czap	Universität Trier, Germany
Mehdi Dastani	University of Utrecht, The Netherlands
Yves Demazeau	LEIBNIZ/IMAG, France
Jörg Denziger	University of Calgary, Canada
Torsten Eymann	University of Bayreuth, Germany
Ana Garcia Serrano	TU Madrid, Spain
Fausto Giunchiglia	University of Trento, Italy
Marie-Pierre Gleizes	IRIT Toulouse, France
Rune Gustavsson	Blekinge Institute of Technology, Sweden
Heikki Helin	TeliaSonera Helsinki, Finland
Heinrich Hussmann	Technische Universität Dresden, Germany
Toru Ishida	University of Kyoto, Japan
Catholijn Jonker	University of Nijmengen, The Netherlands
Hillol Kargupta	UMBC Baltimore, USA
Stefan Kirn	University of Hohenheim, Germany
Franziska Klügl-Frohmeyer	Universität Würzburg, Germany
Matthias Klusch	DFKI Saarbrücken, Germany
Ryszard Kowalczyk	Swinburne University of Technology, Australia

Daniel Kudenko	University of York, UK
Karl Kurbel	EU Viadrina Frankfurt, Germany
Winfried Lamersdorf	University of Hamburg, Germany
Jürgen Lind	Iteratec GmbH, Germany
Gabriele Lindemann	Humboldt-Universtität zu Berlin,Germany
Jiming Liu	Hong Kong Baptist University, China
Stefano Lodi	University of Bologna, Italy
Beatriz Lopez	University of Girona, Spain
Thomas Malsch	TU Hamburg-Harburg, Germany
Jürgen Müller	Berufsakademie Mannheim, Germany
Jörg P. Müller	Clausthal University of Technology, Germany
Werner Nutt	Heriot-Watt University of Edinburgh, UK
James Odell	Agentis Software, USA
Andrea Omicini	University of Bologna, Italy
Sascha Ossowski	Universidad Rey Juan Carlos Madrid, Spain
Paolo Petta	OEFAI Vienna, Austria
Stefan Poslad	Queen Mary University London, UK
Frank Puppe	Universität Würzburg, Germany
Alois Reitbauer	ProFACTOR, Austria
Wolfgang Renz	HAW Hamburg, Germany
Thorsten Scholz	University of Bremen, Germany
Heiko Schuldt	UMIT Innsbruck, Austria
Onn Shehory	IBM Research, Italy
John Shepherdson	British Telecom, UK
Von-Wun Soo	National Tsing Hua University, Taiwan
Steffen Staab	University of Koblenz-Landau, Germany
Robert Tolksdorf	Freie Universität Berlin, Germany
Adelinde Uhrmacher	Universität Rostock, Germany
Rainer Unland	University of Duisburg-Essen, Germany
Wiebe Van der Hoek	University of Liverpool, UK
Laszlo Zsolt Varga	MTA SZTAKI, Hungary
Daniel Veit	Universität Karlsruhe, Germany

External Reviewers

Kanishka Bhaduri	Koen Hindriks	Alexander Pokahr
Lars Braubach	Tobias John	Paul Sabatier
Xavier Clerc	Bin Li	Bjoern Schnizler
Klaus Dorer	Zekeng Liang	Klaus Stein
Souptik Dutta	Kun Liu	Paolo Turrini
Jean-Pierre George	Dimitri Melaye	Andrzej Walczak
Roberto Ghizzioli	Dagmar Monett	Anke Weidlich
Daniel Göhring	Gianluca Moro	Boris Wu
Rainer Hilscher	Lutz Neugebauer	Jian Ying Zhang

Table of Contents

Agent Communication and Interaction

Applications and Simulation

Agent Planning

Agent-Oriented Software Engineering

Trust and Security

Adding New Communication Services to the FIPA Message Transport System

Javier Palanca, Miguel Escrivá, Gustavo Aranda,
Ana García-Fornes, Vicente Julian, and Vicent Botti

Dept. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
{jpalanca, mescriva, garanda, agarcia, vinglada, vbotti}@dsic.upv.es

Abstract. Agent communication is one of the most important aspects in the multi-agent system area. In recent years, several works have been developed that are related to the agent communication problem.

This paper presents a new method for agent and agent platform communication in accordance with FIPA proposals. It uses the Jabber protocol as a new message transport protocol (MTP). This protocol provides additional services that are not included in the current standard FIPA MTP. It provides facilities for "presence notification", "multi-user conference" and "security services". As a result of this work, a new plug-in for the JADE platform that incorporates this transport protocol has been developed.

1 Introduction

Some multi-agent platforms allow external communication by means of using their own protocol and schema of communications. Good performance can be achieved this way, but it is not possible to inter-operate with a different platform that does not understand this particular way of communication. There are platforms that are capable of communicating using standard protocols agreed upon by consortiums like the FIPA [1]. The most widely used standard protocol proposed by the FIPA is the Hypertext Transfer Protocol (HTTP) which is a protocol that was originally conceived to transfer web pages from a server to a web browser. Although this protocol is rather simple and allows some kind of bidirectional communication between client and server, it is not designed to sustain lengthy conversations over time as it is focused towards a 'query and result' routine. These are detected drawbacks in current Message Transport Protocols for agent platforms.

This paper proposes a new method of communicating agents and platforms using a more *human-oriented* way to develop conversations: Jabber [2,3], an Instant Messaging (IM) protocol designed to sustain lengthy bidirectional communications among entities on the Internet. This new Message Transport Protocol has been proposed to the FIPA consortium as a new preliminary specification. Besides, this paper introduces new services into agent communication that are only

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 1–11, 2006.

possible by integrating the Jabber technology in the FIPA Message Transport System.

A software add-on to the JADE agent platform that uses the new communication structure presented in this work has been developed and accepted by the JADE development team [4].

Next sections are structured as follows: Section 2 introduces a brief description about Instant Messaging and, in particular, the Jabber protocol; Section 3 introduces our proposal of using Instant Messaging to communicate agents. This section describes how an Instant Messaging protocol can be used to improve agent communication taking advantage of its features (like presence notification or multi-user conference); Section 4 describes our contributions to the Jabber protocol in order to support FIPA agent communication. This contribution have been submitted to the FIPA Consortium; Finally, conclusions are presented in Section 5.

2 Jabber Protocol

Jabber is an open protocol that is based on standard XML (eXtensible Markup Language) for the exchange of messages and presence information between two Internet points. This protocol was proposed by the Jabber Software Foundation [5].

The main use of Jabber technology is an extensible IM network that has similar features to other IM services like AIM, ICQ, MSN Messenger and Yahoo. Therefore, Jabber is an open, secure, and free alternative to consumer IM services. Under the hood, Jabber is a set of streaming XML protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and other structured information in close to real-time.

Jabber is a communication protocol and can be applied to many scenarios. Its base is an XML-based messaging technology that can be used to communicate different entities. Its most common use is a decentralized instant messaging network, but this is not its sole application, as seen in the present work.

The term Jabber makes reference to a number of elements that, together, conform an infrastructure for the interchange of messages in real-time. The *technical* name of Jabber is XMPP, for eXtensible Messaging and Presence Protocol. It is a standard technology formalized by the Internet Engineering Task Force (IETF) based on the core protocols created by the open-source community [2]. The Jabber and XMPP terms can be used interchangeably.

XMPP technology has many advantages over existing proprietary closed ones. Being an open and free standard, which is also based in other open standards (XML, SSL, ...), it is not subject to sudden changes in its definition that may, for instance, render it to be incompatible with previous versions of the technology (as is the case with the MSN Messenger technology). It comes from a well-documented, open specification, which has eased the appearance and availability of many different types of clients, servers and *bridges* that enable the possibility

of interacting with other communication networks, such as email, SMS messages for mobile phones or other IM systems.

The Jabber network has a distributed architecture, which means that there is no central server that handles all the message delivery. Jabber servers cooperate together to bring messages from one user to another, like email servers do.

In the following list, we present the advantages of Jabber for the communication, specially over other similar proprietary networks: *Open, public, free, asynchronous, standard, tested, decentralized, secure, extensible and flexible.*

Bearing this in mind, it is clear that Jabber can be deployed to provide solutions far beyond the IM space as well as within it. Jabber applications beyond IM include network management, content syndication, collaboration tools, file sharing, gaming, remote systems monitoring and, now, agent communication.

3 Instant Messaging for Agents

In this section our contribution to a new agent communication framework is presented. This work presents the adaptation of the Instant Messaging technology to improve communication in multi-agent systems. We have studied instant messaging features that were interesting for agent communication and adapted them into a proposal for the FIPA standards.

3.1 Main Features

From a certain point of view, agents can be viewed as *'living'* intelligent entities, and so, all the advantages that benefit human users can also benefit agents. *'If IM is good for humans... Is it good for agents?'.*

Instant Messaging (IM) was initially developed to communicate humans. This technology has the ability of sending messages between two entities in a network. These messages are delivered in real-time (that's why it is called *'instant'* messaging) and are usually of text nature, although they can be any other kind of data (like binary or meta-information) if the protocol allows it.

Instant Messaging Networks also have more properties such as Presence Notification or Multi-User Conference. Presence Notification allows a user in the IM network to be notified when another user (who is a member of his/her contact list) changes his/her state.

Multi-User Conference is a technology that provides *chat rooms* where users can virtually *come together* and easily talk to more than one user. This feature is similar to the common Internet Relay Chats (IRC).

IM networks provide communication between human users in an unstructured way, using natural language. By changing the natural language used by humans for a common conversation with a standard structured language, like FIPA-ACL, and by inserting the necessary meta-information in a message (as shown in Section 4.1) it could be possible to re-use a common used Instant Messaging platform for agent communication. This is a more natural way to perform conversations between entities that involve interaction protocols.

Using an IM technology like Jabber for communicating agents seems like a natural step, since its innermost structure is quite analogous to what is commonly referred to as a multi-agent system (MAS) or platform. In IM, where there are users, in a MAS there are the agents; where there is a user directory, there is an agent directory; where there is a browseable components directory, there is a directory facilitator, and so on. From a certain point of view, one could say that agents behave as users that send or receive messages and make use of the resources and services of the platform.

When building a multi-agent platform, most of the design and implementation effort is put in the message transport mechanism. If an IM technology like Jabber is used, this *problem* is already solved. This model also simplifies the location issues of other users. The same user can connect from different locations without having to change addresses, the Instant Messaging server solves this.

Jabber is the ideal candidate to undergo this adaptation since it is standard, extremely adaptable, free, and has all the advantages and features presented in the previous section.

3.2 Presence Notification

Presence notification is one of the most useful features that Jabber provides. It is based on the Contacts List system. Contacts List is a mechanism through which a user can manage a list of *known friends* (called *roster*) to know at any time what the current status of any of the contacts is. Presence Notification allows a user to change its state in the network (e.g. *'Available', 'Busy', 'Don't disturb',* ...) to notify its contacts of its availability.

When a user changes its current status, all the other entities who share a bond with the user are automatically notified. An entity status usually means its availability and readiness to engage in a communication, but it is not closed to a short predefined list. Entities may define their own status and their meanings.

Two or more Jabber entities can be bonded (or *subscribed*) to each other. Entities who share a bond can make use of presence notification as described above. In order to form a bond, two entities must agree in a simple negotiation process that can be automated (–see Figure 1).

This simple and powerful system is useful to know when a contact is online and you can send messages to him/her. Otherwise, the contact is offline and a message sent to him/her will be stored in the server (so it will not be read in real-time).

Presence notification can be used for agent communication and Multi-Agent System platform communication. A *heartbeat pulse* (i.e., the ability for an entity to know when a bonded entity is online or not) can be built over this mechanism. This eliminates the need for a *'Ping'* service in the agents or in the platform, as all of them can be bonded to each other or to a hypothetical *central* AMS-like agent that can control the life-cycle of the rest of agents.

Bonds can also be used by agents to form *social circles*, that is, groups of agents that are aware of each other's presence and status. This eliminates the need for making several queries to a central *white pages* service or *yellow pages*

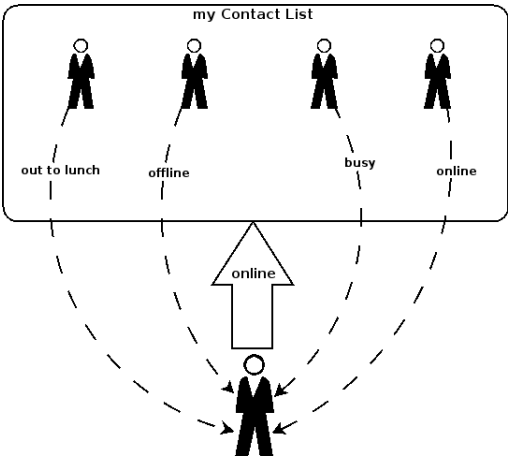


Fig. 1. Presence Notification

service to find out the availability of agents and services and, in the end, improves the general platform throughput.

3.3 Multi-User Conference

Another interesting Jabber feature is Multi-User Conference (MUC, in short). Jabber offers public conversation channels where users can join and make conversations with more than one user at the same time. These channels can be created by any entity (with an authorization in the server) and can be protected using a password, so only the entities that know the password can enter. A channel creator (or *administrator*) can also add or remove permissions to the entities that share a channel, such as the right to *speak* in it (–see Figure 2).

All these MUC features can be used to create forums dynamically for agents to connect to when performing some multi-agent activity. For instance, in a virtual auction scenario performed by agents, an agent playing the auctioneer role can create a password-protected channel and give the password only to previously certified agents. Then, in the channel, the auctioneer can give the right to speak only to agents who are going to play the bidder role, leaving the rest of agents in spectator mode only. Once the auction starts, every time a bidder wants to place a bid it just needs to *'speak'* in the channel, and all the other bidders and the auctioneer will know the bid at the same time.

MUC channels can also be used for more useful purposes. A simple method to send broadcasting announcements would be by sending a message to an appropriate chatroom. Every agent connected to that chatroom would receive the broadcast transparently. These chatrooms can be used to broadcast public platform information or to manage selected distribution lists. As an example imagine an scenario where a group of agents want to be notified when a particular

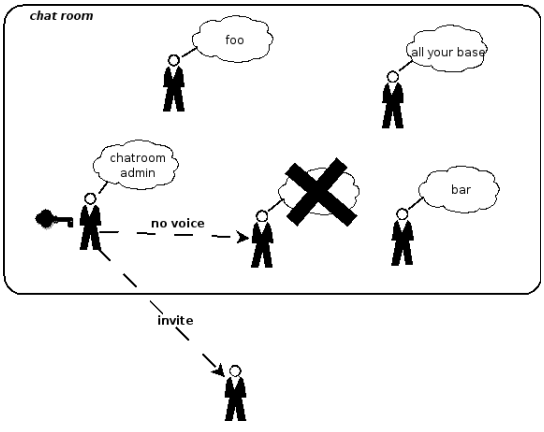


Fig. 2. Multi-User Conference

service is available. Thanks to Multi-User Conference mechanism, every agent that wants to be notified simply joins the chatroom associated with the service and when the service provider wants to announce the service availability sends a message to the chatroom. Note that the subscription list is automatically managed by the chatroom. Of course, this example does not replace the search service of the Directory Facilitator, it provides a new method similar to mailing distribution lists.

Using this powerful Jabber mechanism more MUC uses are quickly developed for agent communication improvement like federations, coalitions, hierarchies, etc. Also complex interaction protocols, like auctions, can be easily built with these facilities.

3.4 Distributed Network

As mentioned above, Jabber is decentralized by nature. Therefore, a Jabber user does not only choose a username, but also the server where the account will be created. What could at first seem to be a usability disadvantage, is one of the main advantages of Jabber as opposed to other IM systems.

For an agent, this means it can retain the same contact address even if it moves through different hosts, networks or even platforms. For a multi-agent platform, this feature provides several advantages. Let's see some aspects of this feature and their potential impact on an agent platform:

– Workload balance: Since there are servers distributed everywhere, divided by geography, by theme or by provider (university, city council, Internet provider...), the network workload generated by the users tends to distribute itself among all of them. In addition, some components of a Jabber server (users directory, network bridges, etc ...) can be executed in a different host than the one running the server itself, making the distribution better.

- Network reliability: When a Jabber server goes offline, the only affected users are those connected to that particular server. The rest of the Jabber network remains untouched. Therefore, only a relatively small number of users are affected by the mishap, instead of collapsing the entire communications network.
- Private services: The independence of the servers allows the set up of an IM network inside an intranet without needing any external resources from the Internet, as it is able to work on isolated networks.

3.5 Security

Using Jabber to communicate agents provides some built-in security mechanisms that help maintain the system's integrity.

- Logging to a Jabber server requires a **username and a password**. This mechanism prevents an unauthorized connection to the Jabber server if it has not been approved by the administrator. The administrator can set up the Jabber server to accept any registering attempt, or can set it up to deny any registering attempt, or only some privileged connections (i.e: some domain). This can be extended for agents in a way that only certain agents will be able to connect to a server (i.e join a platform).
- A connection to the Jabber server can be encoded with a symmetric cryptographic algorithm using SSL (Secure Socket Layer). SSL provides data cyphering, server **authentication**, message **integrity** and, optionally, client authentication for TCP-IP connections. This mechanism ensures that every message that flows through the Jabber network is strongly encoded and cannot be read by any malicious entity. This is necessary to ensure **confidentiality**.
- To ensure the authentication of a message, the Jabber server provides another mechanism to avoid identity theft. Jabber prevents identity spoofing by overriding a '*from*' field that does not match the sender's. In FIPA-ACL a conceptually similar message field exists: the '*sender*' field. This mechanism can be extended to watch and alter this '*sender*' field accordingly.

3.6 Performance and Scalability

Performance and scalability are important bottlenecks on current agent platforms [6, 7, 8, 9, 10]. Let's suppose a scenario where a Jabber server is integrated in a multi-agent platform as its message transporter to route XML messages. The server connected to the platform can be any Jabber server implementation, so that the most accurate version of the server can be selected depending on the platform constraints.

There is a reference implementation (called `jabberd`) that perfectly fulfills a standard use of a multi-agent platform (it supports around 10,000 concurrent connections). If more agent load is needed in the platform, other specialized servers can be used, like `WPJabber` (around 50,000 concurrent connections)

or `eJabberd`, a distributed (clustering) and fault-tolerant Jabber server that supports around $1,000,000$ concurrent connections (using the necessary *horsepower*).

3.7 Shared and Persistent Database

XML DataBase (XDB) is a Jabber server component that provides an interface to any kind of data source used to store and recover server-side data. A Jabber client can store any arbitrary XML on the server side.

Using this method, an agent can store private data on the server and retrieve it whenever necessary. Thanks to this mechanism agents can store the information needed to be persistent in their platform. An agent can recover this information even if it has been offline for a time. Part of the information stored in this XML Database can also be public. It is a simple method to advertise data to all the agents in the platform and make it accessible. The data stored might be anything, as long as it is valid XML. One typical usage for this namespace is the server-side storage of agent-specific preferences. Another usage is bookmark storage, scheduling and task information or any other interesting information.

4 Integrating Jabber in FIPA

The new Message Transport Protocol presented in this work is based on the data transfer representing the entire agent message, including the message envelope embedded in a XMPP message. Jabber uses three main building blocks in its communication: (the `<message/>` block, the `<presence/>` block and the `<iq/>` block), each of which has a different purpose. Jabber can perform all of its goals using only these blocks.

Next sections present the interface definition that has also been submitted to the FIPA consortium as a new preliminary specification.

4.1 Interface Definition

This new MTP has a proposed name called: `fipa.mts.mtp.xmpp.std`.

A FIPA Jabber message is represented by a `<message/>` Jabber element. Inside this element, there is the envelope of the FIPA message and the FIPA-ACL body of the message.

The message structure is detailed as follows:

- **Building Block**
  - The building block type is the `<message>` XMPP element.
  - The mandatory attribute `type` of the message element must have the value `normal` [2].
  - The `to` attribute of the message element is the physical XMPP address of the agent. (i.e. the Jabber ID of the agent or the Jabber ID of the platform)

```
<message id='2113' to='acc@foo.com' type='normal'>
        <body>
                (inform
                    :sender (agent-identifier
                        :name sender@bar.com
                        :addresses (sequence xmpp://acc@bar.com ))
                    :receiver
                       (agent-identifier
                        :name receiver@foo.com
                        :addresses (sequence xmpp://acc@foo.com )))
                    :language fipa-sl0
                    :ontology planning-ontology-1
                    :content   "((done task1))"
                )
        </body>
        <x xmlns='jabber:x:fipa' content-type='fipa.mts.env.rep.xml.std'>
                <envelope>
                    <params index="1">
                       <to>
                          <agent-identifier>
                            <name>receiver@foo.com</name>
                            <addresses> <url> xmpp://acc@foo.com </url> </addresses>
                          </agent-identifier>
                       </to>
                       <from>
                          <agent-identifier>
                             <name>sender@bar.com</name>
                             <addresses> <url> xmpp://acc@bar.com </url> </addresses>
                          </agent-identifier>
                       </from>
                       <acl-representation> fipa.acl.rep.string.std </acl-representation>
                       <payload-encoding> US-ASCII </payload-encoding>
                       <date> 20000508T042651481 </date>
                    </params>
                </envelope>
        </x>
</message>
```

Fig. 3. XMPP Example Message

– **Envelope Representation**
  - The envelope is placed in a `<x>` tag, which is used for ad-hoc extensions that add value, context, and information to any type of packet.
  - The XMPP namespace created for this purpose is called `jabber:x:fipa`.
  - The content type of the envelope is defined as an attribute of the `<x>` tag called `content-type`, where the value of the attribute is the component name given in each envelope specification.
  - The content of the `<x>` tag is the envelope body encoded in the defined representation.
– **Message Body**
  - The `<body>` tag of the message block contains the agent message payload. This payload is encoded with the representation described in the envelope.

Figure 3 shows an example of how a FIPA Jabber message is composed. This message is sent from the agent `sender@bar.com` to the agent `receiver@foo.com`,

which is resident on an Agent Platform that has an Agent Communication Channel with an external Jabber interface.

5 Conclusions

One of the main drawbacks detected on multi-agent platforms is the use of an inappropriate message transport protocol to communicate agents. To solve this problem, we have presented in this paper a new communication model using the Jabber Instant Messaging Protocol. We have also studied interesting Instant Messaging features and modified them in order to add new communication capabilities to multi-agent systems. This new communication model is FIPA compliant and supports all the required features, plus adding new ones that are made possible by using the Jabber technologies.

This protocol, which is based on Instant Messaging systems, uses a distributed network to route messages from one agent to another.

Nowadays, there are Instant Messaging Networks that are used by millions of people to communicate. These IM Networks can also be used by agents to communicate with each other.

Using the same IM Networks that humans do provides several advantages, such as a more comfortable and easy interaction between humans and agents. These networks have been extensively tested and can support a very high workload.

Other important matters are the new mechanisms presented in this paper (Presence Notification and Multi-User Conference) that can improve the agent communicative acts. These characteristics, which are not directly mentioned in the FIPA standard, perform new communication capabilities between agents which make them more versatile.

This work has been developed and tested with one of the most common and extended Multi-Agent Platforms: JADE. A plug-in [4] implementing this new Message Transport Protocol has been integrated in JADE and has been accepted by the JADE Team.

Thanks to this new MTP, a XMPP-capable platform (like the MAS platform developed by our research group: SPADE [11]) and a JADE platform can communicate with each other using the Jabber protocol. Actually, two or more JADE platforms can use this new XMPP plug-in to communicate their agents and take advantage of this new proposed Transport Protocol.

Acknowledgements

This work is partially supported by the TIC2003-07369-C02-01 and TIN2005-03395 projects of the Spanish government.

References

1. FIPA. Abstract architecture specification. Technical Report SC00001L, 2002.
2. Jabber Software Foundation. Extensible Messaging and Presence Protocol (XMPP): Core. Technical report, http://www.ietf.org/rfc/rfc3920.txt, October 2004.

3. Jabber Software Foundation. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Technical report,http://www.ietf.org/rfc/rfc3921.txt , October 2004.
4. JADE XMPP-Plugin. http://jade.tilab.com/community-3rdpartysw.htm.
5. Jabber Software Foundation. http://www.jabber.org, 2005.
6. L. Mulet, J.M. Such, J. M. Alberola, V. Botti, A. Espinosa, A. Garcia, and A. Terrasa. Performance Evaluation of Open Source Multiagent Platforms. In *Autonomous Agents and Multi-Agent Systems Conference (AAMAS06)*, 2006.
7. E Cortese, F Quarta, and G Vitaglione. Scalability and performance of JADE message transport system. In *In Proc. of AAMAS Workshop on AgentCities, Bologna*, 16 June 2002.
8. David Camacho, Ricardo Aler, Csar Castro, and Jos M. Molina. Performance evaluation of Zeus, JADE, and skeletonagent frameworks. In *In Proc. of the 2002 IEEE Systems, Man, and Cybernetics Conference*, 2002.
9. K. Burbeck, D. Garpe, and S. Nadjm-Tehrani. Scale-up and Performance Studies of Three Agent Platforms. In *International Performance Computing and Communications Conference (IPCCC 2004)*, 2004.
10. L. C. Lee, D. T. Ndumu, and P. De Wilde. The stability, scalability and performance of multi-agent systems. *BT Technology Journal*, 1998.
11. M. Escrivá, Palanca J., G. Aranda, A. García-Fornes, V. Julian, and V. Botti. A Jabber-based Multi-Agent System Platform. In *Autonomous Agents and Multi-Agent Systems Conference (AAMAS06)*, 2006.

Analysis of Multi-Agent Interactions with Process Mining Techniques

Lawrence Cabac, Nicolas Knaak, Daniel Moldt, and Heiko Rölke

University of Hamburg,
Department of Informatics,
Vogt-Kölln-Str. 30, D-22527 Hamburg
http://www.informatik.uni-hamburg.de/TGI/index_eng.html,
http://asi-www.informatik.uni-hamburg.de

Abstract. Process mining and multi-agent models are powerful techniques for the analysis of processes and organizations. However, the integration of both fields has seldom been considered due to the lack of common conceptual background. We propose to close this gap by using Petri nets as an operational semantics and consider process mining a useful addition to monitor and debug multi-agent systems in the development phase. Mining results can be represented in the formalized form of Petri nets that allows to validate or verify the actual behavior.

On our way to mining complex interactions within (simulated) organizations, we present a plug-in extension of our Petri net-based agent platform Mulan/Capa for recording interaction logs. Using process mining, the logs can be mapped by some intermediate steps to agent protocols e.g. represented as AgentUML interaction protocol diagrams. These diagrams are a descriptive representation form that combines organizational and control flow information. Furthermore, they can be mapped to executable Petri net, thus allowing to feed mining results back into the design phase.

Keywords: agent interactions, conversations, high-level Petri nets, interaction mining, mining, multi-agent systems, Mulan, modeling, nets-within-nets, process mining, reference nets, Renew, simulation.

1 Introduction

The concept of Multi-Agent Systems (MAS) has gained increasing importance in computer science during the last decade. MAS research considers systems as aggregations of goal-oriented, autonomous entities (agents) interacting in some common environment (see e.g. [1]). Since no or only minor central control is exposed on the agents, a coherent global system behavior emerges merely from their cooperative or competitive interactions.

The design, implementation, and validation of MAS still remains a demanding task. Petri nets are frequently applied for modelling agent behavior due to the

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 12–23, 2006.

typical combination of formal conciseness and visual clearness as well as the possibilities of displaying and formally analyzing concurrent systems [1]. Petri nets also support the verification and validation of MAS, since formal methods can be applied to assess liveness and safety properties of such models.

Unfortunately, the applicability of formal verification techniques is limited to simple and often practically irrelevant classes of MAS [2]. Furthermore such techniques can only be applied in a confirmative fashion; i.e. to verify (or falsify) previously posed hypotheses about a system's behavior. Agent-oriented software engineering (AOSE), however, is primarily an experimental process [2] consisting of prototypical design, simulation, observation and a-posteriori analysis in order to explore the system's behavior. Since the observation of even simple MAS might produce large and complex amounts of data [3], data mining has occasionally been proposed as a support technique for such analysis (see e.g. [4,5]).

To aid the understanding of dynamic processes – in particular interactions – in MAS, it seems straightforward to apply techniques from *process mining* originally developed in the domain of business process intelligence (see e.g. [6,7]). These techniques seem especially appropriate in Petri net-based AOSE due to their ability to reconstruct concurrent Petri net models from execution traces. This leads to a number of potentially interesting applications during the AOSE development cycle.[1] (1) In the *system analysis phase*, process mining can be employed to aggregate behavior or interaction traces of relevant agents from the real system to Petri net models that flow into the design phase. (2) In the *design phase*, process mining seems to be a promising approach to integrate adaptability into Petri net-based agents by providing them with the ability to learn executable models of behavior from the observation of other agents' interactions. (3) In the *validation phase*, process mining can be used to aggregate large amounts of trace data observed from the running system. These models can be visualized, formally analyzed or compared to design models to validate the system's behavior. Also, process mining might support the detection of unforeseen, implicit interaction patterns emerging at runtime.

In this paper, we present an approach towards the application of process mining techniques to the analysis, design and validation of multi-agent interactions. In particular, we pursue the goal of reconstructing models of agent interaction protocols from sample interactions. Our approach is integrated into the FIPA-compliant, Petri net-based agent platform Mulan/Capa.

The paper is organized as follows: Section 2 briefly introduces Mulan and Capa. In Section 3 we review existing work on agent interaction analysis and introduce process mining as an advanced analysis technique. In Section 4 we present our approach towards analyzing agent interactions by means of process mining, where Petri nets build an important intermediate representation. In Section 5 we discuss our prototypical implementation of a tool for interaction monitoring, debugging and validation. Finally, Section 6 concludes the paper with a discussion of our results reached so far and of possible future research.

[1] Similar applications of general data mining to MAS are discussed in [5].

2 The Multi-Agent System Architecture Mulan

The MAS architecture Mulan (MULti Agent Nets) [1] is based on the nets-within-nets paradigm [8], which is used to describe the natural hierarchies in an agent system. It includes four system views depicted in Figure 1: MAS infrastructure (1), platform (2), agent (3), and protocol (4). These are related by the mechanism of *token refinement*. A MAS is modelled as a Petri net whose places contain tokens representing platforms. Each platform is itself a net whose central place hosts all agents that currently reside on this platform. An agent consists of exactly one *agent net* that is its interface to the outside world and an arbitrary number of *protocol nets* defining its behavior. The variety of protocols ranges from simple linear step-by-step plans to complex dynamic workflows.

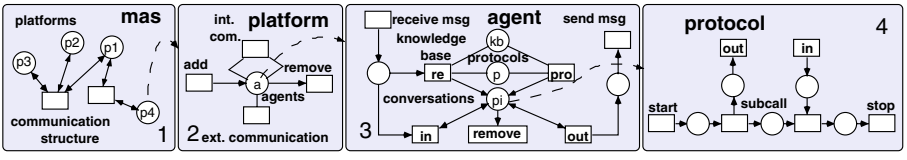


Fig. 1. Agent system as nets-within-nets

Mulan is implemented in reference nets and Java using the Petri net simulator Renew [9,10]. Reference nets provide a concurrency semantics and a natural concept of distribution and locality. Java allows us to seamlessly include external functionality such as graphical user interfaces. Compatibility of the Mulan framework to the FIPA specifications [11] is ensured through its partial re-implementation Capa (Concurrent Agent Platform Architecture) [12], which provides a seamless integration with other FIPA-compliant MAS frameworks. It also allows us to participate in heterogeneous environments such as OpenNet [13]. Especially in these environments, the determination, analysis and coordination of interactions are challenging tasks. In the following sections we show how these challenges can be approached using techniques from process mining.

3 Interaction Analysis and Process Mining

Interaction analysis is currently an important topic in MAS research for the reasons mentioned above. In the following, we review related work on interaction analysis, and introduce process mining as an advanced analysis technique.

3.1 Interaction Analysis in Multi-Agent Systems

Many frameworks for multi-agent application development include debugging tools that allow to monitor the message traffic on the agent platform. An example is the *Sniffer agent* integrated into the JADE framework [14]. This tool

displays observed message sequences as UML sequence diagrams and provides basic filtering capabilities. Monitoring agent interactions leads to large amounts of data. Important behavior patterns are in danger to go unrecognized when the analysis is performed by hand. Therefore data mining techniques are increasingly applied in this context (see e.g. [5]).

The algorithms for this task are mostly based on computational logic and stochastic automata: Nair et al. [4] e.g. propose an approach towards team analysis in the domain of (simulated) robot soccer (*RoboCup*). They consider three complementary perspectives: The *individual agent model* is a situational decision model of a single agent represented by means of association rules. The *multiple agent model* represents agent interactions as a stochastic automaton. The *global team model* shows relations between team properties (e.g. ball possession time) and game results in a rule-based fashion.

Botia et al. [15] focus on mining social networks at multiple resolutions from message logs using the *ROCK* cluster algorithm. In addition, their monitoring tool *ACLAnalyser* can automatically observe the execution of predefined interaction protocols on the JADE platform. Mounier et al. [16] present an approach towards *agent conversation mining* using stochastical grammar inference. Mining results are represented as a stochastic automaton whose edges are labelled with message performatives. The approach neglects concurrency and interaction roles. Hiel [17] applies extended Hidden Markov Models for the same task; also neglecting the aforementioned aspects. However, he suggests to improve the reconstruction of (concurrent) protocols by process mining techniques as a possible direction for future research. Parallel to the publication at hand, Dongen et al. report on the application of process and decision tree mining to communication logs observed in an auctioning simulation based on ad-hoc agent concepts [18]. This work proposes the introduction of adaptability by means of process mining.

3.2 Process Mining

Process mining (*workflow mining*) is a subfield of data mining concerned with "method[s] of distilling a structured process description from a set of real executions" [19]. The task is – given an *event log* recorded during process execution – to reconstruct properties of the generating processes. While most research is done in the area of *business process management* [6], other application domains such as the analysis of *web service interactions* [20] have recently been considered.

A large number of process mining techniques are available, that can be classified by the *perspective* that the analysis focuses on. The most prominent perspectives are control flow and organizational perspective [7]. The objective in the *control flow perspective* is to reconstruct the observed process' control structure – i.e. sequences, branches, loops and concurrency. The *organizational perspective* focuses on the "structure and the population" of the organization in which the processes are observed. This covers "relations between roles [...] groups [...] and other artifacts" [7]. Tool support for process mining is increasingly becoming available. Aalst et al. developed the *ProM* process mining tool that is extensible through a plugin mechanism by mining, import, export and analysis plugins [21].

An often-cited mining technique for the control flow perspective is the α-Algorithm: From an event-based process log, this algorithm builds a concurrent Petri net model on the basis of a *direct successor* relation. An extension of the algorithm can be proven to reconstruct any net belonging to the class of *extended sound workflow nets* [22], but it cannot cope with noise, hidden tasks, and duplicate tasks.[2] Herbst [6] developed an algorithm for mining process models containing duplicate tasks from activity-based logs.[3] Research on mining in the organizational perspective has so far focused on the reconstruction of *role assignments* [23,24] and *social networks* [25]. Further tasks in process mining are *log segmentation* (i.e. the mapping of messages from the process log to process instances and process classes) and *condition mining* (i.e. inference of branching conditions in the process model). Both are covered in an approach by Schütt [26].

Interaction mining – i.e. the reconstruction of interaction models from message logs – covers aspects of both control flow and organizational structure. Gombotz et al. [20] apply interaction mining to analyze the operation of web services at different levels (operation, interaction and workflow). One of the mining results is a so-called *web service interaction graph* representing the relations of a particular web service and its neighbors. Aalst [24] shows that the α-algorithm can be used to mine sequence diagram-like Petri net-structures from message logs. The approach is restricted to 1:1 interactions and does not explicitly abstract from senders and receivers to interaction roles.

4 An Approach Towards Agent Interaction Mining

Though the similarities between the analysis of multi-agent interactions and the research field of process mining have recently been recognized in the literature (see above), the integration of process mining into practical methods and tools for AOSE is still in its infancy. In the following, we present our approach towards analysing agent interactions with process mining techniques.

4.1 Context

Our approach towards Agent Interaction Mining (AIM) is integrated into a larger framework for Process Mining in (Agent Oriented) Software Engineering (ProMiSE, see [27]). This framework covers several analysis perspectives related to the four conceptual levels of MULAN: (1) the *decision perspective* focusing on decision models encoded in an agent's knowledge base, (2) the *internal control perspective* regarding the processes running within a single agent, (3) the *external control perspective* concerned with multi-agent interactions, (4) the *structural perspective* focusing on (static) platform and MAS structures, and (5) the *multi-level perspective* regarding relations between the perspectives mentioned before.

[2] A *hidden task* is a nameless activity not registered in the log. *Duplicate tasks* occur if the same activity is executed under different preconditions.

[3] In an activity-based log we can identify start and end events of activities which eases the detection of concurrency.

On our way to applying mining techniques to the analysis of MAS on multiple levels, we choose the *external control perspective* as a starting point. From the observation of message traffic, we proceed *bottom up*, i.e. we try to reconstruct basic interaction protocols in the first step. Through the recursive application of mining techniques to the results of the previous level, we aim to proceed to hierarchical protocols and higher level dynamical and structural patterns.

4.2 Techniques

The task of AIM at the protocol level is formulated as follows: Given a message log recorded during the execution of a MAS, find the unknown set of interaction protocols involved in the generation of this log. This task can be divided into several sub-phases depicted in Figure 2. Generally, we consider the FIPA ACL message attributes `performative`, `sender`, `receiver`, and some conversation control tags. By masking message content, we keep the following stages application-independent.

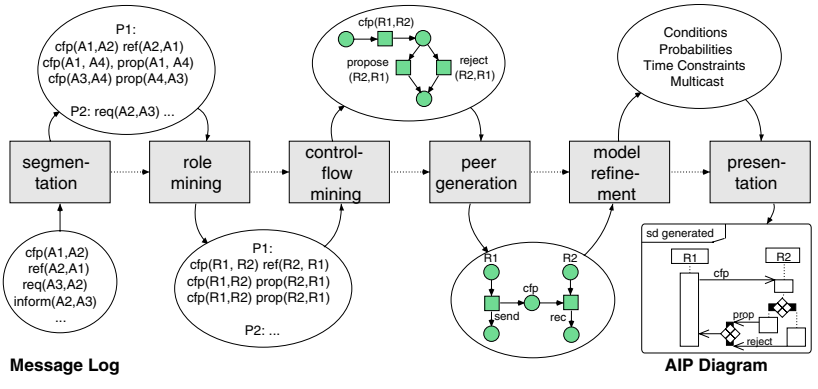


Fig. 2. A mining chain for agent interaction mining

The first phase – log segmentation – is necessary because a log normally contains messages from several conversations, generated by multiple protocols. These messages must be sorted by assigning them to a *conversation*; and by assignging each conversation to a *protocol type*. Given the information available in FIPA ACL messages (e.g. `conversation-id`) this segmentation is trivial.

However, these tags are not excessively used on the Capa platform and might generally prove as too inflexible for detecting complex patterns of interaction. Therefore, we reconstruct conversations by *chained correlation* [28] of messages based on the `in-reply-to` tag: Messages without this tag are assumed to start a new conversation. Other messages are appended to the conversation currently ended by a message with a corresponding `reply-with` tag. In doing so, we obtain $1:1$ *conversation threads*. However, these might be part of a larger multi-party

conversation that we reconstruct by merging all conversation threads sharing at least one `reply-with` or `in-reply-to` tag.

Assigning conversations to protocol types is a clustering task. For each conversation, we build a feature vector representing a *direct successor* relation of performatives.[4] Each vector component represents one possible succession of two performatives. It is assigned a value a counting the number of appearances of this succession in the conversation. To regard for protocols with a typically branched control structure, combinations of performatives appearing near the start of a conversation are weighted stronger than those appearing at the end. Finally, we apply the *nearest neighbour* algorithm [30] to cluster similar vectors based on the Euclidian distance.

The result of the segmentation phase are traces of conversations ordered by protocol types. In the second phase – role mining – we further abstract the messages by replacing names of sender and receiver agents with conversation roles. We currently apply a simple unification algorithm that binds agent names to role names in the order of their appearance in the conversation. However, this simple approach might fail in branched or concurrent protocols. Alternatively, we consider using a role detection mechanism based on sets of sent and received performatives similar to the approach described in [29].

In the third phase – control flow mining – we collect the abstracted conversation traces of each protocol type and try to induce a model of the protocol's control flow. Interaction protocols such as those specified in AgentUML might contain concurrent, hidden, and duplicate tasks. Therefore, the algorithm by Herbst [6] seems to be a good choice at first sight. However, this algorithm requires an activity-based log, while the message log is event-based.

Based on ideas from [6] and [26], our preliminary process mining technique consists of two stages – automata inference and concurrency detection: First, we reconstruct a deterministic finite automaton (DFA) from each set of samples using the k-RI algorithm [31]. The edges of the DFA are labelled with message performatives and sender/receiver roles. The k-RI algorithm can detect loops and duplicate tasks, but not concurrency. We therefore apply a modified version of the α-algorithm to the DFA next. Based on the successor relation of labelled transitions, the algorithm detects hints for concurrency in the DFA's structure.

Control flow mining results in an overall Petri net model of each protocol. This model can be split straightforwardly into protocol templates for every conversation role. Each of these peers corresponds to one lifeline in an AgentUML interaction protocol diagram (AIP, see [32]), that might be used to visualize the mining results. Additionaly, we are planning to refine the reconstructed model by infering temporal relations between messages with techniques described in [7]. We will also apply the C 4.5 decision tree learning algorithm [30] to reconstruct branching conditions from message content attributes as proposed in [6]. The attachment of branching conditions to the protocol templates leads to executable Mulan protocols.

[4] A similar metric is used in a preliminary approach by Vanderfeesten towards detecting conversation roles [29].

5 A Tool for Agent Interaction Mining

In this section, we present a prototypical tool and show an example for the application of our interaction mining techniques.

5.1 Monitoring Tool

To integrate process mining facilities into the CAPA platform, we developed a monitoring tool named MULAN *Sniffer* as a RENEW plugin [33]. The name indicates that the tool's functionality was derived from typical MAS debugging tools such as the *JADE Sniffer* [14]. The MULAN Sniffer monitors all ACL messages sent between agents on the platform during a simulation. The resulting message log is displayed textually as a list or graphically as a UML sequence diagram. Filters can be applied to select messages containing certain performatives, etc.

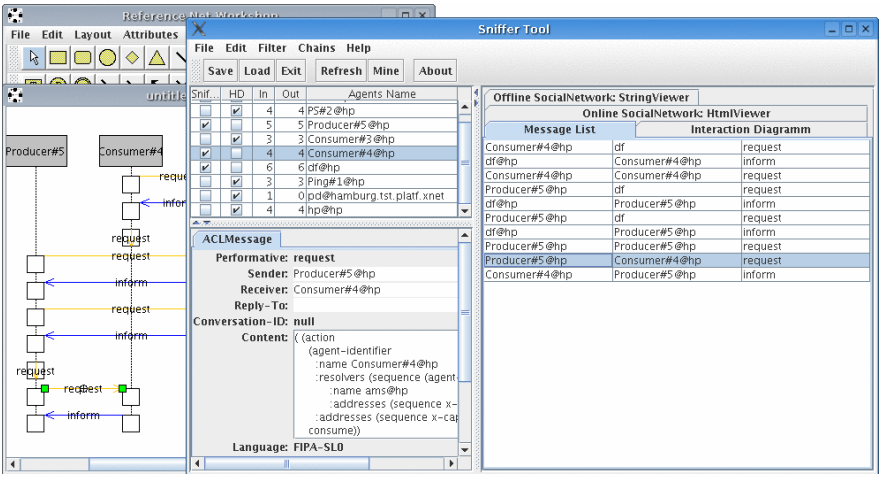


Fig. 3. MULAN Sniffer UI with observed interactions and RENEW UI

Figure 3 shows the user interface of the Sniffer with an observed message log. The messages in the diagram are color coded to ease the monitoring of the MAS. They can be inspected in the bottom left view of the Sniffer window. The upper left view shows a list of observed agents which can be sniffed or blocked. It also shows the numbers of messages sent and received per agent. The tool allows to observe changes in the diagram on the fly, i.e. when the message is sent.

The MULAN Sniffer differs from its 'ancestors' in two aspects that are important for our approach: (1) The recorded sequence diagrams are stored in the same format used by the MULAN design tools. They can therefore be edited and mapped to executable agent protocols. (2) More important, the *Sniffer* is a pluggable RENEW plugin [33] that can be extended by plugins for process mining and filtering itself.

The interfaces for filtering and mining plugins are reminiscent of similar tools such as *ProM* [21]. Special emphasis is put on the *recursive* character of process mining algorithms: These algorithms operate on data and provide data for higher-level analysis. We therefore introduce the concept of *mining chains*. Complex process mining algorithms are constructed by combining basic building blocks in data flow networks as proposed in [34]. This visual modeling technique is frequently used in data mining tools and can be supported by the Petri net editor of Renew.

5.2 AIM Plugin and Example

The example in Figure 4 shows a plugin that applies the algorithms described in Section 5 to the message log provided by the Sniffer. The messages partly result from multiple executions of a concurrent protocol simulating negotiations between a customer, a mediator and a service provider to allocate an order.

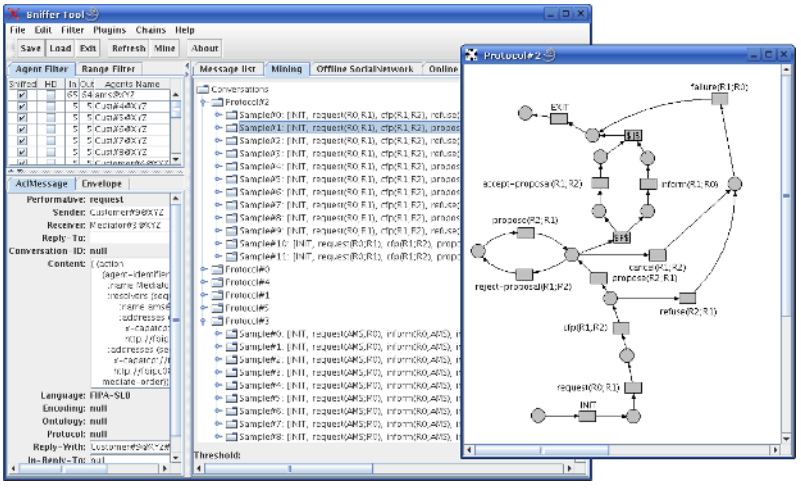


Fig. 4. AIM Plugin of the Mulan Sniffer showing mined conversations

In Figure 4 the Sniffer UI can be seen (background) with a tree-view that shows the results of the log segmentation. Each tree node represents one identified protocol type with the respective conversations as children. On selecting a conversation, the associated messages are automatically highlighted in the sequence diagram view (see also Figure 3).

In the example, the messages belonging to the order allocation protocol where successfully separated from the surrounding 'noise', i.e. conversations executed during the initialization of agents and platform. However, the performance of the clustering procedure strongly depends on a threshold for cluster similarity that needs careful calibration. The window in the foreground shows the correctly reconstructed Petri net model of the order allocation protocol.

6 Conclusion

The Mulan / Capa framework offers an integrated tool set supporting the development of Petri net-based MAS. It includes features for the specification, creation, documentation, monitoring and debugging of multi-agent applications. However, in concurrent, distributed and heterogeneous environments the analysis of multi-agent interactions is extremely difficult. Thus there is a need for elaborated techniques to handle large amounts of data. Process mining is one technique that can be successfully applied. The more abstract view of interaction mining allows to emphasize the desired perspectives (e.g. external control perspective) that are important for agent-based development and analysis.

This paper shows how to embed interaction mining into agent-oriented software engineering. We have developed an approach to reconstruct interaction protocols from message logs; integrating and extending several process mining techniques. It allows us to structure message logs by means of clustering and to reconstruct non-trivial concurrent protocols. However, we have encountered several cases the techniques cannot handle yet. Enhancing and validating them in greater detail is an important topic at issue. We have furthermore presented the Mulan Sniffer, a monitoring tool that is extensible by mining and filtering plugins. It is also applicable to many other FIPA-compliant MAS, which allows to monitor and mine in heterogeneous multi-agent environments and thereby evaluate our mining techniques in numerous real-world situations.

In our future work, we will validate the presented process mining techniques in empirical and analytical studies and provide necessary extensions. An important drawback of the current approach is the inability to appropriately handle multicast-protocols where the number of agents bound to each conversation role is not constant (e.g. the well-known *ContractNet* protocol). Furthermore, we are planning to tackle the challenging problem of reconstructing hierarchical protocols (see also [17]) and develop mining techniques for the other perspectives described in Section 4.1. In this context, the potential of Mulan as a conceptual framework for process mining will be investigated further in two directions: for providing a classification of techniques and for providing background knowledge to improve mining results.

Besides the analysis and validation of MAS, we are planning to apply process and interaction mining in a broader context of AOSE: On the one hand, we will improve the adaptability of Petri net-based agents through mining capabilities within our *Socionics* project [35]. On the other hand, the developed techniques might support online monitoring as well as the work of software developers in terms of reflecting their behavior to provide a feedback about best practices. Again this requires a sufficiently powerful toolset which we have sketched in [36]. In the context of inter-organizational processes the agent metaphor is highly applicable when considering the actors. Techniques that are applied to automated systems can be applied to the users in such environments as well. Therefore the legal, social, ethical and practical issues resulting from the application of process mining within the environment of people's computer support (with or without their knowledge) require urgent investigation.

References

1. Rölke, H.: Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen. Volume 2 of Agent Technology – Theory and Applications. Logos Verlag, Berlin (2004)
2. Edmonds, B., Bryson, J.: The insufficiency of formal design methods - the necessity of an experimental approach - for the understanding and control of complex MAS. In: AAMAS. (2004) 938–945
3. Sanchez, S.M., Lucas, T.W.: Exploring the world of agent-based simulations: Simple models, complex analyses. In Yücesan, E., Chen, C.H., Snowdon, J.L., Charnes, J.M., eds.: Proceedings of the 2002 Winter Simulation Conference. (2002) 116–126
4. Nair, R., Tambe, M., Marsella, S., Raines, T.: Automated assistants for analyzing team behaviors. In: Autonomous Agents and Multi-Agent Systems 8. (2004) 69 – 111
5. Remondino, M., Correndo, G.: Data mining applied to agent based simulation. In Merkuryev, Y., Zobel, R., Kerckhoffs, E., eds.: Proceedings of the 19th European Conference on Modelling and Simulation, Riga, SCS-Europe (2005) 374–380
6. Herbst, J.: Ein induktiver Ansatz zur Akquisition und Adaption von Workflow-Modellen. PhD thesis, University of Ulm (2001)
7. van der Aalst, W., Weijters, A.: Process mining: a research agenda. Computers in Industry **53**(3) (2004) 231–244
8. Valk, R.: Petri nets as token objects - an introduction to elementary object nets. In Desel, J., Silva, M., eds.: 19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal. Number 1420 in LNCS, Berlin, Springer-Verlag (1998) 1–25
9. Kummer, O.: Referenznetze. Logos-Verlag, Berlin (2002)
10. Kummer, O., Wienberg, F., Duvigneau, M.: Renew – The Reference Net Workshop. `http://www.renew.de` (2006) Release 2.1.
11. FIPA: Foundation for Intelligent Physical Agents. `http://www.fipa.org` (2005)
12. Duvigneau, M., Moldt, D., Rölke, H.: Concurrent architecture for a multi-agent platform. In Giunchiglia, F., Odell, J., Weiß, G., eds.: Agent-Oriented Software Engineering III. Third International Workshop, Agent-oriented Software Engineering (AOSE) 2002, Bologna, Italy, July 2002. Revised Papers and Invited Contributions. Volume 2585 of Lecture Notes in Computer Science., Berlin Heidelberg New York, Springer-Verlag (2003)
13. openNet: Project. `http://www.x-opennet.org/` (2005)
14. JADE: Java Agent DEvelopment Framework. `http://jade.cselt.it` (2005)
15. Botía, J., A.López-Acosta, A.F.Gómez-Skarmeta: ACLAnalyser: A tool for debugging multi-agent systems. In de Mántaras, R.L., Saitta, L., eds.: Proceedings of the 16th Eureopean Conference on Artificial Intelligence, Valencia, IOS (2004) 967–968
16. Mounier, A., Boissier, O., Jacquenet, F.: Conversation mining in multi-agent systems. In: Proceedings of the CEEMAS 2003. (2003) 158 – 167
17. Hiel, M.: Learning interaction protocols by overhearing. Master's thesis, Utrecht University (2005)
18. van Dongen, B., van Luin, J., Verbeek, E.: Process mining in a multi-agent auctioning system. In Moldt, D., ed.: Proceedings of the 4th International Workshop on Modelling of Objects, Components, and Agents, Turku (2006) 145–160
19. Maruster, L., Weijters, A., van der Aalst, W., van den Bosch, A.: Process mining: Discovering direct successors in process logs. In: ICDS: International Conference on Data Discovery, LNCS (2002)

20. Gombotz, R., Baina, K., Dustdar, S.: Towards web services interaction mining architecture for e-commerce applications analysis. In: International Conference on E-Business and E-Learning, Amman, Jordan, Sumaya University (2005)
21. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. In: ICATPN. (2005) 444–454
22. Medeiros, A., Dongen, B., Aalst, W., Weijters, A.J.M.M.: Process mining: Extending the α-algorithm to mine short loops. BETA Working Paper Series, WP 113, Eindhoven University of Technology (2004)
23. Ly, T., Rinderle, S., Dadam, P., Reichert, M.: Mining staff assignment rules from event-based data. In: Workshop on Business Process Intelligence (BPI), in conjunction with BPM 2005, Nancy, France (2005)
24. van der Aalst, W.: Discovering coordination patterns using process mining. In Bocchi, L., Ciancarini, P., eds.: First International Workshop on Coordination and Petri Nets (PNC 2004), STAR, Servizio Tipografico Area della Ricerca, CNR Pisa, Italy (2004) 49–64
25. van der Aalst, W., Song, M.: Mining social networks: Uncovering interaction patterns in business processes. In: Proceedings of the 2nd International Conference on Business Process Management, Potsdam (2004)
26. Schütt, K.: Automated modelling of business interaction processes for flow prediction. Master's thesis, University of Hamburg, Department for Informatics (2003)
27. Cabac, L., Knaak, N., Moldt, D.: Applying process mining to interaction analysis of Petri net-based multi-agent models. Technical Report 271, University of Hamburg, Department of Informatics (2006)
28. van der Aalst, W., Dumas, M., Ouyang, C., Rozinat, A., Verbeek, H.: Choreography conformance checking: An approach based on BPEL and petri nets. Technical Report BPM-05-25, BPMcenter.org (2005)
29. Vanderfeesten, M.: Identifying Roles in Multi-Agent Systems by Overhearing. Master's thesis, Utrecht University (2006) in preparation.
30. Dunham, M.H.: Data Mining: Introductory and Advanced Topics. Prentice Hall, Upper Saddle River (NJ) (2003)
31. Angluin, D.: Inference of reversible languages. Journal of the ACM **29**(2) (1982) 741–765
32. Cabac, L., Moldt, D., Rölke, H.: A proposal for structuring Petri net-based agent interaction protocols. In van der Aalst, W., Best, E., eds.: Lecture Notes in Computer Science: 24th International Conference on Application and Theory of Petri Nets, ICATPN 2003, Netherlands, Eindhoven. Volume 2679., Berlin Heidelberg: Springer (2003) 102–120
33. Cabac, L., Duvigneau, M., Moldt, D., Rölke, H.: Modeling dynamic architectures using nets-within-nets. In: Applications and Theory of Petri Nets 2005. 26th International Conference, ICATPN 2005, Miami, USA, June 2005. Proceedings. (2005) 148–167
34. Jessen, E., Valk, R.: Rechensysteme: Grundlagen der Modellbildung. Studienreihe Informatik. Springer-Verlag, Berlin (1987)
35. Homepage: Socionics in Hamburg. `http://www.informatik.uni-hamburg.de/TGI/forschung/projekte/sozionik/` (2005)
36. Lehmann, K., Cabac, L., Moldt, D., Rölke, H.: Towards a distributed tool platform based on mobile agents. In: Proceedings of the Third German Conference on Multi-Agent System Technologies (MATES). Volume 3550 of Lecture Notes on Artificial Intelligence., Springer-Verlag (2005) 179–190

Engineering Agent Conversations with the DIALOG Framework

Fernando Alonso, Rafael Fernández, Sonia Frutos, and Javier Soriano

School of Computer Science, Universidad Politécnica de Madrid,
28660 Boadilla del Monte, Madrid, Spain
{falonso, rfdez, sfrutos, jsoriano}@fi.upm.es

Abstract. This paper presents the rationale behind DIALOG: a formal framework for interaction protocol (IP) modeling that considers all the stages of a protocol engineering process, i.e. the design, specification, validation, implementation and management of IPs. DIALOG is organized into three views. The *modeling view* allows visual IP design. The *specification view* automatically outputs, from the design, the syntactic specification of the IPs in a declarative-type language called ACSL. This improves IP publication, localization and communication on the Web, as well as IP machine learning by agents. Finally, the *implementation view* provides a formal *structural operational semantics* (SOS) for the ACSL language. The paper focuses on the developed SOS, and shows how this semantics allows protocol property verification and eases automatic rule-based code generation from an ACSL specification for the purpose of simulating IP code execution at design time, as well as improving and assuring correct IP compliance at run time.

1 Introduction

Agent communication languages (ACLs) such as the *ARPA KSI Knowledge Query and Manipulation Language* (KQML) [1] and the *FIPA* Agent Communication Language (ACL) [2] are based on the concept of agents interacting with each other by exchanging typed messages that model the desired *communicative action* (e.g. inform, request, response, etc.), also referred to as *speech act* or *performative*, and a declarative representation of its content.

However, agents do not participate in isolated message exchanges, they enter into *conversations* [3], i.e. coherent message sequences designed to perform specific tasks that require coordination, such as negotiations or agreements. Societies of agents cooperate to collectively perform tasks by entering into conversations. In order to allow agents to enter into these conversations without having prior knowledge of the implementation details of other agents, the concept of *interaction protocols* (also known as *conversation policies*) has emerged [4]. *Interaction protocols* (IPs) are descriptions of standard patterns of interaction between two or more agents that may be as simple as request/response pairs or may represent complex negotiations involving a number of participants. They constrain the possible sequences of messages that can be sent amongst a set of agents to

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 24–36, 2006.

form a conversation of a particular type. A number of IPs have been defined, in particular as part of the FIPA standardisation process [5]. The importance of IPs in the design of an agent society is evident not only from their fitness for structuring behavior, but also as an organizational factor [6].

This approach to agent interaction necessarily depends on the provision of a *framework* to support the modeling of interactions between agents that considers all the stages of a *protocol engineering* process, i.e. the design, specification, validation, implementation and management of IPs considered as resources. Some relevant aspects to be taken into account when building such a framework are (a) the ease of modeling the communicative agent behavior,mainly, the behavior of agents that obey complex interaction patterns, (b) protocol maintainability and ease of reuse at both the design and specification level, (c) reliability, from the viewpoint of design validation and property verification and as regards assuring proper protocol compliance by participant agents, (d) availability and accessibility of both the protocols (i.e. designs and specifications) and ongoing conversations (i.e. protocol instances, protocol state and participant agents), being related to agent interoperability, and (e) scalability of both the designs and specifications (ease of composition) and the ongoing conversations for adaptation to large MAS. This paper presents the rationale behind DIALOG: a formal framework developed by the authors which deals with all these aspects at the IP architectural design, formal specification and implementation levels.

The remainder of the paper is organized as follows. Section 2 presents an overview of the DIALOG framework. We then concisely review in section 3 the fundamentals of the ACSL protocol specification language, which is at the core of the DIALOG *specification view*. Section 4 focuses on the *implementation view* and describes the formal *structural operational semantics* (SOS) that has been developed for the ACSL language. Finally, we conclude the paper in section 5.

2 DIALOG Framework Overview

The problem of IP specification is not new to *agent societies* developers, and a wide range of solutions have been proposed (cf. [7]). We find, however, that there is a huge void between the existing proposals based on formal techniques, whose design is extremely complex (e.g. Colored Petri Nets [6,8]), and the graphic notation-based techniques (e.g. AUML [9]), which are devoid of precise semantics and rule out automatic specification exchange in a machine readable language and interpretation for the purpose of specification simulation, validation and execution. DIALOG intends to fill this gap by means of three interrelated views:

- The *modeling view* eases the visual design of IPs by means of an AUML-based graphic notation [9]. The proposed notation ($AUML^{+}$) extends existing AUML and furnishes this notation with formal semantics. The latter is essential for developing the *specification view*. See [10] for a detailed description of this view, which has been ommitted here for the sake of briefness.
- The *specification view* automatically outputs the syntactic specification of an IP from its visual design in a declarative-type language called ACSL. This

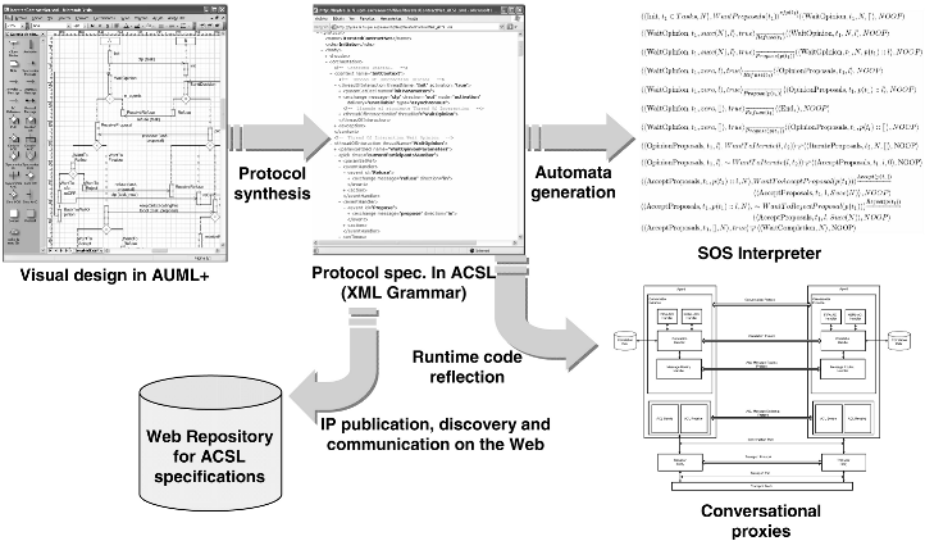


Fig. 1. Tools and artifacts of the DIALOG framework

improves IP publication, discovery and communication on the Web, as well as the machine learning of IP by agents. ACSL is an abstract syntax for which an XML grammar has been developed by means of the XML Schema formalism, in order to be able to validate the specifications syntactically, and to make easier their use in Internet environments. A KIF-based grammar is also available, and the mapping between both grammars is trivial by means of an XSLT-based parser. We concisely review the fundamentals of the ACSL language in section 3, see [11] for a more detailed description of this key component of the DIALOG framework.

– The *implementation view* is based on the provision of a formal *structural operational semantics* (SOS) for the ACSL language. The developed formal semantics allows us to verify the properties of the designed IPs, such as their termination in finite time, conversational state reachability or the absence of deadlocks or starvations. On the other hand, the developed SOS automatically outputs rule-based code from the ACSL specification for the purpose of (1) simulating protocol execution at design time and (2) improving and assuring correct IP compliance at run time. Section 4 focuses on this view.

Figure 1 gathers the different products of the IP engineering process and the tools of the proposed framework (consider each product and tool in the figure as a block. The details in each block are not necessary for understanding the figure). These tools allow: (1) the visual composition of IPs in $AUML^+$ notation, (2) automatic ACSL specifications generation (using an XML grammar) for models built in $AUML^+$, (3) the output of a SOS interpreter associated with these specifications, and (4) the generation, by means of code reflection techniques,

of conversational proxies that improve IP compliance at run time. Both the $AUML^+$ Editor and the ACSL/SOS Generator are open source tools. The source code is being distributed under GPL license, and is available from [12].

3 ACSL Language Fundamentals

The ACSL language defines an abstract syntax that establishes a vocabulary that provides a standard and formal description of the contractual aspects of IPs modeled using $AUML^+$ for use by design, implementation and execution monitoring libraries and tools. ACSL separates internal agent IP implementation from its external description. This is a key point for improving communication interoperability between heterogeneous agent groups and/or agents that run in heterogeneous agencies (platforms). It is based on ACL messages specifying the message flow that represents an IP between two or more agents and requires no special-purpose implementation mechanism.

The overall structure of a protocol specification in ACSL is composed of a *name*, a *header* and a *body*, all defined in the context of a block element *protocol*. The *name* element identifies the protocol for the purpose of referencing from other specifications in which it is to be embedded or with which it is to be interlinked. The *header* element declares the correlation sets and the properties used in the message exchanges for correlation and dynamic linking and to specify the semantic elements, respectively. The *body* of the protocol contains the specification of the basic exchange pattern. This item is formed by the composition of many *threadOfInteraction* elements that fork and regroup to describe the communicative behavior of the agent. The *threadOfInteraction* element is used to directly specify an exchange pattern or reference a protocol definition included in another specification by means of a qualified name (i.e. the conversation is specified in ACSL as IP composition).

A *threadOfInteraction* (Figure 2) combines zero or more atomic actions, references to subprotocols, conditional and iterative constructs and other *threadOfInteraction* that are interpreted sequentially and in the same order in which they are referenced. The sequence finishes when the last element ends.

The following describe such constructs with the level of detail necessary for understanding the remainder of the paper. Use cases of such constructs in ACSL specifications can be found in section 4, as they are needed. See [10] for a more detailed description of the ACSL abstract syntax.

Atomic actions are basic elements upon which an exchange pattern specification is built. ACSL includes four classes of basic actions, as shown in the *actionGroup* element decomposition illustrated in Figure 2: null action (*empty*), message exchange (*exchange*), protocol exception raising (*raise*) and time-outs (*delayFor*, *delayUntil*).

Message exchanges (*exchange* element) are the fundamental atomic actions in agent interaction. ACSL includes only the exchange properties that are part of the protocol specification, according to the ACL approach.

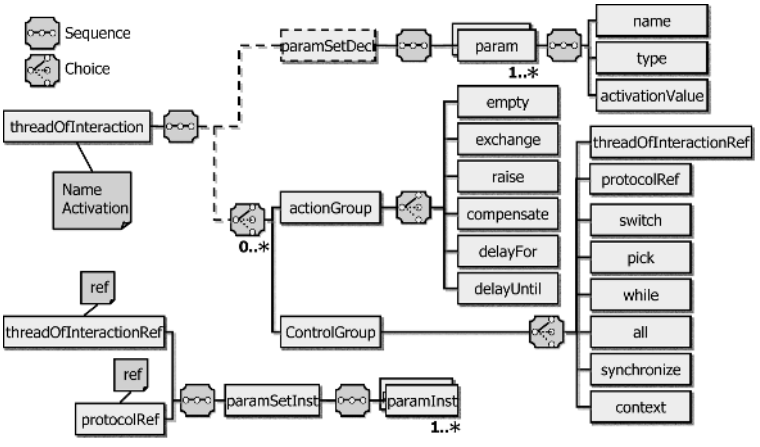


Fig. 2. Constructs of the language for specifying an exchange pattern

A *threadOfInteraction* eases the composition of an exchange pattern by means of a set of control constructs (*ControlGroup* in figure 2) that express conditional, concurrent and iterative interaction flows. These constructs are described below.

Switch: Expresses a conditional behavior equivalent to the XOR in AUML.
While: Repeats the exchange pattern determined by a *threadOfInteraction* an undefined number of times, until the given *condition* is no longer true.
All: Expresses the concurrent execution of a set of interaction flows that is not subject to any time order. *All* expresses the semantics of the AND connector in AUML notation.
Pick: Expresses precondition waits. It waits for the reception of an event (or set of events) and then executes an exchange pattern associated with this event. The possible events are message reception and end of a *delay* action.
Repeat: Repeats the exchange pattern given by a thread of interaction an pre-established number of times. The actual number of times it is repeated is opaque, i.e. is not part of the ACSL specification.
Synchronize: Establishes the set of threads of interaction that should be synchronized after an *All*, a multiple-choice *Switch* or an *Or*.

See [11] for a detailed description of the exception- and compensation-handling related ACSL constructs shown in Figure 2.

4 Implementation View: ACSL Semantics

The definition of an XML grammar for ACSL by means of the *XML Schema* formalism can only validate the IP specifications syntactically. To be able to validate and evaluate these specifications semantically, the ACSL language also needs to be furnished with formal semantics that can unambiguously describe the dynamic meaning of its syntactic constructs.

The provision of formal semantics for ACSL means that the IP specification can be analyzed to find out whether the IP has certain properties, such as termination in finite time, conversational state reachability or no deadlocks and starvations. On the other hand, the provision of operational semantics makes it possible to automatically derive IP implementation from protocol specification, easing its simulation and the automatic generation of proxies that assure that each participant effectively complies with the protocol rules and provides assistance for protocol machine learning.

The features of ACSL have led to the use of the concept of *Structural Operational Semantics* (SOS) [13,14] as an approach for specifying the dynamic meaning of IPs. The *dynamic meaning* of a protocol is obtained from the dynamic meaning of the different syntactic constructs that appear in its specification. It covers the execution of the specification, including expression evaluation, message sending and reception and the execution of other non-communicative actions.

The SOS denotes a formalism that can specify the meaning of a language by means of syntactic transformations of the programs or specifications written in this language. Some special points had to be taken into account to apply the SOS formalism, designed for programming languages, to a specification language such as ACSL. The definition of operational semantics suited for ACSL represents a three-step process:

1. Definition of a terminal and term-rewriting labeled transition system based on the operational semantics described in [15],
2. Definition of the interpreter I for this system, as proposed in [13], whose behavior is specified by a set of production rules.
3. Process of outputting the interpreter for each ACSL construct.

The following subsections describe the process of defining the operational semantics of an ACSL specification, stressing these points.

4.1 Defining the Transition System

This section presents the term-rewriting transition system developed for the ACSL language, based on a proposal by [15]. To produce a *term-rewriting labeled transition system* $\langle \Gamma, \Lambda, \rightarrow, \Upsilon \rangle$ to fit ACSL, it is established that $\Gamma \subseteq \Sigma \times \Theta$.

By way of a configuration, $\gamma = \langle \sigma, \theta \rangle \in \Gamma$ is composed of an identifier of the *thread of interaction* $\sigma \in \Sigma$ and the parameter set $\theta \subseteq \Theta$ that describes the runtime context for that thread, where Σ is the set of *threads of interaction* declared in the ACSL specification of a conversation and Θ is the set of parameters declared in those threads. In other words, a setting γ describes a conversational state of the specified protocol, Σ is the alphabet of conversational states and Θ is an adjustment parameter set for those states.

Similarly, the set of labels Λ is established as a set of pairs $\langle \phi, m \rangle$ composed of the cartesian product of the finite set of message exchanges $m \in M$ occurring in the ACSL specification with the set of predicates about the environment

$\phi \in \Phi(\omega)$. Based on this, the conditions of the language constructs are expressed as *pick*, *while* or *switch* (and, therefore, the parameters are declared by means of the *paramSetDecl* construct). Some of these predicates refer to exchanged messages (appear associated with *exchange* elements in *paramSetRef* elements) and are, therefore, denoted $\phi(m)$ and associated with the exchanged messages for the purpose of specifying this relation.

The transition relation $\rightarrow \subseteq \Gamma \times \Lambda \times \Gamma$ is now $\rightarrow \subseteq (\Sigma \times \Theta) \times (\Phi \times M) \times (\Sigma \times \Theta)$. The $\rightarrow$ relation therefore represents a transition relation between conversational states labeled by means of an action. The idea, as in other labeled transition systems, is that the action associated with a transition provides information about what is happening in the setting during the transition (internal actions) and/or about the interaction between the agents and their environment (external actions). In this case, the actions refer to interagent communication, and therefore the information they supply is the actual messages exchanged and the settings information (parameters of the conversational states) they use.

Accordingly, the language's alphabet is made up of the set of all possible messages exchanged in the course of a conversation $M = \{m_i, i = 1..n\}$. Hence, this language can be defined as a set of possible sequences of exchanged messages, each of which is a word of the language:

$$L \equiv \varepsilon \in L \mid m \in M \rightarrow m \in L \mid s_1, s_2 \in L' \rightarrow s_1 \odot s_2 \in L'$$

Finally, the set of term settings $\Upsilon \subseteq \Gamma$ is determined by those settings with interaction threads in which a message labeled as *term* is exchanged, as these are the only messages for which the following holds:

$$\forall \gamma \in \Upsilon, \forall \gamma' \in \Gamma \cdot \gamma \nrightarrow \gamma'$$

The transition system presented above is based on a definition given by [13]. However, the set of actions $\alpha \subseteq \Lambda(\omega)$ to be executed in each transition can be added to this definition. The transition relation would then be either:

$$\rightarrow \subseteq (\Sigma \times \Theta) \times (\Phi \times M) \times (\Sigma \times \Theta) \times \alpha \subseteq \Lambda(\omega)$$

or

$$\rightarrow \subseteq \Gamma \times \Lambda \times \Gamma \times \Lambda \rightarrow \subseteq (\Sigma \times \Theta) \times (\Phi \times M) \times (\Sigma \times \Theta) \times (\Lambda)$$

4.2 Defining the Interpreter

According to [14], the SOS formalism can be used to build the operational semantics of an ACSL specification by formally describing an interpreter I of that language whose behavior is specified by means of a set of production rules.

Following ideas taken from [16], I is modeled as a function whose argument is an ACSL-specified protocol$P$ and an environment ω, and which describes the behavior of $\langle P, \omega \rangle$ as an [in]finite series of productions like $\langle P, \omega \rangle \rightarrow \langle P_1, \omega_1 \rangle \rightarrow \langle P_2, \omega_2 \rangle \rightarrow \ldots$ If P ends, then the result is $\langle END, \omega_n \rangle$.

Accordingly, the automaton specification, which acts as an interaction protocol interpreter and therefore determines the operational semantics of an ACSL specification, defines a set of production rules that constitutes the definition of

the respective interpreter, whereas the sequences of messages sent and received is the program that is to be interpreted. Supposing that the agents are modeled on an internal BDI architecture, the set of beliefs (β), desires (δ) and intentions (ι) makes up the environment ω and predicates about that environment (e.g. WantToPropose(p), IntendToDo(t), etc.). Consequently, $\Phi(\omega) \subseteq \beta \cup \delta \cup \iota$ and the actions set (including the exchanged messages) are the lateral effects on ω in the same way as a variable is assigned in a programming language.

As mentioned earlier, the operational semantics developed is based on a production system that maps conversational states to new conversational states for a given ACSL specification. The format of a production rule is shown in the following.

This transition system can be viewed as a production system in which each transition is determined by a rule being fired when an action takes place and subject to the validity of the predicate ϕ. Let a transition be denoted

$$\langle\langle e \in \Sigma, \theta \in \Theta\rangle, \phi \in \Phi(\omega)\rangle \xrightarrow[msg/\phi(msg)]{} \langle\langle e' \in \Sigma, \theta' \in \Theta\rangle, [\alpha]/\alpha \in \Lambda(\omega)\rangle$$

This action can represent message sending, reception or an internal agent action. Therefore, three production rule types are accounted for: (a) production rules fired by message sending, if ϕ is true, (b)production rules fired by message reception, if ϕ is true, and (c) production rule fired by an internal agent action, if ϕ is true.

They all qualify the transition relation with $msg/\phi(msg)$ message template sending $\xrightarrow[msg/\phi(msg)]{}$ or reception $\xrightarrow{msg/\phi(msg)}$, or with $\overrightarrow{\varepsilon}$ for internal actions. The msg format is identical to the one used to represent a parameterized setting $\langle m \in M, \theta \in \Theta\rangle$, where M is the alphabet of performatives and θ is a parameter adjustment tuple for the message.

Using the predicate $\phi(msg)$ about the messages received, it is possible, for example, to find out if the message source already sent another message earlier (the source's membership of the group of agents participating in the protocol would also have to be considered). This predicate represents the part of the predicate that appears in the premise directly related to the message.

$$\phi(msg) = true \leftrightarrow \neg\exists m \in msgQ/m.from = msg.from$$

The parameterized description of a conversational state of the protocol $\langle e \in \Sigma, \theta \in \Theta\rangle$ is a state identifier e belonging to the states alphabet Σ and a tuple θ of adjustment parameters for the state e. The adjustment parameters tuple includes (1) variables of type Int, Char, List, Tuple for template adjustment or (2) variables for representing beliefs, desires and intentions.

Different types of template adjustment are accounted for depending on the parameter type to which they are applied. Accordingly, the adjustments considered for the type Int are $Succ(N), N \neq 0$ and $Zero$, and the transition function $Succ(Succ(N)) \rightarrow Succ(N)$. On the other hand, the adjustments $p :: rest$ and $[]$, and the transition function $p :: rest \rightarrow rest, rest \neq []$ and $p :: [] \rightarrow []$ are considered for the type List.

$\Phi(\omega)$ represents the set of predicates about the environment that can be evaluated by the agents participating in the conversation. These predicates are

usually intrinsically related to agents' beliefs, desires and intentions. However, no assumptions are made about how the agents conduct the evaluation, as this may be related not only to the protocol conversational states but also to the agents' internal state.

4.3 Process of Outputting the Interpreter for Each ACSL Construct

The following subsections detail the process of generating the interpreter for key ACSL language constructs. Concurrent and synchronization related ACSL constructs are left for a forthcoming paper.

Simplifications. The following simplification rule is used with the aim of simplifying the generation of production rules in embedded constructs:

If there are two rules

$$\langle\langle A \in \Sigma, \theta \in \Theta\rangle, \phi \in \Phi(\omega)\rangle \xrightarrow[msg/\phi(msg)]{} \langle\langle B \in \Sigma, \theta' \in \Theta\rangle, \alpha \in \Lambda(\omega)\rangle$$
$$\langle\langle B \in \Sigma, \theta' \in \Theta\rangle, \phi' \in \Phi(\omega)\rangle \xrightarrow{\varepsilon} \langle\langle C \in \Sigma, \theta'' \in \Theta\rangle, \alpha' \in \Lambda(\omega)\rangle$$

they can be simplified as

$$\langle\langle A \in \Sigma, \theta \in \Theta\rangle, \phi \in \Phi(\omega)\rangle \xrightarrow[msg/\phi(msg)]{} \langle\langle C \in \Sigma, \theta'' \in \Theta\rangle, \alpha' \in \Lambda(\omega)\rangle$$

Provided that $\phi \rightarrow \phi'$. The same applies to message sending rules.

Production Rules for Receiving n Messages. The delay in receiving n messages from different agents is expressed by means of the pick instruction:

```
Pick := pick [Times] [ParamSetRef] {EventHandler} [OnTimes]
EvenHandler := eventHandler Event ActionBlock
OnTimes := onTimes (ThreadOfInteraction | ProtocolControlGroup | Action)
EventHandler := eventHandler Event ActionBlock
Event := event Id (DelayFor | DelayUntil | Exchange | Catch)
ActionBlock := action (ThreadOfInteraction | ProtocolControlGroup | Action)
Times := Expression
Id := id string
```

The following standard set of SOS production rules is obtained by each *EventHandler* for this instruction:

$$\langle\langle A \in \Sigma, (\ldots Succ(t) \ldots)\rangle, true\rangle \xrightarrow[msg_1/\phi(msg_1)]{} \langle\langle A, (\ldots (t) \ldots)\rangle, [\alpha \in \Lambda(\omega)]\rangle$$
$$\langle\langle A \in \Sigma, (\ldots Succ(t) \ldots)\rangle, true\rangle \xrightarrow[msg_2/\phi(msg_2)]{} \langle\langle A, (\ldots (t) \ldots)\rangle, [\alpha \in \Lambda(\omega)]\rangle$$
$$\ldots \langle\langle A \in \Sigma, (\ldots Zero \ldots)\rangle, true\rangle \xrightarrow{\varepsilon} \langle\langle B \in \Sigma, \{\theta \in \Theta\}\rangle, [\alpha \in \Lambda(\omega)]\rangle$$

where the expressions $(\ldots Succ(t) \ldots)$ include all the parameters referenced in the body of the pick (including events) and the expression $(\theta \in \Theta)$ will be composed of the set of values that instantiate the thread parameters referenced in the <onTimes> expression.

All the pick handlers have been assumed to be concerned with message delay. Otherwise, the handler delay condition would be stated in $\phi \in \Phi(\omega)$, which would no longer be *true*, and the transition rule would switch to $\xrightarrow{\varepsilon}$.

Accordingly, for the next use of *pick*, taken from the ACSL specification of the *FIPA IteratedContractNet* protocol [5] (the *proposer* agent gather *inform*, *failure*, and *end* messages from participants):

```
<pick times="length(apl)">
  <paramSetRef><paramRef mode="match">apl</paramRef></paramSetRef>
  <eventHandler> <event> <exchange message="Inform" direction="in" mode="middle">
                           <paramSetRef> <paramRef mode="adjust">p(t1)-->ipl</paramRef>
                                         <paramRef mode="match">t1</paramRef>
                           </paramSetRef></exchange></event>
                 <action><empty/></action></eventHandler>
  <eventHandler> <event> <exchange message="Failure" direction="in" mode="middle">
                          <paramSetRef> <paramRef mode="adjust">p(t1)-->fpl</paramRef>
                                        <paramRef mode="match">t1</paramRef>
                          </paramSetRef></exchange></event>
                 <action><empty/></action></eventHandler>
  <onTimes> <threadOfInteractionRef threadRef="End">
              <paramSetInst> <paramInst> <ref>t1</ref> <value>t1</value></paramInst>
                             <paramInst> <ref>pl</ref> <value>fpl</value></paramInst>
              </paramSetInst></threadOfInteractionRef> </onTimes> </pick>
```

we get the following SOS rules:

$$\langle\langle A \in \Sigma, (t_1, Succ(n), fpl, ipl)\rangle, true\rangle \xrightarrow[Fail(p(t_1))]{} \langle\langle A, (t_1, n, p :: fpl, ipl)\rangle, NO)\rangle$$
$$\langle\langle A \in \Sigma, (t_1, Succ(n), fpl, ipl)\rangle, true\rangle \xrightarrow[Inform(p(t_1))]{} \langle\langle A, (t_1, n, fpl, p :: ipl)\rangle, NO)\rangle$$
$$\langle\langle A \in \Sigma, (T_1, Zero, fpl, ipl)\rangle, true\rangle \overrightarrow{}_{\varepsilon} \langle\langle B \in \Sigma, t_1, fpl\rangle, NO)\rangle$$

Iteration Production Rules. Iterations are expressed in ACSL by means of the while instruction:

```
While := while Condition ActionBlock
Condition := condition [ParamSetRef] Expression
ActionBlock := action (ThreadOfInteraction | ProtocolControlGroup | Action)
```

for which the following set of standard SOS production rules is obtained:

$$\langle\langle A \in \Sigma, \theta \in \Theta\rangle, \phi \in \Phi(\omega)\rangle \overrightarrow{}_{\varepsilon} \langle\langle A \in \Sigma, \theta_1 \in \Theta\rangle, \alpha \in \Lambda(\omega)\rangle$$
$$\langle\langle A \in \Sigma, \theta_2 \in \Theta\rangle, \phi \in \Phi(\omega)\rangle \overrightarrow{}_{\varepsilon} \langle\langle B \in \Sigma, \theta_3 \in \Theta\rangle, \alpha \in \Lambda(\omega)\rangle$$

When the while instruction ends, Θ converges to a state in which θ_2 is true.

The following example assumes that a message is to be sent to all the agents identified in a list:

```
<threadOfInteraction>
 <while> <condition condition="existProposalInProposals">
           <paramSetRef> <paramRef mode="adjust">p(t1)::pl</paramRef>
                         <paramRef mode="match">t1</paramRef></paramSetRef> </condition>
         <action> <exchange message="msg" direction="out" delivery="unreliable"
                                          mode="middle" type="asynchronous">
                    <paramSetRef> <paramRef mode="match">p.from</paramRef>
                                  <paramRef mode="match">p</paramRef></paramSetRef>
      </exchange></action></while> </threadOfInteraction>
```

for which the following set of SOS rules is produced:

$$\langle\langle A \in \Sigma, \ldots p :: l \ldots\rangle, \phi \in \Phi(\omega)\rangle \xrightarrow[msg(p.from)]{} \langle\langle A \in \Sigma, \ldots l \ldots\rangle, \alpha \in \Lambda(\omega)\rangle$$
$$\langle\langle A \in \Sigma, \ldots [] \ldots\rangle, \phi \in \Phi(\omega)\rangle \overrightarrow{}_{\varepsilon} \langle\langle B \in \Sigma, \theta \in \Theta\rangle, \alpha \in \Lambda(\omega)\rangle$$

In this case, the while instruction is guaranteed to end, since $\Theta \equiv p :: l$ y $\theta_1 \equiv l$, which necessarily has Θ converge to [], making θ_2 true.

Optionality Production Rules. ACSL can be used to express optionality in the course of a conversation by means of the *switch* construct. The overall structure of this construct is shown below:

```
Switch := switch Multichoice {Branch} [Default]
Multichoice := multichoice boolean
Branch := branch Case ActionBlock
Default := default (ThreadOfInteraction | ProcolControlGroup | Action)
Case := case Condition [ParamSetRef]
Condition := condition [ParamSetRef] Expression
ActionBlock := action (ThreadOfInteraction | ProtocolControlGroup | Action)
ParamSetRef := paramSetRef {ParamRef}
ParamRef := paramRef Mode string
Mode := match | adjust
```

The following rule template is obtained for each *branch*:

$$\langle\langle A \in \Sigma, \{\theta_i \in \Theta\}\rangle, \phi \in \Phi(\omega)\rangle \overrightarrow{\varepsilon} \langle\langle B_j \in \Sigma, \{\theta'_k \in \Theta\}\rangle, [\alpha \in \Lambda(\omega)]\rangle$$

where $A \in \Sigma$ denotes the conversational state generated for the switch instruction, $\{\theta_i \in \Theta\}$ is the list of referenced parameters (*paramSetRef*) in the respective branch condition, $\phi \in \Phi(\omega)$ is the actual condition, $B_j \in \Sigma$ denotes another conversational state that will be used in the antecedent of the rules generated for the interaction thread defining the action of this branch. If this is a reference to an interaction thread, $\{\theta'_k \in \Theta\}$ will be the set of values that instantiate the parameters of the respective thread (*paramSetInst*).

The same rule template is obtained for the *default* branch considering:

$$\phi = \bigcup_{i=1}^{n} \phi_i \qquad \text{and} \qquad \theta = \neg \bigvee_{i=1}^{n} \theta_i$$

The next section gives an example of a set of optionality and iteration rules.

Structure Composition Rules. The rules resulting from applying the templates discussed in earlier sections (including the simplification template) to an ACSL specification are shown below. This ACSL specification (excerpt) is made up of a *while* structure plus a *switch* structure. The fragment, taken from the ACSL specification of the *FIPA IteratedContractNet* protocol, models how the proposer, after receiving the proposals from the contract net, notify to each agent having sent a proposal if its proposal has been accepted (*Accept* message) or rejected (*Reject* message):

```
<threadOfInteraction>
 <while> <condition condition="existProposalInProposals">
        <paramSetRef> <paramRef mode="adjust">p(t1)::pl</paramRef>
                      <paramRef mode="match">t1</paramRef> </paramSetRef></condition>
        <action> <switch multiChoice="false">
                 <branch> <case condition="WantToAcceptProposal">
                           <paramSetRef> <paramRef mode="match">p</paramRef>
                                         <paramRef mode="adjust">p-->apl</paramRef>
                           </paramSetRef></case>
                          <action> <exchange message="Accept" direction="out" mode="middle"
                                             delivery="unreliable" type="asynchronous">
                                   <paramSetRef> <paramRef mode="match">p.id</paramRef>
                                                 <paramRef mode="match">p</paramRef>
                                   </paramSetRef></exchange></action></branch>
```

```
<branch> <case condition="WantToRejectProposal">
          <paramSetRef> <paramRef mode="match">p</paramRef>
          </paramSetRef></case>
         <action> <exchange message="Reject" direction="out" mode="middle"
                            delivery="unreliable" type="asynchronous">
                  <paramSetRef> <paramRef mode="match">p.id</paramRef>
                                <paramRef mode="match">p</paramRef>
                  </paramSetRef></exchange></action></branch>
</switch> </action> </while> </threadOfInteraction>
```

Firstly, before any simplifications are made, we have:

$$\langle\langle \mathrm{A}, t_1, p(t_1) :: pl, apl\rangle, true\rangle \overrightarrow{\tau}\langle\langle \mathrm{B}, t_1, pl, p, apl\rangle, \mathrm{NO}\rangle$$
$$\langle\langle \mathrm{B}, t_1, pl, p, apl\rangle, WantToAcceptProposal(p)\rangle \xrightarrow{Accept(p)} \langle\langle \mathrm{A}, t_1, pl, p :: apl\rangle, NO\rangle$$
$$\langle\langle \mathrm{B}, t_1, pl, p, apl\rangle, WantToRejectProposal(p)\rangle \xrightarrow{Reject(p)} \langle\langle \mathrm{A}, t_1, pl, apl\rangle, NO\rangle$$
$$\langle\langle \mathrm{A}, t_1, [], apl\rangle, true\rangle \overrightarrow{\tau}\langle\langle \mathrm{C}, l\rangle, \mathrm{NO}\rangle$$

which, simplified (according to), becomes:

$$\langle\langle \mathrm{A}, t_1, p(t_1) :: pl, apl\rangle, WantToAcceptProposal(p)\rangle \xrightarrow{Accept(p)} \langle\langle \mathrm{A}, t_1, pl, p :: apl\rangle, NO\rangle$$
$$\langle\langle \mathrm{A}, t_1, p(t_1) :: pl, apl\rangle, WantToRejectProposal(p)\rangle \xrightarrow{Reject(p)} \langle\langle \mathrm{A}, t_1, pl, apl\rangle, NO\rangle$$
$$\langle\langle \mathrm{A}, t_1, [], apl\rangle, true\rangle \overrightarrow{\tau}\langle\langle \mathrm{C}, l\rangle, \mathrm{NO}\rangle$$

Sequential Statement Composition Rules. The transition for the sequential statement composition $s_1; S$ is derived from the transition for the statement s_1.

$$\frac{\begin{array}{l}\langle\langle A \in \Sigma, s_1, \theta_1 \in \Theta\rangle, \phi_1 \in \Phi(\omega)\rangle \overrightarrow{\tau}\langle\langle A_1 \in \Sigma, \phi, \theta_1' \in \Theta\rangle, \alpha_1 \in \Lambda(\omega)\rangle \ldots \\ \langle\langle A \in \Sigma, s_1, \theta_2 \in \Theta\rangle, \phi_2 \in \Phi(\omega)\rangle \overrightarrow{\tau}\langle\langle A_2 \in \Sigma, \phi, \theta_2' \in \Theta\rangle, \alpha_2 \in \Lambda(\omega)\rangle\end{array}}{\begin{array}{l}\langle\langle A \in \Sigma, s_1; S, \theta_1 \in \Theta\rangle, \phi_1 \in \Phi(\omega)\rangle \overrightarrow{\tau}\langle\langle A_1 \in \Sigma, S, \theta_1' \in \Theta\rangle, \alpha_1 \in \Lambda(\omega)\rangle \ldots \\ \langle\langle A \in \Sigma, s_1; S, \theta_2 \in \Theta\rangle, \phi_2 \in \Phi(\omega)\rangle \overrightarrow{\tau}\langle\langle A_2 \in \Sigma, S, \theta_2' \in \Theta\rangle, \alpha_2 \in \Lambda(\omega)\rangle\end{array}}$$

The DIALOG project Web site [12] contains, both for reference and for better understanding of the paper, examples of complete $AUML^+$ diagrams, ACSL specifications, and SOS interpreters for a number of relevant IPs.

5 Conclusions

In this paper, we have stressed that the problem with existing approaches to agent interaction modeling is that there is a huge void between the proposals based on formal techniques, whose design remains extremely complex, and the graphic notation-based techniques, which are devoid of precise semantics and rule out automatic specification exchange and interpretation for the purpose of specification simulation, validation and execution.

Bearing this in mind, we have presented the rationale behind DIALOG: a formal framework that considers all the stages of a protocol engineering process, i.e. the design, specification, validation, implementation and management of IPs, thanks to the three views into which it is organized. The paper has focused on the developed SOS and has highlighted how this formal semantics allows protocol property verification and eases automatic rule-based code generation from an

ACSL specification for the purpose of simulating IP code execution at design time, as well as improving and assuring correct IP compliance at run time. We have also stressed throughout the paper how the availability of a syntactic specification of an IP in a declarative-type machine-readable language such as the proposed ACSL helps to improve IP publication, discovery and communication on the Web, as well as the machine learning of IP by agents.

References

1. Finin, T; Labrou, Y.; and Mayfield, J. KQML as an agent communication language. In J. M. Bradshaw (Ed.) Software Agents. MIT Press (1997)
2. Foundation for Intelligent Physical Agents. FIPA ACL message representation in string specification. http://www.fipa.org/specs/fipa00070/ (2000)
3. McBurney, P., Parsons, S., and Wooldridge, M. Desiderata for Agent Argumentation Protocols. In Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS02), Bologna, Italy (2002)
4. Greaves, M.; Holmback, H.; and Bradshaw, J. What is a conversation policy?. In F. Dignum and M. Greaves (Eds.) Issues in Agent Communication, volume 1916 of Lecture Notes in Artificial Intelligence, pages 118–131. Springer (2000)
5. Foundation for Intelligent Physical Agents. FIPA Interaction protocol Library Specification. http://www.fipa.org/specs/fipa00025, FIPA (2001)
6. Hanachi, C., Sibertin-blanc, C.: Protocol Moderators as Active Middle-Agents in Multi-Agent Systems. Autonomous Agents and Multi-Agent Systems, 8, 131-164, Kluwer Academic Publishers, The Netherlands (2004)
7. Dignum, F, Greaves, M (eds.): Issues in Agent Communication. LNAI 1916 State-of-the-Art Survery , Springer, Heidelberg (2000)
8. Gutnik, G. and Kaminka, G.A. Representing Conversations for Scalable Overhearing, Journal of Artificial Intelligence Research, Volume 25, pages 349–387 (2006)
9. Odell J. et al. Representing agent interaction protocols in UML. In Proceedings of 1st International Workshop on Agent-Oriented Software Engineering, Limerick, Ireland (2000)
10. Alonso, F; Frutos, S; López, G; and Soriano, J. A Formal Framework for Interaction Protocol Engineering, LNAI, vol. 3690, pp. 21-30, Springer-Verlag: Berlin (2005)
11. Soriano, J; Alonso, F; and López, G. A Formal Specification Language for Agent Con-versations, LNAI, vol. 2691, pp. 214-225, Springer-Verlag: Heidelberg, (2003)
12. DIALOG Project Web Site. Computer Networks & Web Technologies Lab. Available at http://hydra.ls.fi.upm.es/research/conwetlab
13. Plotkin, G.: A structural approach to operational semantics. Technical Report DAIMI FN-19. Aarhus University, Computer Science Department, Denmark (1981)
14. Hennessy, M.: The Semantics of Programming Languages: An Introduction Using Structured Operational Semantics. Wiley (1990)
15. R. van Eijk, F. de Boer, W. van der Hoek and J-Ch. Meyer: Operational Semantics for Agent Communication Languages. In F. Dignum and M. Greaves (eds.) Issues in Agent Communication, LNCS 1916, 80-95, Springer, Heidelberg (2000)
16. Koning J.; and Oudeyer, P. Introduction to POS: Protocol Operational Semantics. International Journal of Cooperative Information Systems, 10(2):101–123 (2001)
17. Haddadi, A.: Communication and Cooperation in Agent Systems: A Pragmatic Theory. volume 1056 of LNCS. Springer Verlag, Heidelberg, Germany (1996)

Agents' Bidding Strategies in a Combinatorial Auction

Tim Stockheim[1], Michael Schwind[2], and Oleg Gujo[2]

[1] Business Information Systems and Operations Research, Technical University Kaiserslautern, Gottlieb-Daimler-Strasse 47, D-67663 Kaiserslautern, Germany
stockheim@wiwi.uni-kl.de
[2] Institute of Information Systems, Johann Wolfgang Goethe University, Mertonstrasse 17, D-60054 Frankfurt, Germany
schwind, gujo@is-frankfurt.de

Abstract. This paper presents an *agent-based simulation environment* for task scheduling in a grid. Resource allocation is performed by an *iterative combinatorial auction* in which *proxy-bidding agents* try to acquire their desired resource allocation profiles. To achieve an efficient bidding process, the auctioneer provides the bidding agents with approximated shadow prices from a linear programming formulation. The objective of this paper is to identify optimal bidding strategies in multi-agent settings with respect to varying preferences in terms of resource quantity and waiting time until bid acceptance. On the basis of a utility function we characterize two types of agents: a *quantity maximizing agent* with a low preference for fast bid acceptance and an *impatient bidding agent* with a high valuation of fast allocation of the requested resources. Bidding strategies with varying initial bid pricing and different price increments are evaluated. Quantity maximizing agents should submit initial bids with low and slowly increasing prices, whereas impatient agents should start slightly below market prices and avoid 'overbidding'.

1 Introduction

We present an agent-based simulation environment for resource allocation in a distributed computer system that employs a combinatorial task scheduler. Our environment enables the simulation of a mechanism for the simultaneous allocation of resources in the distributed computer system. In contrast to traditional grid allocation approaches, our allocation process considers production complementarities and substitutionalities for these resources thus raising the efficiency level of the resulting resource [1]. The central scheduling instance of our system is comparable to an auctioneer that performs an iterative *combinatorial auction (CA)*. Agents try to acquire the resources required in computational tasks for the provisioning of *information services and information production (ISIP)* by submitting package bids. We introduce a utility function that will allow us to represent different preferences of agents, i.e. a trade-off between quantity maximization and fast acceptance of bids. In a first setting we identify the strategy

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 37–48, 2006.

that maximizes the utility of four homogeneous, quantity maximizing bidders. Subsequently, we introduce an additional (competitive-bidding) agent that requires the requested resources as soon as possible. The earlier this agent receives the acceptance the higher is its gained utility. We compare the bidding strategies under changing resource capacity situations with respect to their allocation efficiency. Finally, we interpret the outcome in terms of gained utility for the various bidding strategies.

2 Combinatorial Auctions for Resource Allocation in Distributed Computer Systems

Various auction protocols have been proposed for resource allocation in distributed computer systems. This may derive from the fact that auctions have been thoroughly investigated by economists and have proved to be efficient allocation mechanisms [2]. The transfer of economic principles to resource attribution in grid systems, such as *price controlled resource allocation (PCRA)*, allows the *flexible* implementation of control mechanisms in decentralized systems [3,4].

CAs are a suitable tool to allocate interdependent resources according to the willingness-to-pay *(W2P)* of the participants. The production process for information services in distributed systems comprises an allocation problem with strong complementarities. An example of such an information service is the provisioning of a video conference service via the web or the off-line calculation of distributed database jobs that have to be processed on different computers and acquires CPU time. Without obtaining communication network capacity between the computers, the acquired CPU time is useless. The application of CAs for resource allocation in distributed computer systems is still in its infancy despite its excellent applicability to grid computing. In a recent approach, Chun et al. [5] present a CA based mechanism for resource allocation in a *SensorNet* testbed where the devices have different capabilities in various combinations. The periodically performed combinatorial sealed-bid auction is implemented within the *microeconomic resource allocation system (MIRAGE)*. The system uses a very simple combinatorial allocation mechanism to achieve sufficient real time performance. MIRAGE users have accounts based on a virtual currency enabling a bartering process for the SensorNet resources. A continuation of this work is the grid computing environment *Bellagio* [6]. The approach relies on Berkeley's CA based allocation scheme *SHARE* [7]. Each bidder has a budget of a virtual currency available for task payment purposes. The required resources are allocated to the particular tasks through a combinatorial second-price auction, which can be regarded as a strategy proof mechanism [7]. In several experiments, the system is tested with respect to scalability, efficiency, and fairness. A simple greedy algorithm guarantees the system's scalability, however, the resulting allocation does not reveal a satisfactory efficiency level. Most recent approaches in "Grid Economics" use double sided CAs for the exchange of resources [8]. However, there is no CA grid system that makes use of proxy-bidding agents to autonomously procure the resources required for ISIP provision.

3 An Agent-Based Simulation Environment for Combinatorial Resource Allocation

The presented CA environment is based on the JADE 3.3 agent workbench and goes beyond the recent research approaches in several points:

- The system allows the usage of several winner determination algorithms such as greedy, simulated annealing, genetic programming, and integer programming methods according to the users' requirements in terms of allocation quality and computation time.[1]
- The simulator provides tools to investigate various bidding behaviors of the proxy agents in the resources acquisition process.
- The framework can simulate changing resource capacities to test allocation efficiency and system stability.
- The ontology-based bidding protocol opens the system to additional agents, e.g. to test strategies based on machine learning or reasoning.

The results presented in this paper concentrate on the second aspect.

3.1 Scenario for a Price Controlled Resource Allocation

This section gives a brief overview on the *resource allocation scenario* for ISIP provision used in our work. The scenario includes four resource types: *Central processing units (CPU)* that are required for the processing of the data, *volatile memory capacity (MEM)* which is necessary to store short-term processing data and program codes for the central processing units, *non-volatile storage capacity (DSK)* which is necessary to keep mass data on databases, and *network bandwidth (NET)* that is required for data interchange between different computer units.[2]

The task agents submit several bids as exclusively eligible bundles *(OR-of-XOR)*. For the formal representation of the bids, a two-dimensional *bid-matrix (BM)* is used. One dimension describes the time $t \in \{1, \ldots, T\}$ at which the resource is required within the request period. The other dimension $r \in \{1, \ldots, R\}$ denotes the resource types MEM, CPU, NET, and DSK. The request for a certain quantity of an individual resource r at time t is then denoted by a matrix element $q_{i,j}(r,t)$. A price $p_{i,j}$ is assigned to each BM expressing the agent's W2P for the resource bundle. In both cases the corresponding bid bundle is identified by the index i and the single BM by j.

The value q^{bmax} denotes the maximum resource load that can be requested by a bidder for a single BM element $q_{i,j}(r,t)$. These elements are occupied with *time slot occupation probability* wk^{tso}. The value q^{bmax} denotes the maximum

[1] Schwind et al. provides a description of the algorithms [9].

[2] Network connections themselves exhibit complementarities due to their peering character. For simplicity we assume that NET capacity can be managed as one single system resource. To consider the individual connections explicitly in our model, they should be treated as additional resources, one for each connection type.

resource load that can be requested by a bidder for a single BM element $q_{i,j}(r,t)$. These elements are occupied with *time slot occupation probability* wk^{tso} and have a maximum allocable resource quantity q^{max}. We use a structured bid matrix, i.e. up to t^{max} time slots are occupied in a row [9].

The agents used within the combinatorial grid simulator comply with three different *roles*. *Task agents* bid for the required resource combination via the mediating agent. This auctioneer receives the resource bids and calculates an allocation profile for the available resources managed by the *resource agents* according to the allocation mechanism. *Resource agents* manage available resources on their particular IT systems and offer them to the task agents via the mediating agent. If the auctioneer accepts a bid, he reserves the resources via the resource agents.

3.2 The Combinatorial Auction

Following the description of the scenario, Figure 1 illustrates the course of action of the system. The AUML sequence diagram depicts the message flow based on the FIPA definition of the English auction protocol.[3]

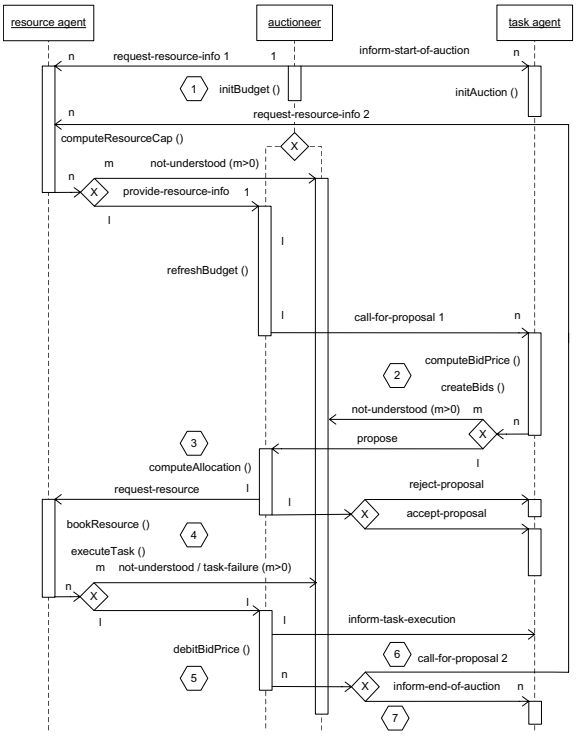


Fig. 1. FIPA AUML diagram for the iterative combinatorial scheduling auction

[3] `www.fipa.org/specifications/fipa00031D`.

While in our closed economy each bidder has two roles (as provider and user of resources), Figure 1 separately depicts both roles (*resource agent* and *task agent*) to provide a higher generalization and better readability. Each step is marked by a ⬡-symbol and detailed in the corresponding paragraph:

1. The auctioneer requests the resource agents to evaluate the available resource capacities and informs the bidders about the bidding terms. He also awards an initial budget to the task agents. Subsequently, he announces the start of the auction.
2. Following the auctioneer's call for proposal, the task agents create their bids according to the desired resource combination.
3. The auctioneer receives the bids and calculates the return-maximizing combinatorial allocation. He informs the task agents about bid acceptance/rejection and requests the resource agents to reserve the awarded resources.
4. Resource agents inform the auctioneer about the status of the task execution.
5. The auctioneer propagates task status information to the task agents and debits the bid price for the awarded bids from their accounts, followed by a call for proposal for the next round.
6. Task agents can renew their bids in the next round in case of non-acceptance or non-execution. The agents' bid pricing follows rules defined in the subsequent paragraph.
7. The process is repeated until the auctioneer announces the end of the auction.

In the following, the three crucial elements of the combinatorial grid scheduling system are described in more detail: the budget management mechanism, the combinatorial auctioneer, and the task agents' bidding behavior.

3.3 The System's Budget Management Mechanism

We assume that the ownership of the resources is distributed among a group of companies and that each proxy-agent represents a company. In order to avoid an expiration of the agents' budgets during the iterative auctioning process, the agents are integrated into a monetary circuit. The system simulates an economy with a constant circulating budget, i.e. a *closed-loop* grid.

Each agent is initialized with a monetary budget BG^{ini}. At the beginning of each round k, the task agents' budgets are refreshed (see Figure 1 - ⬡1) such that each agent is able to acquire *'computational capacity'* proportional to the amount of resources provided to the system. Task and resource agents act as a unit of consumer and producer both owning the resources of their peer system. The resource agent does the reporting of resource usage and provisioning for the task agent owning the peer computer resources (see Figure 1 - ⬡1, and ⬡4). All task agents receive the same budget in each round of the simulation. The accounting of the agents' budgets in the grid system is done by the combinatorial auctioneer (see Figure 1 - ⬡1, and ⬡5).

3.4 The Combinatorial Auctioneer

The combinatorial auctioneer controls the iterative allocation process of the grid system. For this purpose, the auctioneer awaits the bids that have been submitted by the task agents for the current round. The bids that are submitted in the form of j XOR-bundled BMs in bid i and represent the task agents' requested capacity $q_{i,j}(r,t)$ of the resources r at a particular point of time t. After having received all alternative BMs submitted by the task agents, the auctioneer has to solve the *combinatorial auction problem (CAP)* which is NP-hard [10,11]. The CAP is often denoted as the *winner determination problem (WDP)*, according to the traditional auctioneers task of identifying the winner. The formal description of the CAP is formulated as:

$$\begin{aligned} &\max \sum_{i=1}^{I} \sum_{j=1}^{J_i} p_{i,j}\ x_{i,j} \\ \text{s. t.}\quad &q(r,t) = \sum_{i=1}^{I} \sum_{j=1}^{J_i} q_{i,j}(r,t)\ x_{i,j} \leq q^{max}(r,t),\ \forall_{r\in\{1,\ldots,R\},t\in\{1,\ldots,T\}} \quad (1) \\ &\sum_{j=1}^{J_i} x_{i,j} \leq 1, \quad \forall_{i\in\{1,\ldots,I\}}. \end{aligned}$$

The following variables are used: number of resources $R \in \mathbb{N}$, number of time slots: $T \in \mathbb{N}$, number of bid bundles $I \in \mathbb{N}$, number of bids in bundles $J_i \in \mathbb{N}$, W2P of bundle j in bid i as $p_{i,j} \in \mathbb{R}^+$, and the acceptance variable $x_{i,j} \in \{0;1\}$.

The auctioneer's primary goal is to maximize the received income under the limitation of the available resources and a maximum of one accepted bid per XOR bundle (equation 1). In order to accelerate the price-finding process, the auctioneer provides feedback on resource availability to the bidders. As mentioned above, it is not always possible to calculate *unambiguous prices* (anonymous prices) for the individual resources in a CA. In many cases, explicit resource prices can only be calculated for each individual bid. Kwasnica et al. [12] describe a pricing scheme for all individual goods in a CA by approximating the prices in a divisible case based on a linear programming *(LP)* approach first proposed by Rassenti et al. [1]. Like in a similar approach by Bjørndal and Jørnsten [13], they employ the *dual solution* of the relaxed WDP to calculate the shadow prices. In our simulation model, the approach of Kwasnica et al. [12] is adopted:[4]

$$\begin{aligned} &\min\ z = \sum_{r=1}^{R} \sum_{t=1}^{T} q^{max}(r,t) \cdot sp_{r,t} \\ &s.t. \quad \sum_{r=1}^{R} \sum_{t=1}^{T} q_{i,j}(r,t) \cdot sp_{r,t} + (1 - x_{i,j}) \cdot \delta_{i,j} = p_{i,j} \end{aligned} \quad (2)$$

Reduced cost: $\delta_{i,j} \in \mathbb{R}_0^+$
Shadow price of one element: $sp_{r,t} \in \mathbb{R}_0^+$

[4] The result of the following formula is denoted as reduced SPs. Omitting the rejected bids in the calculation of dual prices yields a higher result [13].

The proposed SP calculation uses the *primal solution* for the LP problem delivered from open source LP solver *LPSOLVE 5.5*[5] for the determination of accepted bids. Now the *market value of a resource unit* can be calculated while using the weighted shadow prices and summarizing the utilized capacity of each resource r for all accepted bids as follows:

$$\text{Market value:} \quad v_r = \frac{\sum_{t=1}^{T} sp_{r,t} \cdot q(r,t)}{\sum_{t=1}^{T} q(r,t)} \quad \forall_{r \in \{1,\ldots,R\}} \tag{3}$$

Due to the fact that bid prices are non-linear in this framework, shadow prices $sp_{r,t}$ cannot be calculated in each round, i.e. there is no solution to the LP problem [12]. In such cases the auctioneer relies on an approximation of the market values $\hat{v}_r$ as the averaged market values calculated in the last n rounds.[6]

3.5 The Task Agents' Bidding Model

Except ΔP and P^{ini} all agents show the same behavior. Based on the market values of resources v_r, the task agents of the combinatorial simulation model try to acquire the resources needed for ISIP provision. Besides the market values of resources, their bidding behavior is determined by their budgets and by a *bidding strategy*. In each round, a task agent generates M new bids. In the first round, a market value of the resources is not provided to the bidders. Therefore, bidder agents formulate the W2P for their *initial bids* by dividing the budget by $L \cdot M \cdot J$ to calculate a mean bid price that guarantees the task agents' budget to last for the next L rounds. In the following rounds, if a bid is initialized, the capacities required are multiplied by the corresponding market value v_r and summarized. To control the price adaption process, an additional price acceleration factor P_i^{inc} is introduced.

$$\text{W2P of bundle } j \text{ in bid } i: \quad p_{i,j} = P_i^{inc} \cdot \sum_{r=1}^{R} \sum_{t=1}^{T} v_r \cdot q_{i,j}(r,t) \tag{4}$$

If a bid is rejected, the task agent repeats bidding for this rejected bid in the following rounds. Its W2P is adapted by $P_i^{inc} = P^{ini} + (l_i \cdot \Delta P)$, resulting in the value of P^{ini} in round $l_i = 0$. Rejected bids are *repeated* with an updated W2P up to a maximum of L rounds or until the bid is accepted. When an agent's budget is exhausted, it formulates no new bids until the budget is refreshed.

4 The Preferences of the Task Agents

We will now have a closer look at the bidders' different *preferences*. Two proxy-agent types are used in the context of this paper to represent these preferences:

[5] `http://www.geocities.com/lpsolve/`

[6] If the market value is approximated, the value $\hat{v}_r$ is not saved in the history.

- A *quantity maximizer* that requires high resource capacities but has weak preferences regarding the timing. However, the time of execution and the complementarities of the resources within the bundle have to be satisfied. The hypothesis is that a *smooth bidding strategy*, i.e. to slowly increase the bid prices, maximizes the utility of this agent. The economic rationale for this proxy-agent strategy can be the fact that it bids for resources required for the fulfillment of an ISIP task that is not time-critical. An example of this is the generation of reports based on large databases on a distributed system that have to be done in a relaxed time window.
- An *impatient bidder* that benefits from the possibility to instantaneously use the resources will apply an *aggressive bidding strategy* to maximize his utility. This agent has to submit high initial prices, but overpaying will reduce the quantity he can acquire. We analyze whether a fast inclining pricing strategy combined with lower initial bids can help to further increase the utility of this agent. The economic motivation of this utility function can be a proxy agent that bids for the execution of time-critical tasks. A good example of this is the performance of a video conference in the distributed computer system which is scheduled for a narrow time window.

Clearly, the amount of acquired ISIP resources has a positive but diminishing marginal impact on the agent's utility. The strength of this impact will be defined by α. Opposing the positive impact of the amount of acquired resources, the number of periods an agent has to wait before its bids are accepted has a negative impact. To calculate the decreasing impact of the waiting time, we use a time index $\bar{l_a}$ which is defined as the averaged number of periods an agent bids until it has placed a successful bid and β to adjust the influence of the waiting time. The utility of an agent is calculated by the following function:

$$U_a = \frac{\left(\sum_{(i,j)\in B_a} x_{i,j} \cdot \sum_{r=1}^{R} \sum_{t=1}^{T} q_{i,j}(r,t)\right)^{\alpha}}{\left(\bar{l_a}\right)^{\beta}} \tag{5}$$

Utility function of agent a: $U_a \in \mathbb{R}^+$
Bids of agent a: $B_a \in \{(i,j) \mid i,j \in \mathbb{N}\}$

Using the utility function the two different agent types are: (1) the *quantity maximizer* with $\alpha = 0.5$ and $\beta = 0.01$ and (2) the *impatient bidder* with $\alpha = 0.5$ and $\beta = 1.0$.

5 Results

The primary objective of the experiments is to find out the test agent's optimal bidding strategy in competition with the remaining default bidding agents, given the two types of utility function (quantity maximizer, impatient bidder) as defined above.

In all simulations an identical basic setting is used: Beginning with one bundle containing three XOR-bids in the first round, four agents generated three additional bid bundles for each further round k. The task agents increase the W2P of rejected bids by Δp over a maximum of $L = 5$ rounds. The pattern of newly generated BMs is defined by $q^{bmax} = 3$, $wk^{tso} = 0.333$, and $t^{max} = 4$. The auctioneer was able to allocate a maximum load of $q_{max} = 8$ per resource while T was 8 units for the CM. For the evaluation of our model, we set the number of bids per agent (M) to 3, which for four agents results in $I = 12$ bids per round. Resource 1 is reduced to an amount of 4 units in the 25th round.

In a first setting, all agents use a *default bidding* behavior with a constant value of $\Delta P = 0.2$. This supports the price adaptation process in case of resource failures. Figure 2 shows the results of 50 simulations for varying values of $P^{ini} = 0.3 \ldots 1.0$ to identify the optimal bid introduction level.

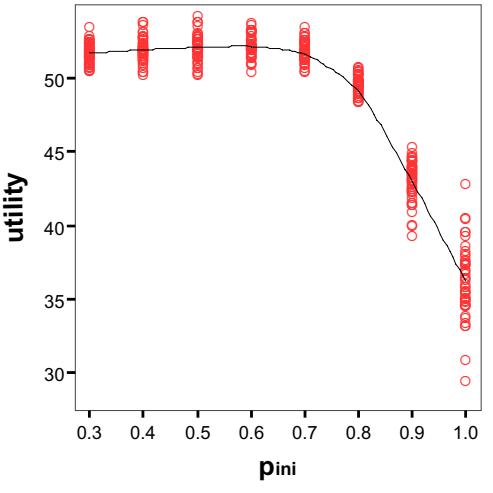


Fig. 2. Averaged utility values of the four homogeneous bidders with varying initial price P^{ini} and static price increment $\Delta p = 0.2$

Assuming quantity maximizing agents, it turns out that setting $P^{ini} = 0.6$ maximizes the average utility of the agents ($\bar{U}_a = 52.21$, capacity = 2762.78, time index = 1.96). The stochasticity of the bidding process in connection with the combinatorial complexity of the CAP leads to an occasional acceptance of low-priced bids. Therefore, a bidding strategy that tries to procure resources below market prices turns out to be successful. In contrary, agents that initialize bids at the market price (which is an averaged value) risk to overpay the required resources. The cumulated capacity used by the agents was 11,051 units, which translates in a utilization rate of 73.7 % (max. capacity: 15,000). While in round 1 to 25 the rate was 82.5 %, it dropped to 67.2 % percent in round 26 to 50 (after reducing the capacity of resource 1 from 8 to 4 units per time slot).

The next setting introduces a competitive bidder that differs in his strategy from the other agents. In compliance with the results from the first setting, we set the default bidding strategy to $P_{ini} = 0.6$ and $\Delta P = 0.2$. Figure 3 shows the resource units acquired by the test agent and the averaged bid acceptance time of 50 simulation runs for each ΔP, Pp^{ini} combination (steps of 0.1).

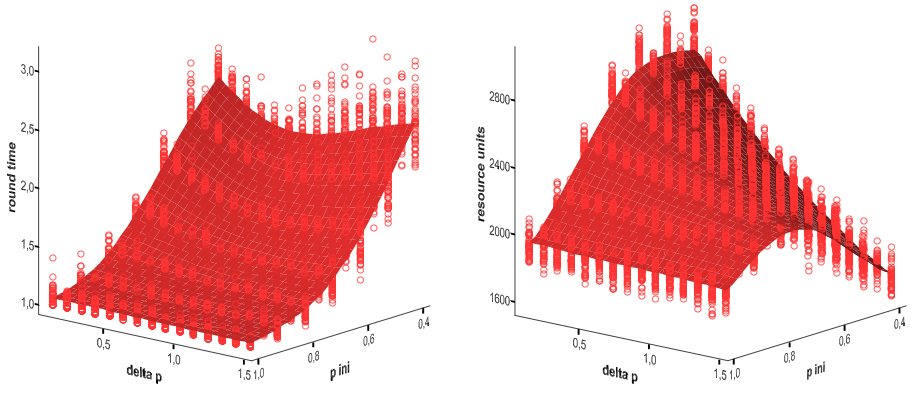


Fig. 3. Mean acceptance time and quantity of resource units for competitive bidder with varying price increment Δp and initial price P^{ini}

In case of small ΔP and P^{ini}, the highest amount of resource units can be acquired by the task (test) agents. An (too) aggressive strategy with high ΔP and P^{ini} leads to a declining amount of acquired resources. While a reduction of acceptance time is mainly achieved by high P^{ini}, increasing ΔP has only impact on average acceptance time if P^{ini} is low.

In Figure 5, the utility (cp. equation 5) resulting from varying price increment ΔP and initial pricing P^{ini} is depicted. It pays off for the quantity maximizer to wait if his bids fit into the current allocation at a relative low price (low increment and initial price). In contrary, the impatient bidder gains low utility from such a strategy (Fig. 5 right side). The impatient agent receives the highest utilities by using an initial bid price close to the market value of the resources ($p^{ini} = 0.9$). Interestingly, the price increment in the following round does not have much impact on the acceptance time and therewith on the utility of the impatient test bidder.[7] However, for bids exactly at market value utility declines sharply, signaling the peril of *'overbidding'* or simply paying too much for the required ISIP resources. This underlines the importance of accurate market value information to achieve allocations that maximize the benefits of the bidders. The shadow price-controlled combinatorial grid enables agents to implement efficient

[7] If the impatient bidder follows its optimal strategy ($p_{ini} = 0.9$ - default value of $\Delta p = 0.2$), the cumulated capacity used by the agents was 11,051 units, which translates in a utilization rate of 65.0 % (max. capacity: 15,000). While in round 1 to 25 the rate was 74.34 %, it dropped to 54.31 % in round 26 to 50 (after reducing the capacity of resource 1 from 8 to 4 units per time slot.

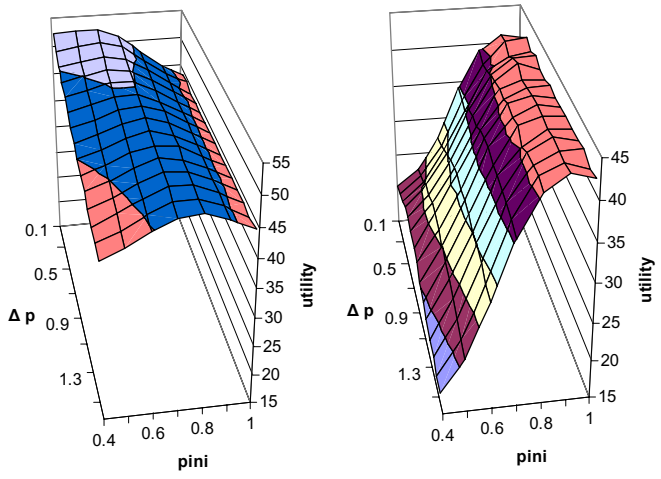


Fig. 4. Utility of the test bidder for quantity maximizing preference $\beta = 0.01$ (left) and impatient bidding behavior $\beta = 1.0$ (right) for determination of the optimal bidding strategy under varying price increment Δp and initial pricing behavior P^{ini}

bidding strategies according to the user's utility functions. Clearly, a high impact of the competitors' behavior remains as challenge.

6 Conclusion

This paper presents an agent-based simulation environment that enables the simultaneous allocation of resources in a grid-like computer system. In this economically inspired approach where proxy-agents try to acquire optimal resource bundles with respect to limited budgets, the allocation is done by a CA. The auctioneer provides price information that is calculated as shadow prices in connection with solving the $\mathcal{NP}$-hard winner determination problem by an integer programing approach. Based on these settings, bidding strategies are evaluated with respect to utility functions that incorporate different levels of time preferences of the bidders. We introduce two characteristic bidders: A *quantity maximizing agent* with low preference for fast bid acceptance and an *impatient bidding agent* with a high valuation of fast allocation of the requested resources. While searching the strategy space by varying the bidding behaviors in terms of initial bid price and price increment strategy for rejected bids, we identified optimal bidding strategies in terms of achieved utility. For the quantity maximizing agent, patience at low initial bids pays off, whereas the impatient agent should avoid 'overbidding'.

Although we used a small number of agents, our simulations turned out to be very time consuming. A way to reduce the volatility of our market prices is the use of a larger population of agents. Therefore, a future objective is to improve the allocation algorithm and analyze heuristic approaches in order to reliable

handle larger settings. A second objective might be the introduction of agents that learn their optimal strategy to find a stable equilibrium.

References

1. Rassenti, J.S., Smith, V.L., Bulfin, R.L.: A combinatorial auction mechanism for airport time slot allocation. The Bell Journal of Economics **13**(2) (1982) 402–417
2. Milgrom, P.: Putting Auction Theory to Work. Cambridge University Press (2004)
3. Buyya, R., Stockinger, H., Giddy, J., Abramson, D.: Economic models for management of resources in peer-to-peer and grid computing. In: Proceedings of the SPIE International Conference on Commercial Applications for High-Performance Computing, Denver, USA (2001)
4. Foster, I., Jennings, N.R., Kesselman, C.: Brain meets brawn: Why grid and agents need each other. In: Proceedings of the 3rd Int. Conference on Autonomous Agents and Multi-Agent Systems (AAMAS04), New York, NY (2004) 8–15
5. Chun, B.N., Buonadonna, P., AuYoung, A., Ng, C., Parkes, D.C., Shneiderman, J., Snoeren, A.C., Vahdat, A.: Mirage: A microeconomic resource allocation system for sensornet testbeds. In: Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II); Sidney, Australia. (2004)
6. AuYoung, A., Chun, B.N., Snoeren, A.C., Vahdat, A.: Resource allocation in federated distributed computing infrastructures. In: Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure, San Francisco, USA. (2004)
7. Ng, C., Parkes, D.C., Seltzer, M.: Virtual worlds: Fast and strategyproof auctions for dynamic resource allocation. In: Proceedings of the third ACM Conference on Electronic Commerce (EC-2003), San Diego, CA, ACM (2003) 238–239
8. Schnizler, B., Neumann, D., Veit, D., Weinhardt, C.: A multiattribute combinatorial exchange for trading grid resources. In: Proceedings of the 12th Research Symposium on Emerging Electronic Markets (RSEEM), Amsterdam, Netherlands, 2005. (2005)
9. Schwind, M., Stockheim, T., Rothlauf, F.: Optimization heuristics for the combinatorial auction problem. In: Proceedings of the Congress on Evolutionary Computation CEC 2003. (2003) 1588–1595
10. Parkes, D.C., Ungar, L.H.: Iterative combinatorial auctions: Theory and practice. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00). (2000) 74–81
11. Fujishima, Y., Leyton-Brown, K., Shoham, Y.: Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence 1999 (IJCAI-99), Stockholm, Sweden. (1999) 548 – 553
12. Kwasnica, A.M., Ledyard, J., Porter, D., DeMartini, C.: A new and improved design for multi-objective iterative auctions. Management Science **51**(3) (2005) 419–434
13. Bjørndal, M., Jørnsten, K.: An analysis of a combinatorial auction. Technical Report 2001-11, Department of Finance and Management Science, Norwegian School of Economics and Business Administration, Bergen, Norway (2001)

Modeling and Simulation of Tests for Agents

Martina Gierke, Jan Himmelspach, Mathias Röhl, and Adelinde M. Uhrmacher

University of Rostock
Institute of Computer Science
Albert-Einstein-Str. 21, D-18059 Rostock, Germany
{mroehl, gie, jh194, lin}@informatik.uni-rostock.de

Abstract. Software systems that are intended to work autonomously in complex, dynamic environments should undergo extensive testing. Model-based testing advocates the use of purpose-driven abstractions for designing appropriate tests. The type of the software, the objective of testing, and the stage of the development process influence the suitability of tests. Simulation techniques based on formal modeling concepts can make these abstractions explicit and operational. A simulation model is presented that facilitates testing of autonomous software within dynamic environments in a flexible manner. The approach is illustrated based on the application Autominder.

1 Introduction

Developing agents essentially means developing software that is able to successfully accomplish specified tasks in an environment that changes over time. This process is of an intrinsically experimental and exploratory nature. Little work has been done on developing methods for testing agents so far [1]. Current methodologies and tools that support agent design and implementation leave a gap between specifications and implementations [2].

In the following we will give a short overview about testing of agents. A discrete-event approach constitutes the basis for developing an experimental frame for testing agents. Various models with different roles in testing will be defined. Among those the interface between the software under test and the simulation system plays a central role and will be discussed in more detail. The setup of a concrete experimental frame will be illustrated on the application Autominder. The paper concludes with a discussion of related work.

2 Testing Agents

Different strategies for designing tests are used. Typically, black box and white box testing strategies are distinguished [3]. Whereas black box testing is concerned with examining the observable behavior of an implementation and focuses on functional requirements, white box testing makes use of knowledge about the internal structure of a program and how results are achieved.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 49–60, 2006.

Furthermore, a software can be tested on different levels and during different phases of development, which necessitates the use of different test design methods. Unit testing is usually associated with white box test design, because units are often of limited (structural) complexity. Testing of whole agent systems has to cope with increased complexity due to multitasking, nondeterminism and timing issues. Consequently, system testing focuses on the discovery of bugs that emerge from interactions of components or result from contextual conditions [4].

In the following we will focus on supporting testing of agents at the system's level.

2.1 Model-Based Testing of Agents

Whereas most of the testing approaches directly derive test cases from the specification of requirements, model-based testing uses additional models for test case selection. To constrain the set of test cases, model-based testing introduces a model of the test scenario to distinguish significant ones.

Types of abstraction that are used in model-based testing are: *functional, data, communicational*, and *temporal* abstraction [5]. Functional abstraction omits behavioral details that are of no interest according to the current test purpose, i.e. behavior is tested only with respect to a constrained environment. Data abstraction maps concrete data types to abstract ones. Communicational abstraction maps sets of interactions to atomic ones. Temporal abstraction moderates precise timing of events, e.g. to consider only the order in which events occur.

The complexity of an agent's environment makes functional abstraction mandatory for testing. Covering a certain set of test cases is hindered by the agent's autonomy. Because of its unpredictable behavior it is difficult to drive an agent into a certain test case [6]. Test cases should not be completely predefined but evolve dynamically depending on both the environment and the reactions of the agent itself [7]. In contrast to non-deterministic testing, environmental models provide a controlled selection of test cases.

3 Modeling Test Architectures

We suggest a rather radical interpretation of "model-based testing" by explicitly defining all abstractions in models. Therefore, we introduce a general model of the test architecture.

A test architecture must be easily adaptable to provide the required granularity and to complement the software under test as far as it has been developed. Thus, the chosen formalism and the test architecture should support modularity, re-use, and refinement. In particular, to be applicable for agent testing, the architecture should support multiple facets of testing and variable structures. Moreover, adequate testing of real-time constraints requires a formalism on a dense time base.

The approach presented here is based on James II, a component-based modeling and simulation system, which supports different modeling formalisms and facilitates different simulation engines [8]. Most of the realized modeling formalisms are based on DEVS [9] and extend the formalism by introducing variable structures [10] or peripheral ports into DEVS.

DEVS distinguishes between atomic and coupled models. An atomic model is described by a state set S, a set of input and output ports X and Y respectively, an internal and external transition function, δ_{int} resp. δ_{ext}, an output function λ, and a time advance function ta. δ_{int} dictates state transitions due to internal events, the time of which is determined by the ta function. At an internal event, the model produces an output in λ. δ_{ext} is triggered by external inputs that might arrive at any time.

Coupled DEVS models support the hierarchical, modular construction of models. A coupled model is described by the set of its component models, which may be atomic or coupled, and by the couplings that exist among them. Coupled models enable the structuring of large models into smaller ones.

3.1 Equipping DEVS with Peripheral Ports

To support the integration of externally running software, DEVS models have been equipped with peripheral ports (XP, YP).

The classical ports (X, Y) of DEVS models collect and offer events that are produced by models. In addition, the peripheral ports allow models to communicate with processes that are external to the simulation. Thereby, the simulation system does not interact with external agents as one black box, but each single model can function as an interface to external processes. The model functions were extended to handle input and output from respectively to external processes. The state transition functions and the $\lambda : S \times XP \rightarrow Y$-function of the interface model describe how incoming data is transformed into data that can be used within the simulation. The functions $\delta_{int} : S \times XP \rightarrow S \times YP$ and $\delta_{ext} : Q \times X \times XP \rightarrow S \times YP$ are furthermore responsible for the transformation of simulation data into data usable by the externally running software.

The simulation system uses the time model function $tm : \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ to translate the resource consumption of the externally running software into simulation time:

External processes are invoked by the simulation system and the information put into the peripheral output ports is forwarded to them. After the external computation has finished, the results of these invocations arrive at the peripheral input ports of the according model at a simulation time which is determined by the time model.

3.2 An Experimental Frame for Testing Agents

Models are designed, tested, and validated based on conditions and assumptions. The so called *experimental frame* [11] makes those explicit. Figure 1 depicts the components that make up an experimental frame for testing agents.

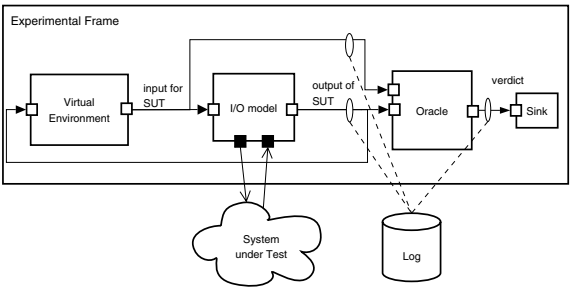


Fig. 1. Model of a test architecture for agents

I/O models form the interface between simulation and Software under test (SUT). It exchanges data between simulation and the externally running software via its peripheral ports. The state transition functions and the λ-function of the interface model describe how incoming data from the SUT is mapped to data that can be used within the simulation and vice versa. Thereby they describe the data abstraction in testing.

Moreover, *I/O* models also describe communicational and temporal abstraction. Using the time model $tm : \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ different types of temporal abstraction can be realized considering the progression of wall clock time and consumption of resources on the one hand and simulation time on the other hand. However, for testing timing requirements the validity of the chosen time model is crucial. Often the consumed wall clock time is used as a resource, as it is easily accessible. Unfortunately, its usage endangers the repeatability of simulation runs and implicitly introduces uncertainties due to hardware configuration and current work load. Better suited time models depend on the type of implementation, the language used, or the underlying operating system [12].

In all cases, an *ExternalProcessThread* has to be implemented per *I/O* model that realizes the actual communication between simulation and the SUT. If the implementation does not possess an own clock, a synchronous interaction of simulation and the external software will prove beneficial, as it gives the simulation system full control over the experiment.

The Virtual Environment models the circumstances under which the behavior of the SUT (i.e. the agent) is to be observed. Thus, it realizes the functional abstraction in testing and constrains the view on the possible behavior of the system. Within the virtual environment the peripheral ports can also be exploited for integrating user interactions, e.g. humans can interact asynchronously with the simulation supported by peripheral ports and a paced execution. Gradual transitions between explicit models and "ad-hoc" testing by humans become possible.

So far, we have merely formalized the experimental set-up for testing agents. Now, we shall see how this general architecture was deployed for testing a concrete agent application and how the general framework can be filled with application specific abstractions.

4 Testing Autominder

Autominder is a distributed monitoring and reminder system developed at the University of Michigan [13]. It is being designed to support older adults with mild to moderate cognitive impairment in their activities of daily living. These are activities a person must complete in order to ensure her physical well-being (e.g. eating, toileting). The software is installed in the client's house together with a set of sensors and given a list of actions that must be performed during the day. At runtime, Autominder evaluates sensor echoes from the environment, reasons about ongoing user activity, and compares its findings to the given client plan to detect forgotten actions and remind the user of their execution.

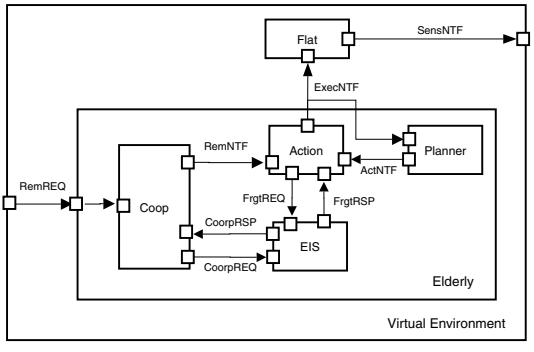


Fig. 2. Refined Virtual Environment for Testing Autominder

4.1 The Virtual Environment

Autominder complements a human in-place nurse and thus has a huge responsibility for its client's safety. Consequently, it must be tested in a variety of application scenarios before its release. Since those scenarios are not always safe for a human client and may be hard to observe during human-in-the-loop field tests, a virtual testing environment for Autominder (AVTE) has been developed [14] in cooperation with the Autominder research group. The AVTE was implemented in JAMES II. Its layout is shown in Figure 2. The testing environment focuses on the model of an elderly who acts as the Autominder user. Besides, it includes a model of the client's living environment where Autominder is supposed to work.

The Elderly model is designed as a Human Behavior Representation that represents an elderly person in terms of actions and mental state [15]. Therefore, it provides two basic client functionalities: emulate suitable elderly behavior and react to incoming reminders. The virtual elderly follows a certain routine and performs actions of daily living accordingly. Depending on its initial memory fitness and current stress level, the Elderly model may forget some of the plan actions. If the model receives an external reminder, it interrupts this default behavior and evaluates the reminder. However, there is no guarantee that the

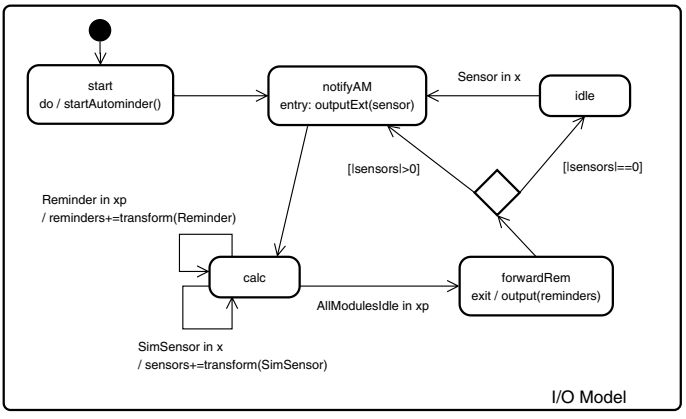


Fig. 3. An I/O model that functions as an ambassador for Autominder

Elderly model will cooperate with Autominder and execute the requested action. Depending on its cooperativeness and current degree of annoyance, the virtual elderly may choose to ignore reminders as well.

In the Elderly model, each of the basic functionalities is implemented by a separate component: the *Cooperation* model deals with reminder acceptance, and the *Action* model is responsible for the execution of actions. It receives information on currently executable actions from the *Planner* model that manages the routine plan of the elderly. The *ElderlyInformationStorage* (EIS) model holds all user-specific information necessary to compute the elderly's present forgetfulness and cooperativeness depending on the past workload.

The *Flat* model composes a sensory picture of the elderly's surroundings. It generates sensor data that reflect the current in-house activity based on the past action history and knowledge about how sensors are activated by plan steps. Examples for such sensors are contact sensors for doors and the medication container cover, heat sensors for the stove, and flush sensors in the bathroom.

4.2 Coupling Autominder and Simulation

For testing Autominder, the software has to be plugged into the virtual environment. Autominder is a complex and autonomous implementation. It offers an interface for external function invocation and is able to proactively start interactions itself.

To support both the testing of method invokation as well as the testing of agent software with an own thread of control, ambassador models have been introduced [16], which form a special kind of I/O models. With ambassador models a representative of the externally running software becomes part of the virtual environment, crucial phases of the software are directly reflected within the simulation model.

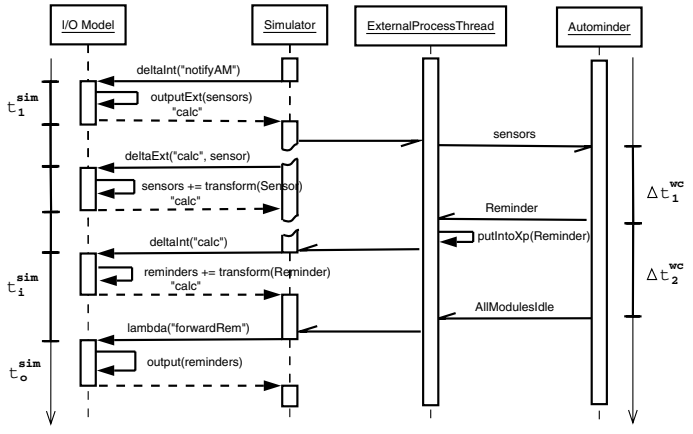


Fig. 4. Interaction between the ambassador I/O model and Autominder

Using ambassadors, the SUT typically needs to be instrumented, i.e. methods of the agent have to be changed to redirect messages into the simulation system. In the case of Autominder, its central module, the Event-Handler, was concerned. Every software module that wants to receive Autominder messages needs to be registered at the EventHandler, and so does the Ambassador model. Furthermore, some new message types were introduced. On the one hand, Autominder has to accept new simulation-generated synchronization messages in order to agree on a common time. On the other hand, marker messages must now be issued by the EventHandler so that the Ambassador model could clearly tell apart Autominder's different processing phases.

Figure 3 shows an ambassador *I/O* model for testing Autominder. This model reacts to incoming simulation messages by forwarding them to Autominder and reacts to incoming messages from Autominder by passing them to the other models of the virtual environment.

Communicational abstraction is used for collecting reminders and sensor notifications during phase *calc*. All reminders that have occurred during a computation cycle of Autominder are passed to the virtual environment jointly. Similarly, all sensor notifications issued by the virtual environment during the current Autominder processing cycle are gathered and held back until the transition to the phase *notifyAM* takes place.

Figure 4 shows a typical interaction between the ambassador *I/O* model and Autominder. Calculation phases of Autominder are initiated by the simulation system. Sensor information from the virtual environment is put into the peripheral output ports by the *I/O* model and forwarded to Autominder. Reminders produced by Autominder are put into the peripheral input port of the *I/O* model and saved until an *AllModuleIdle* message indicates that Autominder has finished a computation cycle. The simulation time of peripheral input events is determined by the time model, i.e. $t_i^{sim} = t_1^{sim} + tm(\Delta t_1^{wc})$, $t_o^{sim} = t_i^{sim} + tm(\Delta t_2^{wc})$. This is the time at which the reminders are scheduled in the virtual environment.

Autominder maintains an own clock, which is advanced either by Autominder itself or externally. Using ambassador models, Autominder time is controlled externally. Autominder advances its clock according to the time stamp of the simulation-generated messages it receives.

4.3 The Oracle

The oracle module judges, whether Autominder adequately reacts (issues reminders) with respect to formerly received inputs from the virtual environment. Adequacy is defined in terms of requirements that are specified at an earlier design phase. For testing Autominder's reminder generation feature, four requirements were identified in order to ensure client awareness and avoid annoyance respectively over-reliance:

1. All critical actions from the client plan shall be executed by the client. Therefore, Autominder needs to make sure the client is aware of upcoming activites: it has to issue reminders for actions the client does not perform on its own.
2. However, not all actions mentioned in the client plan may be essential and worth reminding for.
   Autominder must only remind for crucial client plan actions that are marked as "remindable".
3. If the client does not execute a such mandatory, remindable action on its own, it must be given the opportunity to catch up on time. Hence, reminders need to occur within the allowable time bounds of the related action.
4. In order to personalize reminder plans, both client and caregiver can give recommendations on when to remind best (e.g. earliest, latest, or typical execution time). Autominder's reminding policy should respect these preferences[1].

These requirements are translated into a set of rules the oracle uses in order to evaluate the correctness of Autominder's input/output trajectory.

4.4 Test Cases and Results

For each test case, input is specified by two plans: the routine plan that defines the behavior of the simulated elderly and the client plan based on which Autominder works. These plans can either agree or differ in their plan steps.

When using the same plans in Autominder and the elderly model, Autominder knows everything its user normally does during a day. Nevertheless, the elderly person may forget some of its usual activities. In this case, Autominder is expected to remind the client of the forgotten actions. In real-world applications, Autominder is likely to be given more sparse information on mandatory client activities only. Future testing scenarios should define the routine plan as a superset of the client plan steps, cf. Figure 5(a) and 5(b).

[1] In our test runs, not much emphasis was put on the adherence to the reminding preferences since "there is no good way to validate [...] [the] optimality [of the client plan]" [17].

action	start interval	end interval
getup	[1,1]	[2,2]
eat	[5,7]	[6,8]

(a) client plan

action	start interval	end interval
getup	[1,1]	[2,2]
wash	[3,3]	[4,4]
cook	[5,5]	[6,6]
eat	[7,7]	[8,8]

(b) superset routine plan

action	start interval	end interval
getup	[1,1]	[2,2]
wash	[3,3]	[4,4]
goOut	[5,5]	[6,6]

(c) differing routine plan

Fig. 5. Example relations of client and routine plan

In our first tests, we used the same plan as routine and client plan. A simple, strictly serial plan was employed. Figure 6 displays the six consecutive plan actions that start resp. end each planned activity and highlights the times when reminders were expected to occur for them. Flashes indicate the actual receipt of a reminder.

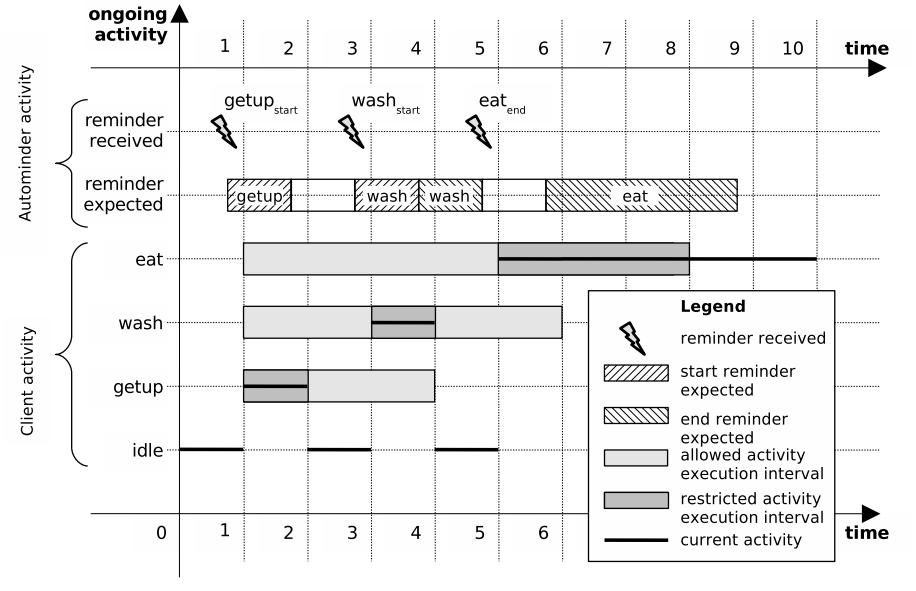


Fig. 6. Summary of simulation runs

In our experiments, reminders always occurred timely. So far, both requirement (1) and (3) are fulfilled. Interestingly, no end reminder was received during the simulation at all. GetUp and Wash both were ended by the model itself, no reminders were necessary. In contrast, the elderly model "forgot" to finish Eating. Here, the omitted end reminder caused problems – the elderly did not comply with its client plan anymore, requirement (1) is violated.

Moreover, this incident also represents an irregularity with requirement (2). Autominder shall remind for crucial actions only, but no reminder is expected for the start of eating. The evidence of a needless start and a missing end reminder nurtured the suspicion that there might be problems with end reminders in Autominder. However, in many safety critical situations ending an activity is an as crucial issue as starting an activity (e.g. turning off stove after cooking). Thus, more extensive testing of end reminders was conducted using an elderly model with a tendency to forget ending activities. The testing revealed that indeed Autominder did not consider end reminders yet. After having been identified, this problem could be resolved easily.

5 Related Work

There exist mature tools for testing real-time constrains by synchronously invoking implemented functions, e.g. TAXYS [18] and Simulink [19]. These works differ from ours in that the present approach also accounts for autonomous agent software.

Testbeds [20], [21] and [22] explored the integration of models rather than implementations of agents that are equipped with an own thread of control and communicate asynchronously with the simulation system.

TTCN-3 is a language for specifying test cases as well as test setups [23]. Whereas TTCN-3 aims to provide a standard notation and execution architecture for all black-box testing needs, we suggest a purpose-oriented modeling of tests and test architectures on an "appropriate" level of abstraction – according to the current development phase and the requirements of a software system.

The abstractions used in model-based testing were already discussed by Hahn *et al.* [6]. In our approach these abstractions became operational by using general purpose modeling and simulation methods.

James II has previously been used for coupling external software to simulation, e.g. synchronous coupling of external software to the simulation system and the use of time models were realized for planning agents [24], mobile agents, [16], and a multi-agent system comprising collaborative agents [12].

6 Conclusion

A modeling and simulation-based approach for testing agent implementations has been proposed. To this end, the idea of experimental frames has been adopted for defining test architectures.

The developed test architecture comprises different sub-models with different roles in the testing process. These sub-model reflect the different types of abstraction usually involved in testing.

Whereas the *Virtual Environment* model signs responsible for the functional abstraction, *I/O* models implement the data and communicational abstraction at least with respect to the communication between SUT and environment. The *Time Model* represents the temporal abstraction underlying the testing.

By using a formal modeling and simulation approach to testing, the mental models that form an intrinsic part of testing are explicitly represented. Furthermore, each of the models can be defined and refined on the level of abstraction required by testing purposes.

The practical use of the methods developed was demonstrated with the Autominder system. In this context particularly the importance of human behavior models became obvious. The integration of human behavior models is gaining importance in other areas, e.g. the evaluation of protocols for mobile ad-hoc networks [25], too. So this might be a general trend when it comes to simulation-based testing of software: to replace ad hoc interactions (of humans in-the-loop) with the software under test, by coherent (although not necessarily validated) and systematically responding user models.

Acknowledgments

This research is supported by the DFG (German Research Foundation).

References

1. Dam, K.H., Winikoff, M.: Comparing agent-oriented methodologies. In: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems, Melbourne (2003)
2. Hilaire, V., Koukam, A., Gruer, P., Müller, J.P.: Formal specification and prototyping of multi-agent systems. In: ESAW 2000. Volume 1972 of Lecture Notes in Artificial Intelligence. Springer Verlag (2000) 114–127
3. Beizer, B.: Software Testing Techniques. 2nd edn. Van Nostrand Reinhold, New York (1990)
4. Bashir, I., Goel, A.L.: Testing Object-Oriented Software: Life Cycle Solutions. Springer (2000)
5. Prenninger, W., Pretschner, A.: Abstractions for model-based testing. In: Proc. Test and Analysis of Component-based Systems (TACoS'04), Barcelona (2004)
6. Hahn, G., Philipps, J., Pretschner, A., Stauner, T.: Prototype-based tests for hybrid reactive systems. In: Proc. 14th IEEE Intl. Workshop on Rapid System Prototyping (RSP'03), IEEE Computer Society (2003) 78–85
7. Kopetz, H.: Software engineering for real-time: a roadmap. In: ICSE - Future of SE Track, ACM Press (2000) 201–211
8. Himmelspach, J., Uhrmacher, A.M.: A component-based simulation layer for JAMES. In: Proc. of the 18th Workshop on Parallel and Distributed Simulation (PADS), May 16-19, 2004, Kufstein, Austria. (2004) 115–122
9. Zeigler, B.P., Praehofer, H., Kim, T.G.: Theory of Modeling and Simulation. 2nd edn. Academic Press, London (2000)
10. Uhrmacher, A.M.: Dynamic Structures in Modeling and Simulation - a Reflective Approach. ACM Transactions on Modeling and Simulation **11**(2) (2001) 206–232
11. Zeigler, B.P.: Multifacetted Modelling and Discrete Event Simulation. Academic Press, London (1984)

12. Röhl, M., Uhrmacher, A.M.: Controlled experimentation with agents – models and implementations. In Gleizes, M.P., Omicini, A., Zambonelli, F., eds.: Post-Proc. of the 5th Workshop on Engineering Societies in the Agents World. Volume 3451 of Lecture Notes in Artificial Intelligence., Springer Verlag (2005) 292–304
13. Pollack, M.E., Brown, L., Colbry, D., McCarthy, C.E., Orosz, C., Peintner, B., Ramakrishnan, S., Tsamardinos, I.: Autominder: An Intelligent Cognitive Orthotic System for People with Memory Impairment. Robotics and Autonomous Systems **44** (2003) 273–282
14. Gierke, M.: Coupling Autominder and James. Master's thesis, University of Rostock (2004)
15. Gierke, M., Uhrmacher, A.M.: Modeling Elderly Behavior for Simulation-based Testing of Agent Software. Conceptual Modeling and Simulation CSM 2005 (2005)
16. Uhrmacher, A.M., Röhl, M., Kullick, B.: The role of reflection in simulating and testing agents: An exploration based on the simulation system james. Applied Artificial Intelligence **16**(9-10) (2002) 795–811
17. Rudary, M., Singh, S., Pollack, M.E.: Adaptive Cognitive Orthotics: Combining Reinforcement Learning and Constraint-Based Temporal Reasoning. 21st International Conference on Machine Learning (2004)
18. Sifakis, J., Tripakis, S., Yovine, S.: Building models of real-time systems from application software. Proceedings of the IEEE **91**(1) (2003) 100–111
19. MathWorks: Simulink. `http://www.mathworks.com/products/simulink/` (2005)
20. Pollack, M.E.: Planning in dynamic environments: The DIPART system. In Tate, A., ed.: Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative, AAAI Press, Menlo Park, CA (1996) 218–225
21. Anderson, S.D.: Simulation of multiple time-pressured agents. In: Proc. of the Wintersimulation Conference, WSC'97, Atlanta (1997)
22. Kitano, H., Tadokoro, S.: RoboCup Rescue: A grand challenge for multiagent and intelligent systems. AI Magazine **22**(1) (2001) 39–52
23. Wiles, A.: ETSI testing activities and the use of TTCN-3. Lecture Notes in Computer Science **2078** (2001) 123–128
24. Schattenberg, B., Uhrmacher, A.M.: Planning agents in James. Proceedings of the IEEE **89**(2) (2001) 158–173
25. DIANE-Projekt: Dienste in Ad-Hoc-Netzen. `http://www.ipd.uni-karlsruhe.de/DIANE` (2005)

Agent-Based Simulation Versus Econometrics – from Macro- to Microscopic Approaches in Route Choice Simulation

Gustavo Kuhn Andriotti and Franziska Klügl

Department of Artificial Intelligence, University of Wuerzburg,
Wuerzburg, 97074, Am Hubland, Germany,
{andriotti, kluegl}@informatik.uni-wuerzburg.de

Abstract. Econometrics is nowadays an established approach to the discrete choice problem relying on statistical methods. It is used in several fields, e.g. route choice modelling, telecommunication analysis, etc. Despite its advantages, there are also some drawbacks. Thus, alternatives for modelling human choice are sought, which can reproduce overall system behavior and be valid at microscopic level.

In this paper, we propose an agent-based approach inspired in econometric techniques producing similar results on the macro level from microscopic behavior. This work aims to be a step forward on searching an alternative for econometrics.

1 Motivation

Discrete choice problems have to be solved by humans in several domains and contexts. Examples are route selection for driving to work or choosing a particular shop to go. Thus, it is highly interesting to find an appropriate modelling approach for analysing that decision making process. Econometrics discrete choice models form the currently most used approach for representing such process not only in the traffic domain. However, it is a macroscopic approach based on utility functions and rational decision making.

However using an agent-based approach relying on local information for reproducing particular macroscopic data, is by no way trivial. Due to the local population problem, it may be not possible to reproduce the results of macro models without extensive parameter calibration [1]. The aim of this paper is to present and discuss an agent-based approach that actually is able to produce similar results. This may form the basis for more elaborated agent-based route choice simulations.

The remainder of the paper is structured as follows. We will first introduce the basic concepts for econometric approach to discrete choice problem modelling. After discussing advantages and drawbacks, we introduce and comment an econometric inspired agent-based model. Following that we show a case study, where combined route and model choice decision is taken.Then the results are presented, that came from our agent-based model. By the end are shown a short comparison to related work, conclusion and a short outlook to future work.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 61–72, 2006.

2 Discrete Choice Modelling

Discrete Choice means that participants must chose an option or alternative from a set where the taken option has the best utility for them. In that case, the chosen option of participant n is given by the expression 1, where i_n is the option taken by the n, O is the set of options and $u(i', n)$ is the utility of alternative i' for participant n.

$$i_n = argmax_{i' \in O}(u(i', n)) \tag{1}$$

Usually econometric models aim at describing and modelling the situation from a macroscopic point of view. That is made based on data sampling, i. e., a potentially large number of participants that have to select between options. It also assumes that every participant takes the alternative with maximum utility for him. However, the participants' function $u(i, n)$ is neither completely known nor formally expressible by a modeller. The partial observable knowledge about utility is generalised for all participants and expressed in V_i. It is formalised in equation 3 and further explaind on section 2.1.

To express everything that is unobservable or unknown, an error component ε is introduced. That component actually represents a drawn from a particular statistical distribution, like Normal Distribution. Thus, the general utility expression is given by equation 2. There U_i is the utility of alternative i generalised for all participants. All errors made when generalising from the individual $u(i, n)$ are integrated into ε. The closer U_i is to every $u(i, n)$, the better is the econometric model from a macroscopic point of view.

$$U_i = V_i + \varepsilon \tag{2}$$

$$V_i = \sum_{a \in Attributes} f(\beta_a, a_i) \tag{3}$$

Econometric approaches to the discrete choice problem differ in the way V and ε are characterised and in how the options are organised. Examples are Logit [2], Probit [3] and derived models form Generalised Extreme Value ([4] and [5]). Examples from the latter are Nested-Logit [6] and Mixed-Logit [7]. In case of Logit, the error distribution, or values for ε, is the logistic function and in case of Nested-Logit it is the extreme value distribution.

2.1 Estimating Econometric Models

Usually, the observable utility V_i is computed based on an utility function. That expression combines all attributes of an option and corresponding scale/sensibility factor β. Mostly that expression means summation, like in equation 3.

The utility of a particular option is a function that has as parameters all weights β_a and its corresponding attributes value a for that option i. A simple example for that function would be $f(\beta_a, a_i) = \beta_a \cdot a_i$. Note that β_a depends only on the attribute and not on the alternative. The set of all βs is the called weight set and here referred as B and the attributes' set as A.

For actually building an econometric model, the following problems have to be solved in advance:

- Find the relevant set of attributes A;
- Model the function $f(\beta_a, a_i)$ and
- Chose an appropriate econometric approach, like Nested-Logit.

As result of a that process a probability expression, that depends on B and A, for all alternatives is given. That set of functions express the altenatives' distribution. Actually, the expression $P(i)$ – probability of alternative i – is almost solved but B must be estimated.

The first problem, when using more sofisticated econometrics approaches like Mixed-Logit [7], can emerge. That problem is to solve $P(i)$. For more simple approaches, like Logit and Nested-Logit, that function is already derived but for others it must be derived. The deduction relies not only on the approach itself (Logit, Probit or Mixed-Logit) but also on alternatives organisation and on utility function's nature. And because of that an analytical solution may not be possible.

It was stated that $P(i)$ is a function of B and that the later must be estimated. That means that the B^* must be found. A B^* express the better possible fine-tune of $P(i)$, or the better $P(i)$ can be approximated to participants real decisions. B^* is usually searched using a maximum likelihood estimator. In a very abstract explanation, that estimator works evaluating a particular hypothesised weight set B_h. Then a quality measure is calculated with a *likelihood* or $log - likelihood$ function. They compare $P(i)$, using B_h, with the real share. After that, according to its quality – using $log - likelihood$ for instance – a new B_{h+1} is proposed.

Here a very simple example of that process using a Logit[2] model is taken (chosen because of its simplicity). It is also assumed that $V_{ni} = B_h \cdot a_{ni}$, where V_{ni} is the rational utility of alternative i observed for participant n. That posted, a probability expression P_{ni} (from [5]) for every participant n and every alternative i can be derived as shown on equation 4.

$$P_{ni} = \frac{e^{B_h \cdot a_{ni}}}{\sum_j e^{B_h \cdot a_{nj}}} \tag{4}$$

With equation 4 one can calculate the likelihood between the model and real data, using a *likelihood* or $log - likelihood$ function, comparing $P(i)$ with P_{ni}. Then an estimation (using a *Maximum Likelihood Extimator*) must be made on B_h until it achieves B^*, where the *likelihood* can no longer be improved.

2.2 Advantages and Disadvantages

Econometric modelling approaches have several advantages that make them highly attractive for describing discrete choice behaviour. Once the optimal weight set B^* is computed a function is given as result. More than that:

- The weight set B^* forms a compact representation of the relevance of distinct attributes in relation to others. Thus, the model can be easily analysed in this direction.
- If attribute values are changed or the option set is manipulated, the effects on the participants distribution $P(i)$ can be immediately computed and thus predicted. It has been shown in different applications, that this prediction is possible [8]. Of course the preciseness of a prediction depends on how changes affect alternatives.

Once B^* is computed, the model is complete and can be used in an elegant way. However, this elegance has costs the generalisation of agents' utility function. As the macro approach assumes homogeneous utility functions, the weights are the same for all participants. Therefore, it cannot take into account individual evaluation of attributes and options, and thus no individual behaviour.

Every participant is assumed to evaluate and select on its own, independently from the others decisions. Thus, direct relations between decisions are very hard to tackle.

The basic structure of the utility function is mostly very simple, namely a linear combination of weights and attribute values – although the computation of these values can be sophisticated. Theoretically the utility function itself may also contain complicated computations. We assume that the practical use is restricted because the function has to be evaluated very often during the already costly search for B^*.

Moreover, the participants are not seen as capable of adaptation or evolution. That means that, the final model is just able to describe the current state, i.e., the weights of the participants in the given situation. Therefore, the validity of the model depends on the situation's stability. For using the model in a different situation one has to rely on modeller's experience, to adapt the existing model.

3 From Econometrics to Agents

Here, the called agent-based econometric model is a multi-agent system, where agents use a decision making algorithm based on rational utility. The first difference is individualisation, i.e., the model is now microscopic. That means that, all participants are simulated, through its agent representation. Because the agents in the proposed model can adapt themselves, they achieve an "optimum" by an iterative process. Note that in this context, "optimum" means equivalence to original econometric macroscopic model results.

3.1 Agent-Based Econometric Model

In our first approach – devoted to reproducing macroscopic econometrics – we restricted agents' individualism. All values that concern resources' attributes are pre-calculated and all agents share the same values. For example, in traffic, the relevant attribute could be route segment cost, that is a function of segment's

occupancy and capacity. In that case, all agents share the same segment cost function – a generalisation from econometrics.

Moreover, all information about the world is available to all agents. And also agents are neither allowed to communicate with each nor share information directly, i.e., they can just observe the others actions' effects.

The adaptation capability is restricted to weighting resources' attributes and taking an alternative based on that weights. To take a decision and to improve its own profit an agent can fine tune its decision making algorithm but not change it.

Those restrictions are necessary to stay as close as possible to the standard econometric approach for facilitating reproduction. Therefore, agents' weighting or decision making fine tune procedures mimic "optimum's" search, or searching for B^*. That "optimum" is achieved when system's equilibrium is achieved. It happens through the local "greedy" agent's maximisation decision making method.

3.2 Potential and Drawbacks

Some disadvantages from econometrics also appears on agent-based approach. Modelling problem is always present and for agents that means developing an agent's reasoning algorithm. Since the model must be computable it implies into simplifications or restrictions. But agents could lead to a better approximation, i.e., nearer to the real world behaviour.

Another problem is human behaviour modelling and here two characteristics are important: adaptation and evolution capabilities. Those characteristics are relevant to avoid agents remodelling in face of new situations. And due to the fact that the closer to human behaviour the more complex is a model to compute, computational complexity became an issue. By being a multi-agent systems (MAS), all participants are represented and have their own reasoning algorithm. Thus, the computational resources needed will be probably higher than those needed for econometrics. It also could implies in a longer stabilisation process.

The individualisation leads also to a lack of clarity on participants decision making process. With an econometric model the attributes relevance is evident, when not obvious. That can be made by glancing the final utility expression and that is not true for MAS. To evaluate attributes significance a statistical analysis over the agents is needed, if that analysis is possible.

On the other hand there are advantages in using agents. As all participants are present, so the utility function $u(i, n)$ is better approached. That leads to a lesser ε's importance, so a better treatment to the unobservable/unknown utility's part. The unknown behaviour can be partially modelled by a heuristic, like reinforment-learning, which is not possible in econometrics. More than that, agents' heterogeneity can be easily modelled, like different social classes or sex (if that plays a role on the decision making observed behaviour).

But the most significant difference is adaptation and evolution. When correctly modelled, it prolongs model's validity, in case of changes in alternatives' set. Using standard econometrics, a modeller has either to determine by hand

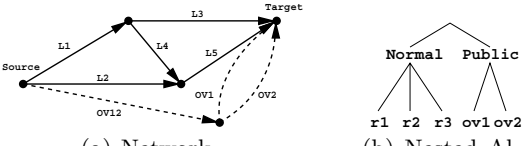
how the distribution changes or re-estimate the complete model, when a new alternative is added or an existing one is removed. Using adaptation, this can be done without external interference or calibration. A user must just wait until the system is stabilised for the new situation. Usually a user's or modeller's direct manipulation on the model is an error prone procedure.

Other advantage from MAS is the absence of a special estimation method or a derived distribution function ($P(i)$). The optimum is achieved by an iterative agents' stabilisation process.

4 Application Example: Combined Route and Mode Choice

To assert that those concepts from section 3.1 are valid, a simple traffic network was adopted. It was originally analysed using econometrics, extracted from Vrtic's PhD thesis [9]. From there, the analysed data was taken for that comparing. The decision to use that example relies on its process' good documentation, simplified structure and availability of results. The problem itself is to identify the network use, or agents' distribution among the avaiable links.

A graphical representation of the network is on figure 1(a) In that network edges represent road segments and nodes crossings. There are special nodes identified on the figure with *Source* and *Target* that represent the origin and destination on that network. Edges with continuous lines represent the segments where private vehicles can circulate and segmented lines where the public transportation takes place.



(a) Network. (b) Nested Alternatives.

Fig. 1. Network Example

The agents' goal is to drive from *Source* to *Target* and 3000 agents were used for that. Equation 5 presents time/cost funtion – used to compute agents' penality (as higher the time as worst perform the agent). On equation 5 $t(s)$ is the time/cost for segment s, $t_0(s)$ a minimum cost, a represents a dependency between occupation and travel time, $q(s)$ is the current occupancy, $L(s)$ capacity and b a scale parameter. For the normal road segments, identified with the letter L, the parameters are $a = 3$ and $b = 8$ and for the public transportation, identified by OV, they are $a = 1$ and $b = 8$. Those parameters and function are extracted from [9] and its validity is not discussed in this text.

$$t(s) = t_0(s) \cdot \left(1 + a \cdot \left(\frac{q(s)}{L(s)}\right)^b\right) \tag{5}$$

The possible routes for that scenario are the following: $R = \{r1, r2, r3, ov1, ov2\}$, $r1 = L1 \rightarrow L3$, $r2 = L2 \rightarrow L5$, $r3 = L1 \rightarrow L4 \rightarrow L5$, $ov1 = OV12 \rightarrow OV1$ and $ov2 = OV12 \rightarrow OV2$.

All routes, identified with r, correspond to routes for normal traffic and with ov for public transportation. That possibilities for route choice determine the transportation mode too. It can be more clear seen on figure 1(b).

4.1 Econometric Macroscopic Model

The results accomplished in [9] are summarised in table 1. Those results were achieved by different econometric approaches. Here it will be not covered details given in [9], but shortly explained procedure for results' generation.

To estimate an econometric model, already taken decisions are needed. But in that artificial scenario a modified method was adopted. There are two main steps, according to [9]: internal equilibrium and external equilibrium. The second is the standard econometric estimation with an end criterion. And the other is a stochastic equilibrium through the occupation's gradient.

In this case, the standard econometric estimation uses a utility equation that is derived from the simplified equation in 6, where U_i is the utility for route i and $t(i)$ the time/cost of route i.

$$U_i = \beta \cdot t(i) + \varepsilon \tag{6}$$

First, an initial segment distribution guess is taken.With that initial estimation an external equilibrium step is made and a new set B_h is disposed. After that, a whole internal equilibrium process takes place. That process finds a local optimum through a stochastic gradient approximation method, it generates a new distribution.

By the end of internal equilibrium, an external equilibrium process is applied. From that process a new set B_{h+1} is achieved and if the gradient between B_{h+1} and B_h, $\nabla(B_t, B_{t+1})$, is less than a certain threshold then B_h is the final estimation and therefore B^*. If that is not the case, another internal equilibrium routine is execute, but now with the values from B_{t+1}.

4.2 Microscopic Agent-Based Econometric Model

Our agent-based econometric model must achieve the same, or almost the same, results as those from econometrics. By following the instructions given on section 3.1 the time/cost function is homogenous and will not undergo adaptation. That leaves freedom just to apply heuristics on route re-evaluation willingness and on route choice algorithms.

To cope with that, the route re-evaluation willingness was modelled as a probability to change the current route. And the route choice algorithm was modelled as a weighted roulette, where the weights are assigned according to a heuristic.

Those two pares are responsible for making agents "greedy" and also providing some stability. The aim si to make agents act egoistic but also to be satisfied with a reasonable profit. Both algorithms are explained on following sub-sections and after that their parameters are discussed.

Route Change Probability. To determine agent's route change probability, the agent base its decision on the past experience with a tolerance. The algorithm is shown in equation 7. There, $P(a)$ is the probability to change route for agent a; P_{min} and P_{max} minimum and maximum probability; $\overline{t_n}$ mean time – based on agents own history – at step n; k a fixed tolerance scale parameter; t_{min} and t_{max} global minimum and maximum time/cost and $t(a)$ current time/cost for agent a.

$$
\begin{aligned}
P(a) &= \begin{cases} P_{min}, & \text{if } \overline{t_n} \leq (\overline{t_{n-1}} + k \cdot \sigma_{t_n}) \\ p_b + p_a \cdot t(a), & \text{if } \overline{t_n} > (\overline{t_{n-1}} + k \cdot \sigma_{t_n}) \end{cases} \\
p_a &= \frac{P_{max} - P_{min}}{t_{max} - t_{min}} \\
p_b &= P_{min} - p_a \cdot t_{min}
\end{aligned} \tag{7}
$$

With the equation 7 an agent just increases its change probability $P(a)$ if its current mean time/cost is greater than last mean, with a threshold. The closer its current decision is to the global minimum more it tries to stay with that decision for the next step, in a linear way.

Weighted Route Choice. If an agent decides to change its route it will chose between all available routes according to a weighted roulette. Routes' weights are evaluated according to the equation 8, where W_a is a set of weights for agent a; $w_{a,r}$ the weight of route r for agent a; w_{min} and w_{max} minimum and maximum weights allowed; w_{ref} weight for own time/cost and $t(r)$ time for route r.

$$
\begin{aligned}
W_a &= \{w_{a,r} \in W_a \wedge r \in R| \\
w_{a,r} &= \left| \begin{matrix} w_{max}, & \text{if } t(a) = t_{min} \\ w(a,r), & \text{if } t(a) > t_{min} \end{matrix} \right\} \\
w(a,r) &= w_{min} + e^{w_b(a) + w_a(a) \cdot t(r)} \\
w_a(a) &= \frac{ln\left(\frac{w_{ref} - w_{min}}{w_{max} - w_{min}}\right)}{t(a) - t_{min}} \\
w_b(a) &= ln(w_{max} - w_{min}) - w_a(a) \cdot t_{min} \\
& w_{min} < w_{ref} < w_{max}
\end{aligned} \tag{8}
$$

With that equation the fastest route will have the maximum weight and the others have an exponential decreasing weight according to its time/cost, but the current decision has a fixed weight w_{ref}. Through that algorithm agents try to maximise its own profit, by taking a faster route.

Parameters' Influence. The influence of parameters can be summarised into the following. As higher the history size as less the route will change, but a higher size gives a less efficient route distribution and, therefore, a higher total cost ($\sum_{a \in Agents} t(a)$), that express how bad are the decisions. P_{min} and P_{max} also interferes on stability, but with less effect on total cost. As higher P_{min} and P_{max} higher the route change, as expected. The parameter k has the inverse effect on route change, as higher as more stable. It acts like a "satisfaction" parameter and can be expressed as the interference of system instability on agent's "willingness" to change its current route, a higher k leads to a lesser sensibility to system's instability.

There are also weights – w_{min}, w_{ref} and w_{max} – that controls how strong it will prefer better routes. It can not be a binary function in terms of assigning w_{max} to all better routes, because it greatly increase the route change instability.

5 Results

Analysis criteria will be explain here. First, the same results from econometrics must be achieved using the agent-base econometric model. That means that the model's quality is not its "distance" from global optimal distribution, but the "optimum" distribution from table 1. The measures used here to analyse the model were route occupation. So the modell was calibrated to reproduce the results from econometrics, assumed to represent current state and thus reality.

Table 1. Route results comparison

	Econometrics				MAS	MAS × Econometrics
Route	$\overline{q_r}$	σ_{q_r}	min	max	q_r	$q_r - \overline{q_{r_e}}$
r1	776.25	107.37	570	990	796	19.75
r2	850.00	142.55	600	1080	876	26
r3	418.75	107.43	120	570	462	43.25
ov1	661,25	75.83	600	930	667	5.75
ov2	307.50	174.54	0	600	199	-108.5

5.1 General Simulation Results and Comparison

The first evaluation value is route total cost on figure 2. Through that, it is possible to detect equilibrium achievement and its type. There are different curves plotted on figure 2. Those curves refer to routes' occupation and total cost. There, is possible to see the agents behaviour and equilibrium process. Note that after 400 simulations steps the system is already stable.

Econometric's values were extracted from [9] (table 15, on page 100) where Probit, Nested-Logit, Cross-Nested, C-Logit, PS-Logit and Nested-C-Logit were used, and they are expressed on columns identified with *Econometrics* from table 1. Results that are in columns identified with *MAS* represent the agent approach

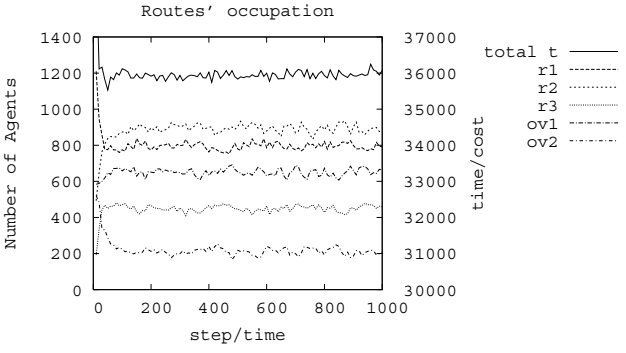


Fig. 2. Route occupation

Table 2. Agent-based econometric model's parameters

Param.	hs.	P_{min}	P_{max}	k	w_{min}	w_{ref}	w_{max}
Value	20	0.05	0.9	0.02	1	1.5	2
eq.	7	7	7	7	8	8	8

and were collected with the parameters shown on table 2, where *hs.* is history size and *eq.* is equation's reference.

The objective was neither to perform better nor worse than econometrics approach and rather to reproduce it. The results are on table 1, where MAS results differ less than one standard deviation from econometrics – refer to column $q_r - \overline{q_{r_e}}$ from table 1. And all values are on the limits established by econometrics – columns *min* and *max* on table 1.

On figure 2 it is clear that no absolut equilibrium is achieve, that would be characterised by flat lines, but rather a dynamic one. That can be seen as advantage or disadvantage. If one expect an absolut equilibrium and clear differences between the options, this model is not an option. But it can also be seen as sign of model's adaptadability where agents are always trying to find a new better route (according to criteria already discussed).

The MAS was modelled using SeSAm [10]. The parameters' setup shown in table 2 were found to be the most suitable for this example. But some fine-tuning on those leads to other results.

6 Related Work

That is not the first approach trying to tackle discrete choice problems using agents. In traffic engineering and simulation, there are several works using different agent technologies, like BDI in [11] and [12]; reinforced learning in [13] and [14]. In all of them, agents try to maximise their rewards with a monetary interpretation. Compared to this work, where agents minimise their costs (with a time interpretation), the approach is very similar.

Other approaches using Artificial Intelligence are also available but they are macroscopic and do not use agents, like Neural Networks in [15]. Also an individual layered fuzzy logic was used for an artificial scenario in [16]. A Lagrangian heuristic on a commuter scenario was used by [17] to schedule traffic demand. Our present work aims at finding a relation between agents and econometric simulation and is focused on a methodological level.

7 Conclusion and Future Work

We presented an econometric agent-based model and discussed how it performs. A simple example was analysed. The results from econometrics were compared with the econometric agent-based model simulation. Those results fit into the acceptable range, determined by the different econometric models. One can argue that the example is too simple, but the objective in this paper was to show that it is possible to get equivalent results using agent technology.

It is important to develop an agent-based alternative to econometrics. Agents are the natural paradigm for modelling and simulating humans, besides the fact that it is easier to model agent behavior than the whole system. Moreover, agent can adapt their parameters and behavior, a characteristic not present on econometrics.

Currently, we are extending the model for applying it to a complex real-world scenario, namely for developing a combined route and mode choice to Bern, Switzerland. We therefore apply also agent-based learning for generation of alternatives, etc. A first prototype already exists which dynamics are currently evaluated.

Acknowledgements

Gustavo Kuhn Andriotti is financially supported by CNPq. We thank Guido Rindsfser (Emch & Berger AG Bern) for valuable discussions on econometrics and traffic simulation.

References

1. Fehler, M., Klügl, F., Puppe, F.: Approaches for resolving the dilemma between model structure refinement and parameter calibration in agent-based simulations. In: Proceedings of the second international joint conference on Autonomous Agents and Multi-Agent Systems, AAMAS, Hakodate, Japan (2006) to appear.
2. Luce, R.D.: Individual Choice Behavior: A Theoretical Analysis. Dover Publications, New York (1959)
3. Marschak, J.: Binary choice constraints on random utility indications. In Arrow, K., ed.: Stanford Symposium on Mathematical Methods in the Social Sciences, Stanford, CA, Stanford University Press (1960) 312–329
4. Luce, D., Suppes, P.: Preferences, utility and subjective probability. In Luce, R., Bush, R., Galanter, E., eds.: Handbook of Mathematical Psychology. John Wiley and Sons, New York (1965) 249–410

5. McFadden, D.: Conditional logit analysis of qualitative choice behavior. Frontiers of Econometrics (1974)
6. Ben-Akiva, M.: The structure of travel demand models. PhD thesis, MIT (1973)
7. McFadden, D., Train, K.: Mixed mnl models of discrete response. Journal of Applied Econometrics **15** (2000) 447–470
8. Train, K.E.: Discrete Choice Methods with Simulation. Cambridge University Press (2003)
9. Vrtic, M.: Simultanes Routen- und Verkehrsmittelwahlmodell. PhD thesis, Technischen Universität Dresdner (2003)
10. Klügl, F., Puppe, F.: The multi-agent simulation environment sesam. In: Workshops Simulation in Knowledge-based Systems, Reihe Informatik, Universität Paderborn (1998) 194
11. Rossetti, R.J.F., Liu, R.: A dynamic network simulation model based on multi-agent systems. In: ATT 2004, 3rd Workshop on Agents in Traffic and Transportation, New York, AAMAS 2004 (2004) 88–93
12. Dia, H.: An agent-based approach to modelling driver route choice behaviour under the influence of real-time information. Transportation Research Part C: Emerging Technologies **10**(5–6) (2002) 331–349
13. Bazzan, A.L.C., Klügl, F.: Route decision behaviour in a commuting scenario: Simple heuristics adaptation and effect of traffic forecast. In: Proceedings of the Euroworkshop on Behavioural Responses to ITS, Eindhoven (2003)
14. Bazzan, A.L., Klügl, F.: Case studies on the braess paradox: Simulating route recommendation and learning in abstract and microscopic models. Transportation Research Part C: Emerging Technologies **13**(4) (2005) 299–319
15. Dia, H.: An object-oriented neural network approach to short-term traffic forecasting. European Journal of Operational Research **131**(2) (2001) 253–261
16. Ridwan, M.: Fuzzy preference based traffic assignment problem. Transportation Research Part C: Emerging Technologies **12**(3–4) (2004) 209–233
17. Castelli, L., Pesenti, R., Ukovich, W.: Scheduling multimodal transportation systems. European Journal of Operational Research **155**(3) (2004) 603–615

Agent Based Simulation Architecture for Evaluating Operational Policies in Transshipping Containers

Lawrence Henesey, Paul Davidsson, and Jan A. Persson

Department of Systems and Software Engineering, Blekinge Institute of Technology.
Box 214, Karlshamn, Sweden
{lhe, pdv, jps}@bth.se

Abstract. An agent based simulator for evaluating operational policies in the transshipment of containers in a container terminal is described. The simulation tool, called SimPort, is a decentralized approach to simulating managers and entities in a container terminal. We use real data from a container terminal, for evaluating eight transshipment policies. The simulation results indicate that good choices of yard stacking and berthing position polices can lead to faster ship turn-around times, for instance, the Overall Time Shortening policy offers a lower cost and when combined with a Shortest Job First sequencing of arriving ships on average yielded a faster ship turn around time. The results also indicated, with respect to the studied performance measures that Stacking by Destination is a good choice of policy.

1 Introduction

The growth in the use of containers for transporting goods has been profound from 39 million containers handled in 1980 to over 356 million in 2004 and the annual growth rate is projected at 10 percent till 2020 [1]. Parallel with the increasing demands for transporting cargo in containers is the increasing importance in improving container terminal (CT) operations. As the primary function of CTs is to provide for efficient, low-cost, inter- and intramodal transfer, inspection, storage, form change, and control of cargo, the CT must be able to effectively act as an integral part of transport chain from origin to destination [2]. According to Frankel [2], port costs can be in excess of 50 percent of the total costs of which 55 percent of these port related costs are the result of poor ship turn-around times and low cargo handling speeds. The increasing demands on CTs is placing pressure on the management of CTs in finding ways to increase capacity and offer more efficient ship handling operations. From Frankel [2] we can infer that CTs can influence the physical constraints on the size and type of ships that can be served, number of containers that can be handled, berth utilization determined by service time and ship operating costs. Research by De Monie [3], has identified several key parameters of CT capacity that can be improved through computerized planning, control and maintenance systems such as: berthing arriving ships, scheduling the ship-to-shore handling, coordinating the terminal transfer, and managing the stacking/un-stacking of containers in the yard.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 73 – 85, 2006.

There has been much research in CT effectiveness, capacity and technology. A literature survey overview on transshipment operations has been provided by Vis and Koster [4] and Meersmans and Dekker [5] followed by a rather comprehensive survey on container terminal logistics by Steenken et al. [6]. A classification of container terminal operations is provided by Henesey [7], which concludes that simulation models have been used extensively in understanding the behavior, experimenting and testing conditions and scenarios due to the cost and complexity of the CT domain. A number of simulators and simulation models have been developed in studying CTs and they differ widely in objectives, complexity and details, but all suggest or propose a centralized system for scheduling or controlling [8]. The distributed systems approach has been investigated by a number of papers in solving scheduling or control problems in CTs by using agent technology, such as [9-16]. However, these papers have mostly focused on techniques for automating or controlling the operations in a CT. This paper presents a multi-agent based simulator called SimPort (Simulated container Port) that is developed, as part of an IDSS (Intelligent Decision Support System) assisting human CT managers in the decision making process for transshipment operations in a CT.

The remainder of the paper is organized as follows; in section 2 a general description of the transshipment processes in a CT. In section 3, a research question is formulated and the methodology is described. The SimPort model is explained in section 4. In section 5, a description of a simulation test and the initial results are presented. In section 6, a conclusion with pointers for future work is presented.

2 Description of Container Terminal Transshipment Operations

Many shipping companies are trying to serve a geographic region, such as Europe, by establishing two or three main hubs from which smaller container ships will "feed" containers to and from other ports or CTs in the region. This 'hub and spoke' method of servicing ship line customers is similar to that used by the airline industry in transporting people in smaller aircraft from a region via large international airports connecting with often larger airplanes to distant destinations or offering many destinations. The amount of transshipping is increasing and according to a study by OCS, [17] total transshipment throughput for Europe and the Mediterranean has increased by 58 per cent over 2000-2004 to 22.5 million TEU (Twenty-foot Equivalent Unit). Many CTs are fast becoming known as transshipment terminals in which they will be linked with 'feeder' ships and the containers from various ports and CTs are consolidated for loading on larger ships for transporting to another region. Specialized transshipment CTs that have been developed as a consequence to the large flow of containers being transshipped are for example; Malta, Gioia Tauro, Salalah, Algeciras, Singapore, Hong Kong, and Shanghai [18].

In managing the CT, the transshipment operations in moving containers can be divided into four sub-processes: *ship arrival, loading /unloading, horizontal transport* and *yard stacking / unstacking* [4]. The four sub-processes in transshipment operations are described as follows:

Ship Arrival - The arrival of a ship requires CT management to locate a berth position so that it can moor along the quay and a service time to schedule operations. This decision on choice of a berth policy has an impact on other decisions in the ship operations. The berth 'policy' is often formulated from choosing a sequence policy and a positioning policy. Basic questions are *when* and *where* to place an arriving ship.

Loading and Unloading - The loading and unloading sub-processes requires operational decisions by the CT management in allocating Quay Cranes (QC) and transport equipment such as Straddle Carriers (SC) or trucks and labor. Usually, the allocation of these resources is conducted in parallel with the ship arrival process. The container stowage planning in a ship is a rather complex problem to solve and has recently been studied by Wilson and Roach [19], to take as much as $1x10^{27}$ years to optimally load a 3,000 TEU. Obviously, we need quicker solutions for loading and unloading a container ship.

Horizontal Transport - An objective that many CT managers share is trying to keep the assigned QCs from being idle or avoiding interruption during operations so as to quickly service a ship. The availability, allocation and efficient use of terminal transport are very important to ensure that the QCs are productive. In [20] mentions that many CT managers view the interface between the QCs and the yard to be a problem. Some problems in the horizontal transport process are: load sequence, routing, pickup sequencing and coordination with QCs.

Yard Stack / Stack on Quay - Containers are usually sorted using a stacking policy which may consider, for example; type (export or import), and size (i.e. 40' foot or 20' foot), destination, or by ship line that owns the container, etc. Ideally, in transshipment operations the ship that is unloading the containers to be loaded by another ship will be serviced at the same time with the other ship in order to avoid problems of stacking containers. This scenario offers a faster service. However, in reality the containers must often 'dwell' or be placed in a yard stack for a period of time while waiting to be loaded onto another arriving ship. Some problems or decisions affecting this process are: stacking density; yard stack configuration; container allocation to a stack according to rules of "polices"; and dwell times.

3 Research Questions and Methodology

In this paper, we pose the following research question; "how could simulation be used to study the impact of the different policies for sequencing of arriving ships, berthing policies, and container stacking on the performance of transshipment operations at a CT?"

The research question stemmed from discussions with CT managers and the results from the reviewed literature [7]. There often appeared to be a gap in understanding the complexity of the decisions by the CT managers in the management of a CT, such as berth assignment, from both theoretical perspectives and from industry practice. Often mentioned, is that existing tools are too cumbersome, do not accurately model the CT, are too expensive and not fast. In addition, some CT experts confided that berth allocation was conducted mostly by middle managers, who did not possess enough information in making the berth assignment decision.

We considered a simulation technique called Multi Agent Based Simulation (MABS) that is suggested by [21] to be applicable to domains that are distributed, complex, and heterogeneous. Before considering using simulation, such as MABS, we first considered other methods for experimenting such as: analytical models in econometrics; mathematics, and optimization However, CTs possess many characteristics listed by [22] that are be deemed suitable for considering simulation such as: random variables, large number of parameters, non-linear functions and behavior of a dynamic system.

Simulation in general can be used to study the dynamics of complex systems and how the various components of the system interact with each other [22]. The reason for simulation is that it is a good way for people to form cognition; *action or process of acquiring knowledge*.

The choice on using MABS specifically is based in the versatility in simulating complex systems and perceived easiness from which modeling a CT can handle different levels of representation, such as real human managers in a management system. Paranak et al. [23] recently compared macro simulation and micro simulation approaches and pointed out their relative strengths and weaknesses. They concluded, "…agent-based modeling is most appropriate for domains characterized by a high degree of localization and distribution and dominated by discrete decision. Equation-based modeling is most naturally applied to systems that can be modeled centrally, and in which the dynamics are dominated by physical laws rather than information processing." As a CT has a high degree of localization and distribution and is dominated by discrete decision, we found agent-based modeling a promising approach worthy to investigate.

4 SimPort Architecture

Using a knowledge engineering methodology known as MAS-commonKADS, we model the CT managers into a management system by identifying the following: their tasks, how they are organized, methods for communication and coordination mechanisms [24]. The management system simulator is based on the following managers that are modeled as agents: *port captain, ship agent, stevedore,* and *terminal manager.* Additional agents, which are modeled in the CT simulator, are the QCs and the SCs. The management model of the CT manager agents is illustrated in Figure 1, which show how the agents are organized hierarchically, communicating and coordinating, represented by arrows, and interacting with the CT simulator by sending actions and receiving observations.

The agents make decisions based on information in the messages they receive from each other. The intelligence level of the agents, such as stevedore, ship and crane agents can be considered reactive in that a specific action in the CT is executed upon a certain message. The major advantage in using reactive agents, according to Wooldridge [21], "*is that overall behaviour emerges from the interactions of the component behaviours when the agent is placed in its environment*". The interaction between the agents is summarized in Fig. 2, which adopts a pseudo AUML (Agent Unified Modeling Language) sequence diagram. The agent's goals are only implicitly

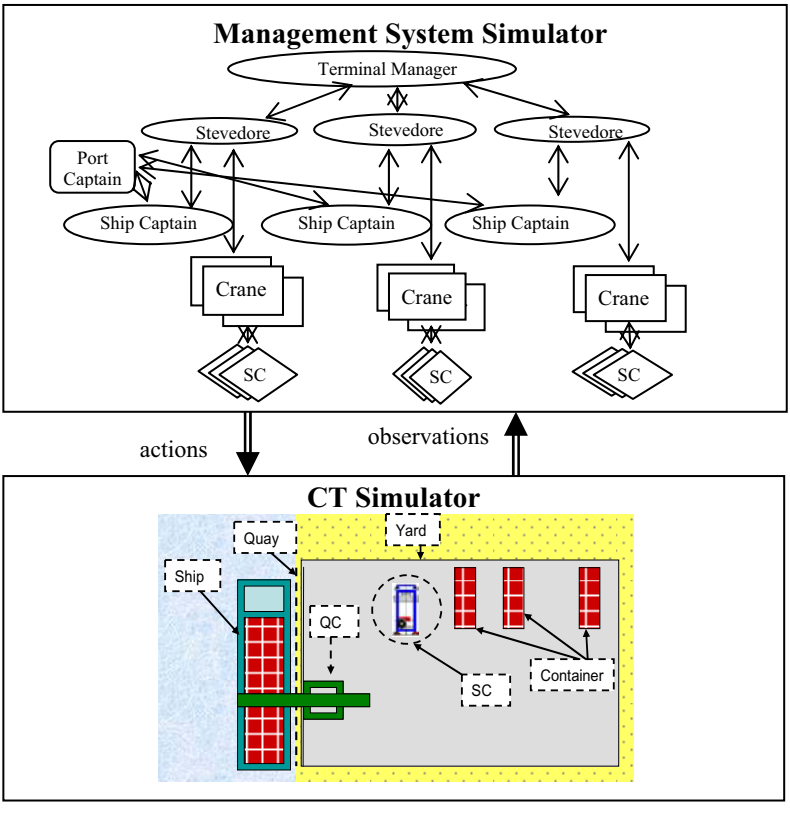


Fig. 1. Simplified view of the SIMPORT Architecture

represented by the rules describing the reactive behavior, illustrated in Fig. 2, for the following agents in SimPort:

Port Captain Agent; is constantly during a 24 hour period searching for arriving ships during the next 24 hour period. Based on the ship's estimated arrival time, number of containers and size of ship, the port captain creates a *ship slot* which decides the order the arriving ships will be served according to a sequence policy, e.g., First In First Out (FIFO), Highest Earning First (HEF) and Shortest Job First (SJB).

Ship agent is created to represent each arriving ship to the CT. The ship agent will possess the following information:

– Ship Properties:
  1. Desired service time (t_i^{serv}) is based on the schedule, from the ship line perspective, listing estimated arrival time (t_i^{arriv}) and estimated departure time (t_i^{dep}).
  2. Length of ship in meters (l_i).
  3. Type of ship (v_i) which is regular, panamax or post panamax,
  4. The Ship line that owns the ship.
  5. The number of bays in the ship (j_i).
  6. The hourly operating cost.

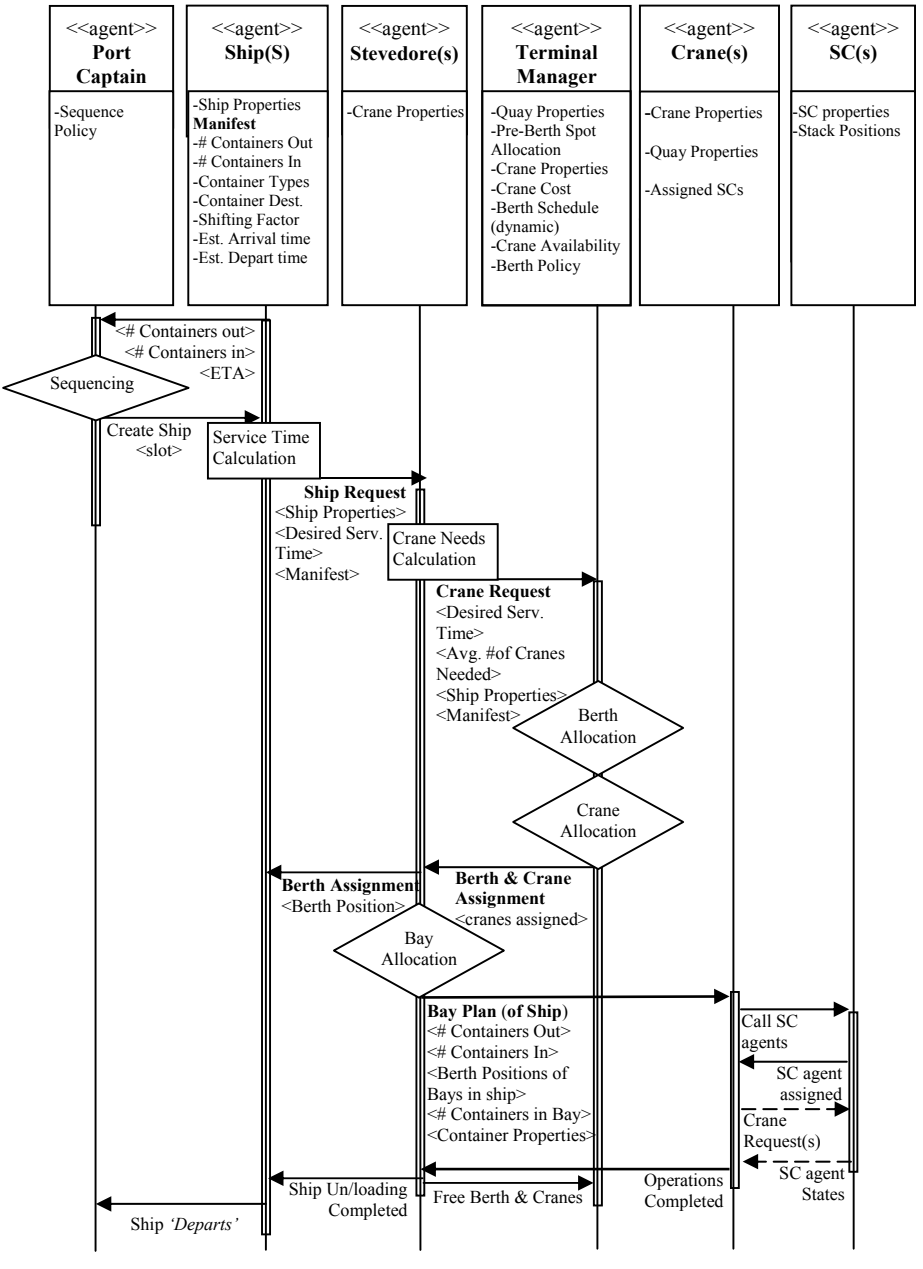


Fig. 2. AUML Sequence Diagram of Agents in SimPort

– For each bay, the 'manifest' provides the following data; number of containers, and for each container type (whether a 40', 20', Refrigerated or Hazardous), destination (from which we can infer it to be either an Export or Import container) and ship

line (containers on board the ship may belong to other ship lines and this will affect in stack assignment).

When the ship is to be served, the ship agent sends its 'ship request' or desired service time, t_i^{serv}, which can be considered a duration of service, to the stevedore agent, which is computed in the following way (where t_i^{wait} is the waiting time):

$$t_i^{serv} = t_i^{dep} - t_i^{arriv} - t_i^{wait} \quad (1)$$

Stevedore agent will try to satisfy each ship agent's request, i.e., to be served in less time than the t_i^{serv}. It will request quay cranes from the terminal agent that can handle the ship type, v_i and a position of the cranes in order to serve the bays in a ship while trying to meet the estimated desired service time. The crane request is based on a calculation of the average number of cranes needed to work the ship. For example, if the number of containers to be loaded/unloaded, C_i is 400 and the desired service time corresponds to 4 hours and the average capacity of the cranes, Q^s, is 25 moves per hour, then the number of cranes requested, Q is 4. (The reason for using the average capacity is to mirror the actual computations performed by actual stevedores.) The general formula used is:

$$Q = C_i / (Q^s * t_i^{serv}) \quad (2)$$

The second task of the Stevedore agent is to allocate the cranes provided by the Terminal manager agent to the different bays of the ship. It receives information from the ship agent regarding the number of containers in the bays, number of bays in the ship and the characteristics of the containers (size, type, destination and ship line). The bay allocation is done by assigning cranes to work an average number of containers (both load and unload) for all bays in a ship.

Terminal Manager agent performs two tasks, allocation of berth points to a ship and allocating cranes to service a ship. It receives information from the stevedore agent on ship length (l_i) and will assign a sequence of berth points (b_i) along the quay that the ship will occupy, which will include the spacing between two ships. From the 'request' sent by a stevedore agent, one for each ship, the terminal manager will allocate available cranes that can handle a ship type. Crane allocation is determined by crane type(s) that can work a ship type, v_i and their distance to the berth spot. The number of cranes is limited and this may cause ships to either have slower service times or even wait. Cranes are assigned by the average number of crane moves per hour Q^s, and dividing to the number of containers, C_i to be worked for ship v_i.

The berth positions used by the terminal manger for the arriving ships will be determined by a *berth positioning policy*. From interviews with CT managers and collected data, two types of *berth positioning policies* have been identified that are actually used; Berth Closest To the Stack (BCTS) policy and Overall Time Shortening (OTS) policy.

The BCTS policy's objective is to place a ship closest to a 'target' stack which is the stack that will be the most visited by the SCs during the operations. That is, the one that has the largest sum of (i) containers to be stored and (ii) containers to be fetched. The BCTS will wait if a berth is occupied by another ship until that berth, which is closest to the stack, is available. The OTS policy, on the other hand, tries to place the ships to berth positions in order to minimize the total ship turn-around-time

for all arriving ships in a scheduled period of time. In determining the berth position for an arriving ship the OTS policy is considering the *Waiting Time* during the simulation from a potential set of berth points. The number of possible berth points depends on the berth spacing as well as a ship's length plus a buffer distance. The ship *Waiting Time* includes time left in serving another ship that is occupying a part of the quay. The estimation of the *Service Time* is based on the number of SCs employed, the routes covered by SCs and their average speed. The estimated sums of all the routes traveled by each SC are totaled to provide the distance being covered by the SCs for each ship. From the sum of the estimated *Service Time* and *Waiting Time*, i.e. the turn-around time of the OTS policy will place a ship wherever the shortest estimated ship turn-around-time is achieved.

Quay Crane (QC) agents are coordinated by a stevedore agent during operations. It receives a list from the stevedore agent which states all containers that should be unloaded/loaded from/to each bay. Based on this list, the Crane agent will react by calling its assigned SC agents and based on their replies, select the SC agent most appropriate to pick up a particular container based on a) availability (idle/busy) and b) the distance between the SC and the container. The general objective for the crane agents is to load/unload containers as fast as possible and use the SCs to move the containers to and from the stacks in the most efficient way possible.

Straddle Carrier (SC) agents are reacting to requests from their assigned Crane agent; an assumption based upon observations of real CTs where a number of transporters typically are 'bounded' to a specific crane. The SC agents have a map of the CT and their goal is just to satisfy the request of its crane agent. SC agents will send their state to the Crane agent. For example, if the stack that the SC has been assigned to place a container in is full, the SC will go to the closest available stack. The SC move along one-way paths for safety reasons. The SC agents calculate the distance from the top left corner of a stack to the position of the crane working a ship's bay located at the berth point along the quay. Distance of the stacks may have an influence on the handling rate of the QCs working a ship.

A SC agent determines its next destination through communication with the crane agent. The SC moves to a position in the yard that is generated by communication with the crane agent and subsequently establishes its next position by communicating back to crane agents that it has reached its assigned destination and is waiting for another task. The SC agent's function is to provide specific yard destinations rather than the container processing sequence. The model contains rules which determine an appropriate yard location based on current status of the stacks and stacking policy, and attributes of the SC agent.

5 Initial Experiment

A real CT was having problems in serving arriving ships leading to ship waiting times on average three days. The SimPort model was used to evaluate revised stacking configurations for the yard and the transshipment operational policies.

5.1 Experiment Setup

The managers at the CT provided data and layouts of their terminal for analysis. The following entities of the CT terminal were modeled in SimPort:

- *Terminal***:** Length and width (meters), e.g., 900 m x 1000 m; Operating hours, e.g., 07:00 – 20:00 from Monday to Friday; The terminal handling charge (THC), a cost paid by the ship lines for handling each container unit (100 dollars per container); A "penalty" cost, an extra cost for handling containers out of operating hours (150 dollars per container); *A yard;* and A *quay.*
- *Yard Stacks*: Length of the Yard is 1000 meters and width 890 meters. Six large stacks that can store 180 containers each are created in the SimPort model using data from the real CT. The yard stacks temporarily store containers based upon export or import status. All stacks are assigned to a number of "ports of destinations", which are based on six different import and export destinations.
- *Quay length*: The of the length of the quay that is able to serve docked container ships is tested at 890 meters and the width of the terminal yard is 1000 meters to reflect the actual CT. Four berths are configured with a fixed length of 200 meters along the quay. Additionally, the distance between ships worked at the quay is 20 meters and 5 cranes are assigned to the quay.
- *Quay Cranes*: Five QCs are assigned to work ships along a quay at the CT with a handling rate of 25 container moves per hour.
- *Straddle Carriers*: Twenty SCs are employed during operations; four SCs are assigned to each crane. The SCs have a capacity of lifting one container over three and are set with a maximum speed of 30 km/h.
- *Sequence of arriving ships:* The data was provided by CT managers at the real CT for developing the scenarios, the arrival time intervals of 3 container ships (each 200m long) and the total number of containers for the 3 ships is 1100 for export and 1000 for import, which are identified as either reefer (5%), hazard (5%), and standard (90%). In addition, each container is loaded (exported) or unloaded (imported) to/from a specific bay located on a ship. The arrival times for all 3 ships were randomly generated between 07:00 and 12:00.
- *Berth Policies*: The berth positioning polices tested are the BCTS and OTS. Polices tested for sequencing arriving ships are FIFO, HEF, and SJB. Two container stacking polices are tested, stack by Ship and stack by Line.

The output from the SimPort will be a berth assignment plan for scheduling, which includes the sequencing of arriving ships and the berth position that they will occupy along the quay. Terminal equipment will be assigned, e.g., QCs and SCs, to work ships. Terminal handling costs charged by the CT in handling a TEU are provided. Finally, to compare performance levels of the various operational policies used, the following measures of performance are defined:

- *Total Distance* – Total distances traveled for all the SCs used to serve the QCs for all three ships.
- *Average Ship Turn-Around Time* – Average time for turning-around a ship in a schedule (departure time – arrival time).

- *Average Waiting Time* – Average *Waiting Time* for a ship in a schedule.
- *Total Costs* – Total costs for serving all ships computed from the number of hours that each ship is berthed multiplied with its hourly operating cost plus the THC (Terminal Handling Cost) that is assessed for each container handled for each ship.

5.2 Initial Experiment Results

The simulation results to evaluate policy combinations for a particular CT are presented in Table 1, which presents the averages from 10 simulation runs

Table 1. Simulation results from initial experiment

Simulation Policy:	**Stacking by Ship Line**					
	BCTS			**OTS**		
	FIFO	**HEF**	**SJB**	**FIFO**	**HEF**	**SJB**
Total distance (meters):	213460	219887	208806	239820	240115	238331
Average Ship Turn Around Time:	10:38	10:55	10:21	7:22	7:25	7:17
Average Waiting Time:	03:30	03:48	03:16	00:21	00:25	00:17
Total Costs ($):	187500	200500	187500	150700	151010	149600
Simulation Policy:	**Stacking by Destination**					
	BCTS			**OTS**		
	FIFO	**HEF**	**SJB**	**FIFO**	**HEF**	**SJB**
Total distance (meters):	205250	211430	200775	227333	229720	225875
Average Ship Turn Around Time:	10:20	10:37	10:04	7:13	7:10	7:07
Average Waiting Time:	3:23	3:41	3:10	0:13	0:10	0:07
Total Costs (€):	184500	194000	184500	148200	149300	147600

Total Distance – the shortest distances traveled by the SCs on average were found to be when applying the BCTS with the SJB policy; 208806 for stacking by Ship and 200775 for stacking by Destination. Within the OTS position policy, there are slight differences in distances traveled, which indicated that the HEF will yield the longest distances followed by FIFO and SJB. In comparing stacking policies, the stack by distance yielded the shortest distances compared to stack by line. The shortest distance recorded for OTS was when simulating with the SJB sequence policy, which yielded a distance of 225875.

Average Ship Turn-Around Time – average ship turn around per ship was found to be faster when using the OTS policy with an average of 7:21 hours for stacking by line and 7:10 for stacking by destination. The BCTS policy yielded an average ship turn around of 10:38 hours for stacking by line and 10:20 for stacking by line. The

ship turn around time was faster when simulating with the SJB sequencing policy for both position policies.

Average Waiting Time – average waiting times are longer, when comparing position policies, in the BCTS are 3:31 hours (for all three sequence policies) with stacking by line and 3:25 for stacking by destination. The OTS had shorter waiting times averaging: 21 minutes for stack by line and: 10 minutes for stack by destination. Within the position polices the fastest waiting times are recorded when simulating with the SJB policy. In comparing the average waiting times between stacking policies, the stack by Line on average had a longer waiting time.

Total Costs – The lowest cost for ships was recorded when simulating the OTS policy. The costs are lower in OTS since the turn-around time is lower then the BCTS, which influences the hourly operating costs of the ships. In comparing within the OTS, the sequence policies suggest that there is slight influence. The SJB policy in combination with the OTS suggests the lowest cost for the ships. The BCTS policy results indicated that the sequence policies, FIFO and SJB, were the same and the HEF is the most expensive. Stack by Destination on average was lower then stack by Line by €2,100 when using the BCTS policy and € 4,167 for the OTS policy.

6 Conclusion and Future Work

The results from the experiments answered the main research question that was presented; what is the impact of the different policies for sequencing, berthing, and stacking on the performance of CTs? The objective of using a MABS such as SimPort was to analyze which CT management policies could be best considered in relation to: ship arrival patterns, number of containers to be handled during a time period, changes in container stack layout in the yard and berth. In analyzing the question, the SimPort has proven able to reflect many of these types of changes into the model for simulation.

The agent-based manager system which assigns berth schedules from the various management policies has indicated that some policies have faster ship turn-around times and lower distances traveled by SCs over other polices for certain scenarios. In addition, other performances were revealed in choice of policy such as lower costs for ships depending on the scenario; distribution of arriving ships, number of containers to be handled, characteristics of the containers and yard stacking policies.

The initial experiments could be extended to simulate a larger number of ships, a longer period of time and perhaps use more stacking positions. Future plans are to further develop SimPort in order to evaluate IPSI® AGVs (Automated Guided Vehicles) [25] coordination in a CT. Additional logic for the manager agents could be used for enhancing the decisions made. An optimizer for calculating the best berth position would offer further benefits to the simulation model. Often mentioned by CT managers is to incorporate economic or cost indicators into the simulation, such as cost per hours for groups employed to work a ship, cost for fuel consumed by SCs, number of containers handled during a specific period and profit or loss made.

References

1. Davidson, N.: A global capacity assessment and needs analysis, *Proceedings of the 39th Terminal Operating Conference*, Antwerp, Belgium, 2005.
2. Frankel, E. G.: *Port Planning and Development*. John Wiley & Sons, New York, US, 1987.
3. De Monie, G.: Inter-port cooperation and competition: developing trans-shipment operations in the Mediterranean Environmental Scanning in Ports, *Proceedings of the Fourth MedTrade 2006*. St. Julians, Malta, 2006.
4. Vis, I. F. A. and de Koster, R.: Transshipment of containers at a container terminal: An overview. *European Journal of Operational Research*, Vol. 147, pp. 1-16, 2003.
5. Meersmans, P. J. M and Dekker, R.: Operations Research supports container handling. Technical Report, *The Netherlands Econometric Institute Report EI 2001-22,* Econometric Institute, Erasmus University, Rotterdam, November 2, 2001.
6. D. Steenken, D., Vos, S., and Stahlback, R.: Container terminal operations and operations research - a classification and literature review. *OR Spectrum*, Vol. 26, pp. 3-49, 2004.
7. Henesey, L. *Enhancing Container Terminals: A Multi-Agent Systems Approach*, Licentiate Thesis. Department of Systems and Software Engineering, Blekinge Institute of Technology, Sweden, pp1-132, 2004.
8. Liu, C.-I., Hossein, J. and Ioannou, P. A.: Design, Simulation, and Evaluation of Automated Container Terminals. *IEEE Transaction on Intelligent transportation systems*, Vol. 3, pp. 12-26, 2002.
9. Buchheit, M., Kuhn, N., Müller, J.P., and Pischel, M.: MARS: Modeling a multiagent scenario for shipping companies. *Proceedings of the European Simulation Symposium (ESS-92)*, Dresden, Germany, 1992.
10. Degano, C., and Pellegrino, A..: Multi-Agent Coordination and Collaboration for Control and Optimization Strategies in an Intermodal Container Terminal. *Proceedings of the IEEE International Engineering Management Conference (IEMC-2002),* Cambridge, UK, 2002.
11. Gambardella, L.M., Rizzoli, A.E., and Zaffalon, M.: Simulation and planning of an intermodal container terminal. *Simulation*, Vol. 71, pp. 107–116, 1998.
12. Rebollo, M., Vicente, J., Carrascosa, C., and Botti, V.: A Multi-Agent System for the Automation of a Port Container Terminal. *Proceedings of Autonomous Agents 2000 workshop on Agents in Industry,* Barcelona, Spain, 2000.
13. Rebollo, M., Vicente, J., Carrascosa, C., and Botti, V.: A MAS Approach for Port Container Terminal Management. *Proceedings of the 3rd Iberoamerican workshop on DAI-MAS*, Atiaia, Sao Paulo, Brazil, 2001.
14. Lee, T.-W., Park, N.-K., and Lee, D.-W.: Design of Simulation System for Port Resources Availability in Logistics Supply Chain," *Proceedings of the International Association of Maritime Economists Annual Conference, (IAME 2002),* Panama City, Panama, 2002.
15. Carrascosa, C., Rebollo, M., Vicente, J., and Botti, V.: A MAS Approach for Port Container Terminal Management: The Transtainer Agent," *Proceedings of the International Conference on Information Systems, Analysis and Synthesis*, Orlando, US, 2001.
16. Thurston, T., and Hu, H.; Distributed Agent Architecture for Port Automation, *Proceedings of the 26th International Computer Software and Applications Conference (COMPSAC 2002),* Oxford, UK, 2002.
17. Ocean Shipping Consultants: *European and Mediterranean Containerport Markets to 2015*. Ocean Shipping Consultants, Ltd., Surrey, UK 2006.

18. Baird, A.: Optimising the container transhipment hub location in northern Europe. *Journal of Transport Geography*, vol. article in press, 2006.
19. Wilson, I.D. and Roach, P. A.: Container stowage planning: a methodology for generating computerised solutions. *Journal of the Operational Research Society*, Vol. 2000, pp. 1248-1255, 2000.
20. Henesey, L.: Agent Based Simulation for Evaluating the Operational Policies in the Transhipping of Containers. Submitted for publication to *Transportation Review,* 2005.
21. Wooldridge, M.: *An Introduction to Multi Agent Systems*. John Wiley and Sons Ltd., West Sussex, England, 2002.
22. Wernstedt, F.: Simulation, An Overview. Manuscript, Blekinge Institute of Technology, Karlskrona, Sweden, 2001.
23. Parunak, H. V. D., Savit, R., and Riolo, R. L.: Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide. *Multi-Agent Systems and Agent-Based Simulation*, Vol. LNAI 1534, Eds: Sichman, J. S., Conte, R., and Gilbert, N., Springer-Verlag, pp. 10-26, 1998.
24. Henesey, L., Notteboom, T., and Davidsson, P: Agent-based simulation of stakeholders relations: An approach to sustainable port and terminal management. *Proceedings of the International Association of Maritime Economists Annual Conference, (IAME 2003)*, Busan, Korea, 2003.
25. TTS AB Port Equipment AB. Gothenburg, Sweden, Internet site: http://www.tts-hs.no/, visited last (03.07.2006).

Diagnosis of Multi-agent Plan Execution*

Femke de Jonge[1], Nico Roos[1], and Cees Witteveen[2]

[1] Dept of Computer Science, Universiteit Maastricht
P.O.Box 616, NL-6200 MD Maastricht
{f.dejonge, roos}@cs.unimaas.nl
[2] Faculty EEMCS, Delft University of Technology
P.O.Box 5031, NL-2600 GA Delft
witt@ewi.tudelft.nl

Abstract. Diagnosis of plan failures is an important subject in both single- and multi-agent planning. Plan diagnosis can be used to deal with plan failures in three ways: (*i*) it provides information necessary for the adjustment of the current plan or for the development of a new plan, (*ii*) it can be used to point out which equipment and/or agents should be repaired or adjusted so they will not further harm the plan execution, and (*iii*) it can identify the agents responsible for plan-execution failures.

We introduce two general types of plan diagnosis: *primary plan diagnosis* identifying the incorrect or failed execution of actions, and *secondary plan diagnosis* that identifies the underlying causes of the faulty actions. Furthermore, three special cases of secondary plan diagnosis are distinguished, namely *agent diagnosis*, *equipment diagnosis* and *environment diagnosis*.

1 Introduction

In multi-agent planning research there is a tendency to deal with plans that become larger, more detailed and more complex. As complexity grows, the vulnerability of plans for failures will grow correspondingly. Taking appropriate measures to a plan failure requires knowledge on the causes of these failures. So it is important to be able to detect the occurrence of failures and to determine their causes. Therefore, we consider diagnosis as an integral part of the capabilities of agents in single- and multi-agent systems.

To illustrate the relevance of plan diagnosis, consider a very simple example in which a pilot agent of an airplane participates in a larger multi-agent system for the Air Traffic Control of an airport. Suppose that the pilot agent is performing a landing procedure and that its plan prescribes the deployment of the landing gear. Unfortunately, the pilot was forced to make a belly landing. Clearly, the plan execution has failed and we wish to apply diagnosis to find out why. A first, superficial, diagnosis will point out that the agent's action of deploying the landing gear has failed and that the fault mode of this action is "landing gear not locked". We will denote this type of diagnosis as *primary*

* This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs (the Netherlands). Project DIT5780: Distributed Model Based Diagnosis and Repair.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 86–97, 2006.

plan diagnosis. This type of diagnosis focuses on a set of fault behaviors of *actions* that explain the differences between the expected and the observed plan execution.

Often, however, it is more interesting to determine the *causes behind* such faulty action executions. In our example, a faulty sensor may incorrectly indicate that the landing gear is already extended and locked, which led the pilot agent to the belief that the action was successfully executed. We will denote the diagnosis of these underlying causes as *secondary plan diagnosis*. Secondary diagnosis can be viewed as a diagnosis of the primary diagnosis. It informs us about malfunctioning equipment, unexpected environment changes (such as the weather) and faulty agents. As a special type of secondary diagnosis, we are also able to determine the agents *responsible* for the failed execution of some actions. In our example, the pilot agent might be responsible, but so might be the airplane maintenance agent.

In our opinion, diagnosis in general, and secondary diagnosis in particular, enables the agents involved to make specific adjustments to the system or the plan as to manage current plan-execution failures and to avoid new plan-execution failures. These adjustments can be categorized with regard to their benefits to the general system. Primary diagnosis can contribute to plan repair by identifying the failed action and how they failed. Secondary diagnosis can also can contribute to plan repair by pointing out the broken equipment and the misbehaving agents. Plan repair can either plan to fix the broken equipment or to use other equipment. Moreover, information about agents helps to recover and adjust the agents thereby contributing to a better plan execution. Finally, secondary diagnosis can indicate the agents responsible (accountable) for the failures in the plan-execution. This information is very interesting when evaluating the system, and can also be used to divide costs of repairs and/or changes in the plan amongst the agents.

In this paper we adapt and extend a classical Model-Based Diagnosis (MBD) approach to the diagnosis of plan execution. To enable secondary diagnosis, the system to be diagnosed consists not only of the plan and its execution, but also of the equipment needed for the execution, the environment in which the plan is executed and the executing agents themselves. We introduce an object-oriented description of plans in which objects represent actions, agents, equipment and the environment. The state of these objects may change dynamically by other causes than the execution of planned actions. Primary and secondary diagnosis is used to identify these causes.[1]

To realize the benefits of plan-based diagnosis outlined above, we present in section 3 an adaptation of the object-oriented description of plan execution introduced in [18,16]. Section 4 shows how this object-oriented plan description enables the formalization of primary and secondary plan diagnosis. But first of all, we will place our approach into perspective by discussing some approaches to plan diagnosis in the following section.

2 Related Research

To realize plan diagnosis, we adapt and extend the object oriented representation of plans presented in [18,16]. This representation has been chosen because it is more suited

[1] How to implement diagnosis is outside the scope of this paper. For an example of an approach for distributed diagnosis in a multi-agent system, we refer to [15].

to plan diagnosis than the traditional plan representations such as [10,2,9,17] and it also enables us to extend the classical Model-Based Diagnosis (MBD) approach to the planning domain. Since one of the goals of plan diagnosis is to facilitate plan repair, it might be beneficial to make use of information used to generate a plan and therefore use one of the traditional plan representations. However, we wish to study plan diagnosis independent of any planning approach that is used to generate a plan or that will be used to repair a plan.

Similar to our use of MBD as a starting point of plan diagnosis, Birnbaum et al. [1] apply MBD to *planning agents* relating health states of agents to *outcomes* of their planning activities, but not taking into account faults that can be attributed to actions occurring in a plan as a separate source of errors.

de Jonge et al. [7,8] present an approach that directly applies MBD to plan execution. Here, the authors focus on agents each having an individual plan, and on the conflicts that may arise between these plans (e.g., if they require the same resource). Diagnosis is applied to determine those factors that are accountable for *future* conflicts. The authors, however, do not take into account dependencies between health modes of actions and do not consider agents that collaborate to execute a common plan.

Kalech and Kaminka [12,13] apply *social diagnosis* in order to find the cause of an anomalous plan execution. They consider hierarchical plans consisting of so-called *behaviors*. Such plans do not prescribe a (partial) execution order on a set of actions. Instead, based on its observations and beliefs, each agent chooses the appropriate behavior to be executed. Each behavior in turn may consist of primitive actions to be executed, or of a set of other behaviors to choose from. Social diagnosis then addresses the issue of determining what went wrong in the joint execution of such a plan by identifying the disagreeing agents and the causes for their selection of incompatible behaviors (e.g., belief disagreement, communication errors).

Lesser et al. [3,11] also apply diagnosis to (multi-agent) plans. Their research concentrates on the use of a *causal model* that can help an agent to refine its initial diagnosis of a failing *component* (called a *task*) of a plan. As a consequence of using such a causal model, the agent would be able to generate a new, situation-specific plan that is better suited to pursue its goal. Diagnosis is based on observations of a component without taking into account the consequences of failures of such a component w.r.t. the remaining plan.

Witteveen et al. [18,16] show how classical MBD can be applied to plan execution. To illustrate the different aspects of diagnosis discussed in the introduction, below we present an adapted and extended version of their formalization of plan diagnosis. This formalization enables the handling of the approaches of de Jonge et al. [7,8], Kalech and Kaminka [12,13], and Lesser et al. [3,11]. The work of Birnbaum et al. [1] is not covered by the proposed formalization since it focuses on the planning activity instead of on plan execution.

3 Plans as Systems

Objects. In [18] it was shown that by using an object-oriented description of the world instead of a conventional state-based description, it becomes possible to apply classical

MBD to plan execution. Here, we will take this approach one step further by also introducing objects for agents executing the plan and for the actions themselves. Hence, we assume a finite set of objects $\mathcal{O}$ that will be used to describe the plan, the agents, the equipment and the environment.

The objects $\mathcal{O}$ are partitioned into classes or types. We distinguish four general classes, namely: *actions* $\mathcal{A}$, *agents* $\mathcal{A}g$, *equipment* $\mathcal{E}$ and *environment objects* $\mathcal{N}$.

States and partial states. Each object in $o \in \mathcal{O}$ is assumed to have a domain D_o of values. The *state* of the objects $\mathcal{O} = \{o_1, ..., o_n\}$ at some time point is described by a tuple $\sigma \in D_{o_1} \times ... \times D_{o_n}$ of values. In particular, four projections of the state σ: $\sigma^{\mathcal{A}}$, $\sigma^{\mathcal{A}g}$, $\sigma^{\mathcal{E}}$ and $\sigma^{\mathcal{N}}$ are used to denote the state of the action objects $\mathcal{A}$, the agent objects $\mathcal{A}g$, the equipment objects $\mathcal{E}$ and the environment objects $\mathcal{N}$.

The state $\sigma^{\mathcal{N}}$ of environment objects $\mathcal{N}$ describes the state of the agents' environment at some point in time. For instance, these state descriptions can represent the location of an airplane or the availability of a gate.

The states $\sigma^{\mathcal{A}}$, $\sigma^{\mathcal{A}g}$ and $\sigma^{\mathcal{E}}$ of action, agent and equipment objects respectively describe the working order of these objects. Their domains consist of the health modes of the action, agent and equipment objects. We assume that each of these domains contains at least (*i*) the value nor to denote that the action, agent and equipment objects behave normally, and (*ii*) the general fault mode ab to denote that the action, agent and equipment objects behave in an unknown and possibly abnormal way. Moreover, the domains may contain several more specific fault modes. For instance, the domain of a 'flight' action may contain a fault mode indicating that the flight is 20 minutes delayed.[2]

It will not always be possible to give a complete state description. Therefore, we introduce a *partial state* as an element $\pi \in D_{o_{i_1}} \times D_{o_{i_2}} \times \ldots \times D_{o_{i_k}}$, where $1 \leq k \leq n$ and $1 \leq i_1 < \ldots < i_k \leq |\mathcal{O}|$. We use $O(\pi)$ to denote the set of objects $\{o_{i_1}, o_{i_2}, \ldots, o_{i_k}\} \subseteq \mathcal{O}$ specified in such a state π. The value of an object $o \in O(\pi)$ in π will be denoted by $\pi(o)$. The value of an object $o \in \mathcal{O}$ not occurring in a partial state π is said to be *unknown* (or unpredictable) in π, denoted by $\perp$. Including $\perp$ in every value domain D_i allows us to consider every partial state π as an element of $D_1 \times D_2 \times \ldots \times D_{|\mathcal{O}|}$.

Partial states can be ordered with respect to their information content: given values d and d', we say that $d \leq d'$ holds iff $d = \perp$ or $d = d'$. The containment relation $\sqsubseteq$ between partial states is the point-wise extension of $\leq$: π is said to be contained in π', denoted by $\pi \sqsubseteq \pi'$, iff $\forall o \in \mathcal{O}\ [\pi(o) \leq \pi'(o)]$. Given a subset of objects $S \subseteq \mathcal{O}$, two partial states π, π' are said to be *S-equivalent*, denoted by $\pi =_S \pi'$, if for every $o \in S$, $\pi(o) = \pi'(o)$. We define the partial state π restricted to a given set S, denoted by $\pi \restriction S$, as the state $\pi' \sqsubseteq \pi$ such that $O(\pi') = S \cap O(\pi)$.

An important notion for diagnosis is the notion of *compatibility* between partial states. Intuitively, two states π and π' are said to be compatible if there is no essential disagreement about the values assigned to variables in the two states. That is, for every $o \in \mathcal{O}$ either $\pi(o) = \pi'(o)$ or at least one of the values $\pi(o)$ and $\pi'(o)$ is undefined. So we define π and π' to be compatible, denoted by $\pi \approx \pi'$, iff $\forall o \in$

[2] Note that in a more elaborate approach the value of for instance an equipment object may also indicate the location of the equipment. In this paper we only represent the health mode of the equipment.

$\mathcal{O}$ $[\pi(o) \leq \pi'(o)$ *or* $\pi'(o) \leq \pi(o)]$. As an easy consequence we have, using the notion of S-equivalent states, $\pi \approx \pi'$ iff $\pi =_{O(\pi) \cap O(\pi')} \pi'$. Finally, if π and π' are compatible states, they can be *merged* into the $\sqsubseteq$-least state $\pi \sqcup \pi'$ containing them both: $\forall o \in \mathcal{O}\ [\pi \sqcup \pi'(o) = max_{\leq}\{\pi(o), \pi'(o)\}]$.

Normality assumptions. The health mode of action, agent and equipment objects need not be known explicitly but are usually assumed to be normal: nor. Although it seems reasonable to also assume environment objects such as the weather, to be normal, we cannot make this assumption for every environment object. It makes no sense to assign the state normal to, for instance, a good to be transported. Therefore, we exclude environment objects from our normality assumption. By adding these normality assumptions to a partial state π we create a partial state $\bar{\pi}$ where $\forall o \in \mathcal{O}$ $[\bar{\pi}(o) = nor$ if $\pi(o) = \perp$ and $o \notin \mathcal{N}; \bar{\pi}(o) = \pi(o)$ otherwise$]$.

Goals . An (elementary) goal g of an agent specifies a set of states an agent wants to bring about using a plan. Here, we specify each such a goal g as a constraint, that is, a relation over some product $D_{i_1} \times \ldots \times D_{i_k}$ of domains. We say that a goal g is satisfied by a partial state π, denoted by $\pi \models g$, if the relation g contains some tuple (partial state) $(d_{i_1}, d_{i_2}, \ldots d_{i_k})$ such that $(d_{i_1}, d_{i_2}, \ldots d_{i_k}) \sqsubseteq \pi$. We assume each agent a to have a set G_a of such elementary goals $g \in G_a$. We use $\pi \models G_a$ to denote that all goals in G_a hold in π, i.e. for all $g \in G_a$, $\pi \models g$.

Action execution. Through the execution of a specific action $a \in \mathcal{A}$, the state of environment objects $\mathcal{N}$ and possibly also of equipment objects $\mathcal{E}$ may change. We describe such changes induced by a specific action (also called *plan step*) $a \in \mathcal{A}$ by a (partial) function f^{α} where α is the type of the action (also called *plan operator*) of which a is an instance.

$$f^{\alpha} : D_a \times D_{ag} \times D_{e_1} \times ... \times D_{e_i} \times D_{n_1} \times ... \times D_{n_j} \rightarrow D_{e'_1} \times ... \times D_{e'_k} \times D_{n'_1} \times ... \times D_{n'_l}$$

where $a \in \alpha \subset \mathcal{A}$ is a specific action of type α, $ag \in \mathcal{A}g$ is the execution agent, $e_1, ..., e_i \in \mathcal{E}$ are the equipment objects required, $n_1, ..., n_i \in \mathcal{N}$ are the environment objects required, and $\{e'_1, ..., e'_k, n'_1, ..., n'_l\} \subseteq \{e_1, ..., e_i, n_1, ..., n_j\}$ are equipment and environment objects that are changed by the action a. Note that since the values of equipment objects only indicate health modes of these objects we allow equipment objects to occur in the range of f^{α} in order be able to describe repair and maintenance actions.

To distinguish the different types of parameters in a more clear way, semicolons will be placed between them when specifying the function, e.g.:

$$f^{\text{transport}}(driving : \mathcal{A};\ hal : \mathcal{A}g;\ truck : \mathcal{E};\ goods : \mathcal{N}).$$

The objects whose value domains occur in $dom(f^{\alpha})$ will be denoted by $dom_O(o_a) = \{o_a, o_{ag}, o_{e_1}, ..., o_{e_i}, o_{n_1}, ..., o_{n_j}\}$ and, likewise $ran_O(o_a) = \{o_{e'_1}, ..., o_{e'_l}, o_{n'_1}, ..., o_{n'_j}\}$.

The result of an action may not always be known if, for instance, the action fails or if equipment is malfunctioning. Therefore we allow that the function associated with an action maps the value of an object to $\perp$ to denote that the effect of the action on an object is unknown.

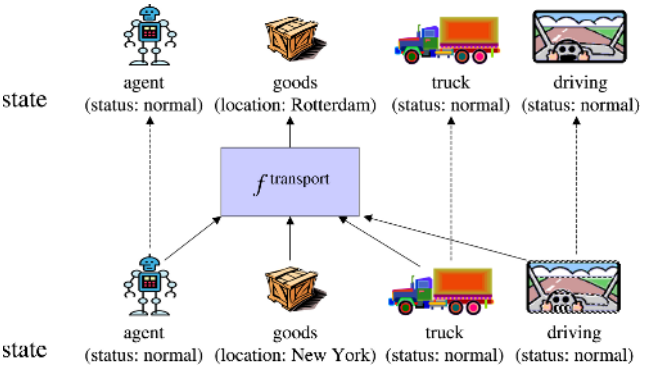


Fig. 1. An action and its state transformation

Figure 1 gives an illustration of the above outlined state transformation as result of the application of a drive action. Note that in this example only the state of the goods is changed as the result of the transport action.

Plans. A plan is a tuple $P = \langle A, < \rangle$ where $A \subseteq \mathcal{A}$ is a subset of the actions (plan steps) that need to be executed and $<$ is a partial order defined on $A \times A$ where $a < a'$ indicates that the action a must finish before the action a' may start. Note that each action $a \in A$ occurs exactly once in the plan P. We will denote the *transitive reduction* of $<$ by $\ll$, i.e., $\ll$ is the smallest sub-relation of $<$ such that the transitive closure $\ll^+$ of $\ll$ equals $<$.

We assume that if in a plan P two actions a and a' are independent, in principle they may be executed concurrently. This means that the precedence relation $<$ at least should capture all resource dependencies that would prohibit concurrent execution of actions. Therefore, we assume $<$ to satisfy the following *concurrency requirement*:

$$\text{If } ran_O(a) \cap dom_O(a') \neq \varnothing \text{ then } a < a' \text{ or } a' < a.^3$$

Figure 2 gives an illustration of a plan with one abnormally executed action. Since an action object is applied only once in a plan, for clarity reasons, *we will replace the function describing the behavior of the action by the name of the action*. The arrows relate actions to the objects it uses as inputs and the objects it modifies as its outputs. In this plan, the dependency relation is specified as $a_1 \ll a_3$, $a_1 \ll a_4$, $a_2 \ll a_4$, $a_2 \ll a_5$, $a_4 \ll a_7$, $a_5 \ll a_8$and $a_4 \ll a_6$. Note that the last dependency has to be included because a_6 changes the value of o_2 needed by a_4. The action a_4 shows that not every object occurring in the domain of an action need to be affected by the action.

Plan execution. For simplicity, we will assume that every action in a plan P takes one time unit to execute. We are allowed to observe the execution of a plan P at discrete times $t = 0, 1, 2, \ldots, k$ where k is the depth of the plan, i.e., the longest $<$-chain of actions occurring in P. Let $depth_P(a)$ be the depth of action a in plan $P = \langle A, < \rangle$.

[3] Note that since $ran_O(a) \subseteq dom_O(a)$, this requirement excludes overlapping ranges of concurrent actions, but domains of concurrent actions are allowed to overlap as long as the values of the object in the overlapping domains are not affected by the actions.

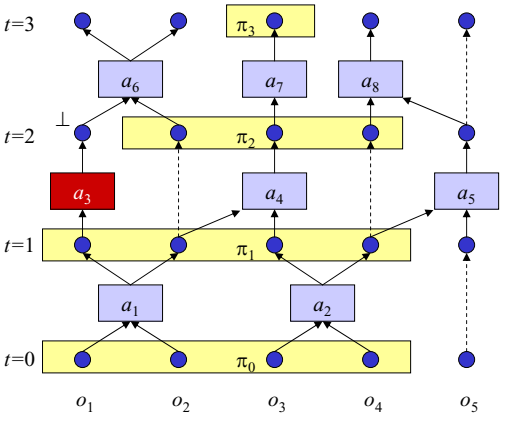


Fig. 2. Plan execution with one abnormal action

Here, $depth_P(a) = 0$ if $\{a' \mid a' \ll a\} = \varnothing$ and $depth_P(a) = 1 + max\{depth_P(a') \mid a' \ll a\}$, otherwise. If the context is clear, we often will omit the subscript P. We assume that the plan starts to be executed at time $t = 0$ and that concurrency is fully exploited, i.e., if $depth_P(a) = k$, then execution of a has been completed at time $t = k + 1$. Thus, all actions a with $depth_P(a) = 0$ are completed at time $t = 1$ and every action a with $depth_P(a) = k$ will be started at time k and will be completed at time $k+1$. Note that thanks to the above specified concurrency requirement, concurrent execution of actions having the same depth leads to a well-defined result.

A timed state is a tuple (π, t) where π is a state and $t \geq 0$ a time point. We would like to consider the predicted effect (time state) (π', t') as the result of executing plan P on a given timed state (π, t). To define this relation in a precise way, we will need the following concepts. First of all, let P_t denote the set of actions a with $depth_P(a) = t$, let $P_{>t} = \bigcup_{t'>t} P_{t'}$, $P_{<t} = \bigcup_{t'<t} P_{t'}$ and $P_{[t,t']} = \bigcup_{k=t}^{t'} P_k$. Secondly, we say that an action a is enabled in a state π if $dom_O(a) \subseteq O(\pi)$.

Now we can predict the timed state $(\pi', t+1)$ using the timed state (π, t) and the set P_t of to be executed actions. We say that $(\pi', t+1)$ is (directly) generated by execution of P from (π, t), abbreviated by $(\pi, t) \rightarrow_P (\pi', t+1)$, iff the following conditions hold:

1. $\pi'(o) = f^{\alpha}(\pi \restriction dom_O(a))(o)$ for each $a \in P_t$ such that $a \in \alpha$ and for each $o \in ran_O(a)$.
2. $\pi'(o) = \pi(o)$ for each $o \notin \bigcup_{a \in P_t} ran_O(a)$, that is, the value of any object not occurring in the range of an action in P_t should remain unchanged.
3. $\pi'(o) = \perp$ otherwise.

For arbitrary values of $t \leq t'$ we say that (π', t') is (directly or indirectly) generated by execution of P from (π, t), denoted by $(\pi, t) \rightarrow_P^* (\pi', t')$, iff the following conditions hold:

1. if $t = t'$ then $\pi' = \pi$;
2. if $t' = t + 1$ then $(\pi, t) \rightarrow_P (\pi', t')$;

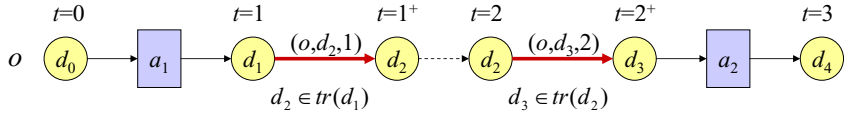


Fig. 3. A Discrete Event System of the object o

3. if $t' > t+1$ then there must exists some state $(\pi'', t'-1)$ such that $(\pi, t) \rightarrow_P^* (\pi'', t'-1)$ and $(\pi'', t'-1) \rightarrow_P (\pi', t')$.

Disruptions of plan execution. In [18,16], Witteveen et al. describe how plan execution can be diagnosed by viewing an action of a plan as a component of a system having a *normal* or an *abnormal* behavior, and by viewing the input and output objects of an action as in- and outputs of a component. This view made it possible to apply classical MBD to plan execution. In their view, a diagnosis is a subset $Q \subseteq A$ of abnormally executed actions.

Here, we will use a modified version of the plan diagnosis proposed by Witteveen et al. First of all, we define the more general notion of a *qualification* κ consisting of triples (o_j, d, t) each specifying an object o_j, the value $d \in D_{o_j}$ of the object o_j and the time point t at which the object o_j takes this value d. Such a triple might be used to specify a state change at time t of an object o_j to a value $\sigma(o_j) = d$. If the object o_j denotes an action, an agent or equipment, value d will usually indicate some fault mode.

Using qualifications, we say that $(\pi', t+1)$ is (directly) generated by execution of P from (π, t) given the qualification κ, abbreviated by $(\pi, t) \rightarrow_{\kappa;P} (\pi', t+1)$, iff the following conditions hold:

1. For each $o_j \in \mathcal{O}$: $\pi''(o_j) = d$ if $(o_j, d, t) \in \kappa$, and $\pi''(o_j) = \pi(o_j)$ otherwise.
2. $(\pi'', t) \rightarrow_P (\pi', t+1)$.

For arbitrary values of $t \leq t'$ we say that (π', t') is (directly or indirectly) generated by execution of P from (π, t) given the qualification κ, denoted by $(\pi, t) \rightarrow_{\kappa;P}^* (\pi', t')$.

An object such as an airplane may have several (fault) modes. Between these modes transitions are possible. For example, continuing to fly with an overheated engine will cause more severe damage, namely a completely ruined engine. Of course, not every transition between the (fault) modes is valid. For example, an airplane with a broken engine cannot become an airplane with only a flat tyre without repairing the engine first. Hence, we need to describe the valid state changes of objects. A *transition function* $tr_j : D_j \rightarrow 2^{D_j}$ will be used to describe the transitions that may occur.

Remark 1. Note that we can view each object o_j as representing a *Discrete Event System* [4]. The triples (o_j, d, t) in a qualification κ describe the unknown *events* that change the state of the object o_j and $tr_j : D_j \rightarrow 2^{D_j}$ is the transition function of the DES. Figure 3 gives an illustration. The goal of diagnosis is to identify these unknown events (o_j, d, t) that have caused the state changes. Also note that actions enforce state changes independent of the transition function tr_j.

The transition function make is possible to judge whether a qualification κ describes valid transitions. It places restrictions on the autonomous state changes (not caused by

actions) that may occur. Hence, we must verify whether a qualification κ induces a *sound* derivation given a plan P and the transition functions tr_j. We say that a qualification κ induces a *sound* derivation $(\pi, t) \rightarrow^*_{\kappa;P} (\pi', t')$ iff:

- for no pair of events $(o_j, d, t), (o_j, d', t') \in \kappa$, we have that $t = t'$, and
- for each event $(o_j, d, t') \in \kappa$, if $(\pi, t) \rightarrow^*_{\kappa;P} (\pi', t')$, then $d \in tr^*_j(\pi'(o_j))$.

4 Plan Diagnosis

By making (partial) observations at different time points of the ongoing plan execution we may establish that there are discrepancies between the expected and the observed plan execution. These discrepancies indicate that the results of executing one or more actions differs from the way they were planned. Identifying these actions and, if possible, what went wrong in the actions' execution will be called *primary plan diagnosis*. Actions may fail because external factors such as changes in the environmental conditions (the weather), failing equipment or incorrect beliefs of agents. These external factors are underlying causes which are important for predicting how the remainder of a plan will be executed. The *secondary plan diagnosis* aims at establishing these underlying causes.

4.1 Primary Plan Diagnosis

In [18,16], Witteveen et al. describe how plan execution can be diagnosed by viewing action instances of a plan as components of a system and by viewing the input and output objects of an action as in- and outputs of a components. Here, we will use a modified version of the plan diagnosis proposed by Witteveen et al. using a qualification κ of events $(a, d, depth(a))$ with $a \in A$. This qualification κ is called an *action qualification*.

Figure 2 gives an illustration of an execution of a plan. Suppose action a_3 is abnormal and generates a result that is unpredictable ($\perp$). Given the qualification $\kappa = \{(a_3, ab, 1)\}$ and the partially observed state π_0 at time point $t = 0$, we predict the partial states π_i as indicated in Figure 2, where $(\pi_0, t_0) \rightarrow^*_{\kappa;P} (\pi_i, t_i)$ for $i = 1, 2, 3$. Note that since the value of o_1 and of o_5 cannot be predicted at time $t = 2$, the result of action a_6 and of action a_8 cannot be predicted and π_3 contains only the value of o_3.

Suppose now that we have a (partial) observation $obs(t) = (\pi, t)$ of the state of the world at time t and an observation $obs(t') = (\pi', t')$ at time $t' > t \geq 0$ during the execution of the plan P. We would like to use these observations to infer the health states of the actions occurring in P. Assuming a normal execution of P, we can (partially) predict the state of the world at a time point t' given the observation $obs(t)$: if all actions behave normally, denoted by $\bar{\pi}$, we predict a partial state $\pi'_\varnothing$ at time t' such that $(\bar{\pi}, t) \rightarrow^*_P (\pi'_\varnothing, t')$. Since we do not require observations to be made systematically, $O(\pi')$ and $O(\pi'_\varnothing)$ might only partially overlap. Therefore, if this assumption holds, the values of the objects that occur in both the predicted state and the observed state at time t' should match, i.e; we should have $\pi' \approx \pi'_\varnothing$. If this is not the case, the execution of some actions must have gone wrong and we have to determine an action qualification

κ such that the predicted state derived using κ agrees with π'. This is nothing else then a straight-forward extension of the diagnosis concept in MBD to plan diagnosis (cf. [14,6]).

Definition 1. *Let $P = \langle A, < \rangle$ be a plan with observations $obs(t) = (\pi, t)$ and $obs(t')$ $= (\pi', t')$, where $t < t' \leq depth(P)$ and let the action qualification κ be a set of triples $(a, d, depth(a))$ with $a \in \mathcal{A}$ and $d \in D_a$. Moreover, let κ induces a* sound *derivation $(\bar{\pi}, t) \rightarrow^*_{\kappa;P} (\pi'_\kappa, t')$ given the plan P and the transition functions $tr_j : D_j \rightarrow 2^{D_j}$ for each action $a_j \in A$.*

Then κ is said to be a primary plan diagnosis *(action diagnosis) of $\langle P, obs(t), obs(t') \rangle$ iff $\pi' \approx \pi'_\kappa$.*

So in a primary plan diagnosis κ, the observed partial state (π') at time t' and the predicted state (π'_κ) at time t' assuming the action qualification κ agree upon the values of all objects $O(\pi') \cap O(\pi'_\kappa)$ occurring in both states.

Consider again Figure 2 and suppose that we did not know that action a_3 was abnormal and that we observed $obs(0) = ((d_1, d_2, d_3, d_4), 0)$ and $obs(3) = ((d'_1, d'_3, d'_5), 3)$. Using the normal plan derivation relation starting with $obs(0)$ we will predict a state π'_κ at time $t = 3$ where $\pi'_\kappa = (d''_1, d''_2, d''_3)$. If everything is ok ($\kappa = \varnothing$), the values of the objects predicted as well as observed at time $t = 3$ should correspond, i.e. we should have $d'_j = d''_j$ for $j = 1, 3$. If, for example, only d'_1 would differ from d''_1, then we could qualify a_6 as abnormal, since then the predicted state at time $t = 3$ using $\kappa = \{(a_6, ab, 2)\}$ would be $\pi'_\kappa = (d''_3)$ and this partial state agrees with the observed state on the value of o_3.

Note that for all objects in $O(\pi') \cap O(\pi'_\kappa)$, the qualification κ provides an *explanation* for the observation π' made at time point t'. Hence, for these objects the qualification provides an *abductive diagnosis* [5]. For all observed objects in $O(\pi') - O(\pi'_\kappa)$, no value can be predicted given the qualification κ. Hence, by declaring them to be unpredictable, possible conflicts with respect to these objects if a normal execution of all actions is assumed, are resolved. This corresponds with the idea of a *consistency-based diagnosis* [14].

4.2 Secondary Plan Diagnosis

Actions may fail because of unforeseen (environmental) conditions such as being struck by lightning, malfunctioning equipment or incorrect beliefs of agents. Diagnosing these secondary causes is more difficult since weather, equipment and agents may play a role in the execution of several actions. Moreover, objects such as equipment and weather may go through several unforeseen state changes.

A *secondary qualification* κ consists of triples (o_j, d, t) where $o_j \in \mathcal{O} - \mathcal{A}$ is an object that changes to the value $d \in D_j$ at time point t. Usually we choose for the time point t the depth $depth(a)$ of the first action instance where change manifests itself. So, for some action a, $t = depth(a)$ and $o_j \in dom_O(a)$.

Definition 2. *Let $P = \langle A, < \rangle$ be a plan with observations $obs(t) = (\pi, t)$ and $obs(t')$ $= (\pi', t')$, where $t < t' \leq depth(P)$ and let the action qualification κ be a set of triples (o, d, t) with $o \in \mathcal{O} - \mathcal{A}$ and $d \in D_o$. Moreover, let κ induces a* sound *derivation*

$(\bar{\pi}, t) \rightarrow^*_{\kappa;P} (\pi'_\kappa, t')$ *given the plan* P *and the transition functions* $tr_j : D_j \rightarrow 2^{D_j}$ *for each object* $o_j \in \mathcal{O}$.

Then the qualification κ *is said to be a* secondary plan diagnosis *of* $\langle P, obs(t), obs(t')\rangle$ *iff* $\pi' \approx \pi'_\kappa$.

The secondary diagnosis can be divided into *agent*, *equipment* and *environment diagnosis* depending on whether the object o in a triple $(o, d, t) \in \kappa$ belongs to $\mathcal{A}g$, $\mathcal{E}$ or $\mathcal{N}$ respectively. Note that agent diagnosis is related to *social diagnosis* described by Kalech and Kaminka [12,13] if the agents' health modes are used to describe the agents' incorrect beliefs.

Predicting the future. Secondary diagnosis offers an important advantage over primary diagnosis. First, secondary diagnosis enables us to the determine which future actions may also be affected by the malfunctioning agents and equipment, and by unforeseen state changes in the environment.

Definition 3. *Let* t *be the current time point and let* κ *be a secondary diagnosis of the plan executed sofar. Then the set of future actions that will directly be affected given the current diagnosis* κ *is:*

$$\{a \in \mathcal{A} \mid (o_j, d, t') \in \kappa, o_j \in dom_O(a), d \neq nor, depth(a) \geq t \geq t'\}$$

Second, besides identifying the actions that will be affected, we can also determine the goals that can still be reached.

Definition 4. *Let* t *be the current time point, let* π *current partial state and let* κ *be an secondary diagnosis of the plan executed sofar. Moreover, let* $(\pi, t) \rightarrow_{\kappa;P} (\pi', depth(P))$. *Then the set of goals that can still be realized is given by:* $\{g \in G \mid \pi' \models g\}$

Responsible agents. Besides knowing the underlying cause of plan execution failures, it is also important to know the agents responsible for the failures. To illustrate this, reconsidering the example in the introduction where the agent responsible for the belly landing can be the pilot agent, the maintenance agent, or the airline agent that reduced the maintenance budget.

Here we will present a very simple model of responsibility. We introduce a responsibility function $res : (\mathcal{O} - \mathcal{N}) \rightarrow \mathcal{A}g$ specifying the agent that is responsible for each of the action, agent and equipment objects.

Definition 5. *Let* κ *be any diagnosis of a plan execution and let* $res : (\mathcal{O} - \mathcal{N}) \rightarrow \mathcal{A}g$ *be a responsibility function.*

Then for each event $(o, d, t) \in \kappa$, *the responsible agent is determined by:* $res(o)$.

5 Conclusion

This paper describes a generalization of the model for plan diagnosis as presented in [18,16]. New in the current approach is (*i*) the introduction of primary and secondary diagnosis, and (*ii*) the introduction of objects representing actions, agents and equipment. The primary diagnosis identifies failed actions and possibly in which way they failed while the secondary diagnosis addresses the causes for action failures. The latter

is an improvement over the plan diagnosis presented in [18,16], where only dependencies between action failures could be described using causal rules. An additional feature of the proposed approach is that all objects can be modeled as discrete events systems. This enables the description of the unknown dynamic behavior of objects such as equipment over time. The secondary diagnosis then identifies the unknown state changes of objects and possibly the agents that can be held responsible for the state changes.

References

1. L. Birnbaum, G. Collins, M. Freed, and B. Krulwich. Model-based diagnosis of planning failures. In *AAAI 90*, pages 318–323, 1990.
2. A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90:281–300, 1997.
3. N. Carver and V.R. Lesser. Domain monotonicity and the performance of local solutions strategies for cdps-based distributed sensor interpretation and distributed diagnosis. *Autonomous Agents and Multi-Agent Systems*, 6(1):35–76, 2003.
4. C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.
5. L. Console and P. Torasso. Hypothetical reasoning in causal models. *International Journal of Intelligence Systems*, 5:83–124, 1990.
6. L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7:133–141, 1991.
7. F. de Jonge and N. Roos. Plan-execution health repair in a multi-agent system. In *PlanSIG 2004*, 2004.
8. F. de Jonge, N. Roos, and H.J. van den Herik. Keeping plan execution healthy. In *Multi-Agent Systems and Applications IV: CEEMAS 2005, LNCS 3690*, pages 377–387, 2005.
9. D. McDermott et al. The pddl planning domain definition language. In *The AIPS-98 Planning Competition Committee*, 1998.
10. R. E. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 5:189–208, 1971.
11. Bryan Horling, Brett Benyo, and Victor Lesser. Using Self-Diagnosis to Adapt Organizational Structures. In *Proceedings of the 5th International Conference on Autonomous Agents*, pages 529–536. ACM Press, 2001.
12. M. Kalech and G. A. Kaminka. On the design ov social diagnosis algorithms for multi-agent teams. In *IJCAI-03*, pages 370–375, 2003.
13. M. Kalech and G. A. Kaminka. Diagnosing a team of agents: Scaling-up. In *AAMAS 2004*, 2004.
14. R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.
15. N. Roos, A. ten Teije, and C. Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. In *AAMAS 2003*, pages 655–661, 2003.
16. N. Roos and C. Witteveen. Diagnosis of plans and agents. In *Multi-Agent Systems and Applications IV: CEEMAS 2005, LNCS 3690*, pages 357–366, 2005.
17. H. Tonino, A. Bos, M. de Weerdt, and C. Witteveen. Plan coordination by revision in collective agent based systems. *Artificial Intelligence*, 142:121–145, 2002.
18. C. Witteveen, N. Roos, R. van der Krogt, and M. de Weerdt. Diagnosis of single and multi-agent plans. In *AAMAS 2005*, pages 805–812, 2005.

Framework and Complexity Results for Coordinating Non-cooperative Planning Agents

J. Renze Steenhuisen[1], Cees Witteveen[1], Adriaan W. ter Mors[1,2], and Jeroen M. Valk[2]

[1] Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, P.O. Box 5031, 2600 GA Delft, The Netherlands, tel.: +31 15 278 7486, fax +31 15 278 6632, {J.R.Steenhuisen, C.Witteveen, A.W.terMors}@tudelft.nl

[2] Almende, Westerstraat 50, 3016 DJ Rotterdam, The Netherlands, tel.: +31 10 404 9444, fax +31 10 404 7773, {Adriaan, Jeroen}@almende.nl

Abstract. In multi-agent planning problems agents are requested to jointly solve a complex task consisting of a set of interrelated tasks. Since none of the agents is capable to solve the whole task on its own, usually each of them is assigned to a subset of tasks. If agents are dependent upon each other via interrelated tasks they are assigned to, moderately coupled teams of agents are called for. Such teams solve the task by coordinating during or after planning and revising their plans if necessary. In this paper we show that such complex tasks also can be solved by loosely coupled teams of agents that are able to plan independently, although the computational complexity of the coordination problems involved is high. We also investigate some of the factors influencing this complexity.

Keywords: Multi-agent system, complex tasks, task assignment, planning, coordination, computational complexity.

1 Introduction

A multi-agent planning problem requires a set of autonomous planning agents to come up with a *joint plan* for achieving a set of tasks. Usually, none of the participating agents is capable of solving all tasks by itself. Therefore, each agent is assigned a subset of tasks to carry out and each agent has to construct a plan to carry out the tasks assigned to it. Obviously, it is required that these individual plans are compatible with each other in the sense that together they should constitute a feasible plan for the complete set of tasks. Therefore, we need some form of *coordination* between the agents.

Like in [1,2], we classify multi-agent planning problems according to the type of coordination needed. This classification is based on a schematic partitioning of multi-agent planning problems into three phases: A *task-allocation* phase where it is decided who does what, a *planning* phase where it is decided how to do it, and a *task-execution* phase where it is done.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 98–109, 2006.

Problems where coordination is only needed in the task-allocation phase are problems that can be solved by *loosely-coordinated agents* that are able to plan and act autonomously, but need to coordinate with respect to the task allocation (e.g., negotiating about the subtasks to be assigned). Typical tasks that can be solved in this way are reconnaissance tasks and simple pick-up and delivery tasks. If coordination is also needed in the planning phase, such problems are said to be solvable by *moderately-coordinated agents*. Such agents need to coordinate in the pre-execution phase, but are not dependent upon each other if the plan is executed. Typical problems in this category are monitoring and multi-modal transportation tasks. Finally, problems where coordination between agents is needed in all three phases are problems that can be solved by *tightly-coordinated agents*. These agents need coordination not only in the preparation but also in the execution phase. Examples are platooning vehicles and moving in formation.

Most approaches to coordination in multi-agent planning, like [3, 4, 5], stress the intertwining of planning and coordination processes allowing the agents to revise their plans by exchanging information during the planning process. Other approaches, like [6, 7, 8], consider coordination as an after-planning process to either remove conflicts between independently developed plans or to improve such plans by exploiting positive interactions between them.

In both these approaches it is assumed that due to coordination requests – either during or after planning– individual agents are prepared to adapt and revise their current (partial) plans in order to obtain a feasible joint plan.

In this paper, we concentrate on solving multi-agent planning problems where agents are self-interested and non-cooperative. Typical problems we are interested in are solving complex tasks that require the joint effort of a set of agents, but where interaction between the agents during planning and execution is absent. Examples of such problems are multi-modal transportation problems that have to be solved by the joint effort of competitive transportation organizations and patient planning in hospitals where self-interested and independent parties are involved in scheduling health-care resources. In all these applications, we have to assume that *(i)* the agents do not want to be interfered during planning and *(ii)* agents do not want to revise their plans afterwards. We will call the problem of ensuring that a feasible joint plan will be created, while taking into account *(i)* and *(ii)*, the *plan-coordination problem*. Clearly, both approaches mentioned above are not suitable to solve this plan-coordination problem.

At first sight seems that the plan-coordination problem can only be used to solve task planning problems that require loosely-coordinated agents, since in that case the planning can be achieved by the agents independently. In such cases, it is often sufficient to deal with coordination in the task-assignment (and task decomposition) process, such that the plan-coordination problem is solved in a trivial way.

In this paper, however, we will show that the plan-coordination problem can also be used to solve tasks that require moderately-coordinated agents. We will develop a general framework for representing and solving the task assignment and plan-coordination problem. The main idea we apply here is that a set of

tasks requiring moderately-coupled agents to solve them can be *reduced* to a set of tasks requiring loosely-coupled agents. This reduction ensures that the set of independently developed plans constructed for the latter set is also a solution for the former set of tasks. We will use this framework to describe this reduction and to establish some relations between properties of tasks, properties of agents, and the complexity of the resulting plan-coordination problems.

Significance and Perspectives The approach we propose can be viewed upon from different perspectives. First, it is an attempt to partially integrate the research areas of task assignment, coalition formation, and multi-agent planning. In particular, we will show that in solving moderately-coupled tasks, one has to recognize the interaction effects between individual planning methods of agents and the task-assignment methods employed, since the outcome of one agent is not only determined by the set of tasks it receives but also by the plans developed by other agents. We will show that such interaction effects have consequences for the computational complexity of the plan-coordination problem.

Second, our approach can be viewed as an attempt to connect other research on coordinating agents like *social laws* [9,10] and *cooperation protocols* [11] with multi-agent task-based planning research. Social laws are general rules that govern agent behavior. If a set of agents abide by these rules, then their behavior will be coordinated without the need for any problem-specific information exchange between the agents. In many situations, however, coordination cannot be achieved (or not efficiently) through general, problem-independent rules alone. In such cases, cooperation protocols can be applied that require simple forms of problem-specific information exchange before the agents can start planning. These protocols guarantee that if the agents adhere to the protocol, then the individual plans can easily be assembled into a feasible joint plan for the overall task. Our approach to solve plan-coordination problems for moderately-coupled agents comes down to reducing these problems to plan-coordination problems for loosely-coupled agents that agree upon cooperation protocols.

Finally, our approach can be viewed upon from a broader computational perspective. Note that tasks requiring loosely-coupled coordination allow the agents to plan independently. Hence, they allow the multi-agent planning problem to be partitioned into independent planning subproblems. For moderately-coupled plan-coordination problems, such a partitioning is not possible. As we will show, solving moderately-coupled plan-coordination problems for self-interested agents comes down to transforming moderately-coupled plan-coordination problems into loosely-coupled plan-coordination problems in such a way that the original problem is changed in a minimal way. In other words, our approach is an example of applying a *minimal change* and a *task decomposition* approach to solve a plan-coordination problem.

2 Framework

The formal framework we present is intended to capture the basic aspects of task-based planning and coordination of non-cooperative agents. It enables us

to distinguish the main components of the plan-coordination problem –and their interactions– that we are interested in:

1. a *complex task* that requires the joint effort of several agents to complete it,
2. a *task-assignment* process by means of which each agent obtains a subset of tasks to complete,
3. a *planning* process enabling each agent to complete its subset of tasks, and
4. a *plan-coordination* mechanism by means of which the completion (if possible) of the original complex task can be ensured.

Complex Task. We consider a set of agents $\mathcal{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ that have to complete a complex task $\mathcal{T}$. Such a complex task is defined as a set of tasks $T = \{t_1, \ldots, t_k\}$ together with two binary relations between them. One of these relations is the *decomposition relation* ρ, relating an abstract task t to a set of more primitive subtasks.[1] Therefore, we define an OR-decomposition relation $\rho_\vee$ to model this type of relation between an abstract task t and its (more primitive) subtasks t_i. On the other hand, running a factory requires multiple tasks to be done, such as the purchasing of goods and hiring of personnel, and optimizing production lines. To model this relation, we define an AND-decomposition relation $\rho_\wedge$ between an abstract task t and a set of subtasks t_i.[2]

The second relation is a precedence relation $\prec$ among tasks, where $t \prec t'$ means that t has to be completed before t' can start. The set of precedence constraints induces a *partial order* on the tasks in T.

In addition, an agent can only complete a task when it is capable of carrying it out at all. Therefore, capabilities need to be associated with both tasks and agents. These capabilities are incorporated by defining capability vectors for both the agents $\boldsymbol{c}(\mathcal{A}_i)$ and the tasks $\boldsymbol{c}(t_j)$. We postpone their detailed description to a separate paragraph about Task Assignment below.

We now formally define a complex task as $\mathcal{T} = (T, \rho, \prec, \boldsymbol{c}(T))$ thereby extending the concept of a *task tree* [12, 13]. In this tuple, T represents the set of abstract and primitive tasks, ρ is the decomposition relation, $\prec$ a precedence order, and $\boldsymbol{c}(T)$ is the abbreviation for the set of task capability vectors.

In Figure 1, an example is shown of a complex task and its decomposition into subtasks with precedence relations

In a complex task $\mathcal{T} = (T, \rho, \prec, \boldsymbol{c}(T))$, we require that the decomposition $\rho(t)$ of a task t is unique and that it is either completely in $\rho_\vee$, or completely in $\rho_\wedge$ (i.e., $\rho_\vee \cap \rho_\wedge = \emptyset$). Note that this does not limit the expressiveness of our framework with respect to tasks that can be decomposed by a combination of OR and AND decomposition (e.g., in Figure 1, we have $\rho(t_1) = \{t_{11}, t_{12}\} \subseteq \rho_\vee$ with $\rho(t_{11}) = \{t_{111}, t_{112}\} \subseteq \rho_\wedge$).

[1] For instance, when goods need to be transported from a harbor to a factory we have an abstract task *transport*. Such an abstract task could be completed by carrying out one of the more primitive subtasks *transport_by_train*, *transport_by_truck*, and *transport_by_ship*.

[2] In [12], a similar decomposition of complex tasks is used.

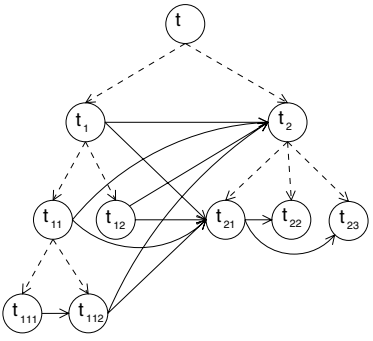


Fig. 1. An example complex task decomposed (dashed arcs) into subtasks with precedence relations (normal arcs) between them

Moreover, the decomposition and precedence relation are related in two ways. First, the relations are *orthogonal*, which means that precedence relations only exist between tasks that have no decomposition relation between them (i.e., $\rho^+ \cap (\prec \cup \prec^c)^+ = \emptyset$).[3] Second, precedence relations are inherited via decomposition (i.e., if $t_1 \prec t_2$, then for all $t_1' \in \rho(t_1)$ and for all $t_2' \in \rho(t_2)$ we have that $t_1' \prec^+ t_2$, $t_1 \prec^+ t_2'$, and $t_1' \prec^+ t_2'$).

Having defined the hierarchical decomposition of a set of tasks, we now define a *task network* (T, ρ) with $\rho = \rho_\vee \cup \rho_\wedge$, to be able to express when a task and a set of tasks has been completed. We say that a task $t \in T$ in a task network (T, ρ) is completed if exactly one of the following conditions holds:

1. t has been completed directly,
2. a task $t' \in \rho(t) \subseteq \rho_\vee$ has been completed, or
3. all tasks $t' \in \rho(t) \subseteq \rho_\wedge$ have been completed.

This notion naturally extends to the completion of a task network. A task network (T, ρ) is said to be completed if all *initial tasks* in (T, ρ) have been completed, that is if all tasks in the set $\{t \mid \rho^c(t) = \emptyset\}$ have been completed. Note that this framework differs from others in the sense that it does not restrict completion to completing the set of leaf tasks $\{t \mid \rho(t) = \emptyset\}$.

Task Assignment Now that we have defined when a task network (T, ρ) associated with a complex task $\mathcal{T}$ is completed, we can deal with the problem of which agent is going to complete which task.

First, an individual agent $\mathcal{A}_i$ must have the required *capabilities* to be able to carry out a certain task $t \in T$. We assume that in the entire multi-agent system, m distinct capabilities $c_1, \ldots, c_m$ can be distinguished. The capabilities of agent $\mathcal{A}_i$ are represented by the vector $\boldsymbol{c}(\mathcal{A}_i) = (c_1(\mathcal{A}_i), \ldots, c_m(\mathcal{A}_i)) \in (\mathbb{N} \cup \{\infty\})^m$, where $c_j(\mathcal{A}_i)$ specifies how much agent $\mathcal{A}_i$ can offer of capability c_j (we will

[3] The following notations are used on a binary relation σ: transitive closure σ^+, transitive reduction σ^-, and the converse σ^c.

assume integral quantities). Similarly, $\boldsymbol{c}(t_j) = (c_1(t_j), \ldots, c_m(t_j)) \in \mathbb{N}^m$ is the vector that specifies how much of each capability is required for carrying out task $t_j \in T$. An agent $\mathcal{A}_i$ is said to be able to carry out a subset of tasks $T_i \subseteq T$ iff $\boldsymbol{c}(\mathcal{A}_i) \geq \sum_{t \in T_i} \boldsymbol{c}(t)$ (where $\boldsymbol{x} \geq \boldsymbol{y}$ iff for all $i = 1, \ldots, m$, $x_i \geq y_i$).[4] In the following, the set of agent capability vectors and the set of task capability vectors are abbreviated by $\boldsymbol{c}(\mathcal{A})$ and $\boldsymbol{c}(T)$, respectively.

A task instance where the agents are not already assigned to tasks is called a *free task instance* and is specified as a tuple $(T, \rho, \prec, \mathcal{A}, \boldsymbol{c}(\mathcal{A}), \boldsymbol{c}(T))$. Such an instance specifies the tasks, their decomposition relation, their order dependencies, the task requirements, and the agent capabilities. To complete the task network (T, ρ) associated with the free task instance, individual tasks $t \in T$ have to be assigned to agents. Therefore, we need to define which sets of tasks can be assigned to agents in order to complete (T, ρ). Such a set $T' \subseteq T$ is called a *candidate-assignment set* and has to satisfy the following constraints:

1. T' is a ρ^+-independent subset of T: if $t, t' \in T'$ then neither $t \rho^+ t'$ nor $t' \rho^+ t$,
2. for all $t \in T$, if $t' \in \rho(t)$ and $\rho(t) \subseteq \rho_\vee$ then $\rho(t) \cap T' = \{t'\}$, and
3. (T, ρ) is completed by completing all tasks in T'.

In Figure 1, the set $T' = \{t_{111}, t_{112}, t_2\}$ is a candidate-assignment set. Note that a candidate-assignment set does not have strict supersets nor strict subsets that are candidate-assignment sets. Moreover, if $\rho = \emptyset$, there is only one unique candidate-assignment set (i.e., $T' = T$).

A partitioning $\{T_i\}_{i=1}^n$ of a candidate-assignment set T' with every $t \in T'$ assigned to an agent $\mathcal{A}_i$ capable of carrying it out is called an *assignment set* if

1. the set $\bigcup_{i=1}^n T_i$ is a candidate-assignment set, and
2. every agent $\mathcal{A}_i$ is capable of completing all tasks in its assigned partition T_i.

Applying an assignment set to a free task instance $(T, \rho, \prec, \mathcal{A}, \boldsymbol{c}(\mathcal{A}), \boldsymbol{c}(T))$ results in a *fixed task instance* $(\{T_i\}_{i=1}^n, \prec, \mathcal{A}, \boldsymbol{c}(\mathcal{A}), \boldsymbol{c}(T))$. The agents $\mathcal{A}_i$ are characterized by the blocks of the partitioning $\{T_i\}_{i=1}^n$ which are assumed, without loss of generality, to be non-empty. Additionally, the decomposition relation and the capabilities are no longer needed, because each agent is able to complete the tasks assigned to it which are ρ-independent. Therefore, fixed task instances will be abbreviate by the tuple $(\{T_i\}_{i=1}^n, \prec)$ in the remainder of this text.

Planning As the result of task assignment, the set of precedence constraints $\prec$ in a fixed task instance $(\{T_i\}_{i=1}^n, \prec)$ is split into two disjoint subsets:[5]

1. the set of *intra-agent* constraints $\prec_{intra} = \bigcup_{i=1}^n \prec_i = \bigcup_{i=1}^n (\prec^+ \cap (T_i \times T_i))^-$, contains all precedence constraints between tasks assigned to agent $\mathcal{A}_i$, and

[4] If $c_j(\mathcal{A}_i)$ is finite, the capability is said to be a *consumable* resource (e.g., fuel, time, or money). If $c_j(\mathcal{A}_i) = \infty$, it is a *non-consumable* capability (e.g., knowledge or a skill).

[5] We would like to present these relations as concisely as possible. Therefore, we will use a transitive reduction of a transitively closed relation whenever it is possible.

2. the set of *inter-agent* constraints $\prec_{inter} = (\prec^{+} \cap \bigcup_{i \neq j}(T_i \times T_j))^{-}$, contains all precedence constraints between tasks assigned to different agents.

Note that each agent $\mathcal{A}_i$ has to complete its part $(T_i, \prec_i)$ of the complex task, which is generated by the set of tasks T_i assigned to it. In order to complete $(T_i, \prec_i)$, each agent has to construct a plan (or schedule) for it. Such a plan can be represented as a partial ordering of the tasks T_i that satisfies the precedence constraints $\prec_i$. Irrespective of the (possibly domain-specific) planning tools employed by $\mathcal{A}_i$, we will therefore assume that the resulting plan always can be represented by a partial order $P_i = (T_i, \pi_i)$ with $\prec_i \subseteq \pi_i$.

From an individual agent's point of view, it wants to be completely autonomous in developing its plan P_i and unwilling to revise it afterwards. Therefore, we define a *joint plan* for a set of agents $\mathcal{A}_i$ on a fixed task instance $(\{T_i\}_{i=1}^{n}, \prec)$ as a plan $P = (\{T_i\}_{i=1}^{n}, \pi)$ where

1. π respects $\prec$, that is $\prec \subseteq \pi$, and
2. each individual plan $P_i = (T_i, \pi_i)$ of agent $\mathcal{A}_i$ is respected, that is, $\pi_i \subseteq (\pi \cap (T_i \times T_i))$.

Clearly, if such a joint plan exists, it implies that the current plans of all the agents are coordinated. There is no need for any revision of the individual plans in executing the joint plan.

Plan Coordination The existence of such a joint plan is by no means guaranteed: Due to the set of inter-agent precedence constraints, combinations of the individual plans together with this set can lead to breaking of the partial order as is shown in the following simple example.

Example 1. In Figure 2, a simple situation is depicted with four agents $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$, $\mathcal{A}_4$ being assigned two tasks each of two complex tasks. It is clear that multiple combinations of individual plans of agents lead to inter-agent cycles. For example, if $\mathcal{A}_1$ chooses a plan where $t_8 \prec t_1$ and $\mathcal{A}_3$ chooses a plan where $t_3 \prec t_6$, there exists such a cycle. Hence, the possibility of such plans prevents the existence of a joint plan respecting every individual plan.

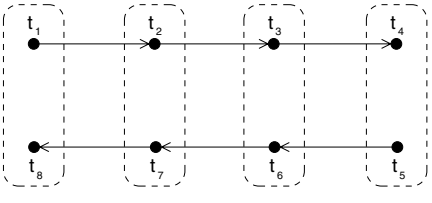


Fig. 2. Problems can occur when planning autonomously

The plan-coordination problem now can be stated as follows: How can we guarantee that *every* possible set of individual plans P_i can be easily combined into a feasible joint plan, without the need to revise the individual plans? In fact, this means that we want to have a solution to a moderately-coupled plan-coordination problem –there are constraints between the tasks assigned to different agents– without needing the agents to coordinate during planning.

A given fixed task instance $(\{T_i\}_{i=1}^n, \prec)$ is said to be *coordinated*, if it holds that for every set of individual plans $\{P_i = (T_i, \pi_i)\}_{i=1}^n$ constructed by the agents there exists a joint plan respecting the individual plans P_i. In [14], it was shown that every fixed task instance can be transformed into a coordinated one by adding a minimum number of intra-agent constraints $\Delta = \bigcup_{i=1}^n \Delta_i$ such that the resulting instance $(\{T_i\}_{i=1}^n, \prec \cup \Delta)$, is a coordinated instance.

In Figure 2, the reader might check that $\Delta = \{t_7 \prec t_2, t_6 \prec t_3, t_5 \prec t_4\}$ is such a set of additional constraints that turns the fixed task instance into a coordinated one. The problem, of course, is to determine how difficult it is to find such a set of additional constraints.

3 The Computational Complexity of Coordination

In this section, we will analyze the computational complexity of plan coordination. We study three variants of the plan-coordination problem and some factors that influence their complexity. Some results have been published in [14]; the results about the complexity of coordination when the number of agents is kept fixed are new.

3.1 Variant I: Pure Coordination

We start with analyzing the complexity of coordination in multi-agent planning isolated from the task-assignment process and define the following decision problem for fixed task instances:

Pure Coordination Recognition (PCR)
INSTANCE: Given a fixed task instance $(\{T_i\}_{i=1}^n, \prec)$.
QUESTION: Is this instance coordinated?

We have shown that PCR is coNP -complete [15], by a non-trivial reduction from the complement of the Path With Forbidden Pairs (PWFP) problem.

While PCR only asks whether a fixed task instance is coordinated, it does ask for the existence of a bounded coordination set as in the following problem:

Pure Coordination (PC)
INSTANCE: Given a fixed task instance $(\{T_i\}_{i=1}^n, \prec)$ and integer $K \geq 0$.
QUESTION: Does there exist a coordination set Δ with $|\Delta| \leq K$ such that the fixed task instance $(\{T_i\}_{i=1}^n, \prec \cup \Delta)$ is coordinated?

Intuitively, guessing a coordination set Δ, we can verify in polynomial time using a PCR-oracle whether the instance $(\{T_i\}_{i=1}^n, \prec \cup \Delta)$ is coordinated. Since PCR is coNP -complete, it follows that PC $\in \Sigma_2^p$. It turns out that PC is Σ_2^p-complete, using a reduction from a quantified version of PWFP [15].

Factors Influencing the Complexity of Pure Coordination It seems reasonable to assume that one source of complexity of the plan-coordination problem can be attributed to the *number of tasks* each agent receives and –indirectly– to the complexity of the single-agent planning problems for these tasks. However, it turns out that the PC problems remain intractable even if the single-agent planning problems are trivial (see Table 1). Notice that the Σ_2^p-completeness of PC is still open for instances where each agent has at most $4, 5$, or 6 tasks.

Table 1. Complexity of PCR and PC with limited number of tasks per agent.

	$\|T_i\| = 2$	$\|T_i\| \leq 3$	$\|T_i\| \leq 4$	$\|T_i\| \leq 7$
PCR	P	P	coNP -complete	coNP -complete
PC	NP -complete	NP -complete	Σ_2^p	Σ_2^p-complete

Due to limited space, we will merely give hints on how these results are obtained. All proofs of these and other complexity results can be found in [16]. First, for PC with $|T_i| = 2$ a reduction can be made from Feedback Vertex Set. For PCR with $|T_i| \leq 3$, a reduction can be made to topological sorting. The completeness of PCR with $|T_i| \leq 4$, and PC with $|T_i| \leq 7$, is derived from the properties of the reductions used in the completeness proofs of the general variants.

Another source of complexity might be the number of agents involved. Indeed, if we limit the number of agents, it can be shown that the PCR problem is in P for any fixed number of agents. This can be proven by reducing the problem to simple inter-agent cycle-testing. Detecting such a cycle can be achieved in polynomial time [16]. As an easy consequence, the associated PC-problems are in NP . They turn out to be NP -complete for all fixed values $|\mathcal{A}| = n \geq 3$, which can be proven by reduction from 3-Partite Vertex Cover [16]. Although we suspect PC for $n = 2$ to be tractable, the complexity of this problem is still open.

Table 2. Complexity of PCR and PC with fixed number of agents.

	$\|\mathcal{A}\| = 2$	$3 \leq \|\mathcal{A}\|$
PCR	P	P
PC	NP	NP -complete

3.2 Variant II: Coordinated Assignment

If a task-assignment problem is part of the coordination problem, we define

Coordinated Assignment Recognition (CAR)
INSTANCE: Given a free task instance $(T, \rho, \prec, \mathcal{A}, \boldsymbol{c}(\mathcal{A}), \boldsymbol{c}(T))$.
QUESTION: Does there exist an assignment set $\{T_i\}_{i=1}^n$ such that the resulting fixed task instance $(\{T_i\}_{i=1}^n, \prec)$ is coordinated?

The CAR problem turns out to be Σ_2^p-complete [14]. Note that here task-assignment problems seem to constitute an independent factor of complexity as the total complexity goes up one step in the polynomial hierarchy compared to PCR. Note that checking a candidate-assignment set for a free task-instance is polynomially verifiable, both for the case where agents have consumable capabilities as well as the case where agents have non-consumable capabilities. However, the problem of deciding whether there exists a suitable task assignment for a free task instance is NP -hard for consumable capabilities[6] but polynomially solvable for non-consumable capabilities. Therefore, one might expect that the consumability of capabilities would influence the complexity of CAR. This turns out not to be the case: The CAR problem turns out to be Σ_2^p-complete for both assignment conditions.

Since adding the task-assignment problem resulted in moving one step up in the polynomial hierarchy for the coordination recognition problem, one would expect the same for the coordination problems themselves. However, this is not the case for the following problem, where the task-assignment problem is added to the pure coordination problem.

Coordinated Assignment (CA)
INSTANCE: Given a free task instance $(T, \rho, \prec, \mathcal{A}, \boldsymbol{c}(\mathcal{A}), \boldsymbol{c}(T))$ and a K.
QUESTION: Does there exist an assignment set $\{T_i\}_{i=1}^n$ and a coordination set $\Delta \subseteq \bigcup_{i=1}^n (T_i \times T_i)$ with $|\Delta| \leq K$ such that the resulting fixed task instance $(\{T_i\}_{i=1}^n, \prec \cup \Delta)$ is coordinated?

Note that it suffices to guess both an assignment and a coordination set Δ to verify in polynomial time using a PCR-oracle that the given instance is a yes-instance. Therefore, the problem cannot be harder than the PC problem. In fact, it turns out that the CA is Σ_2^p-complete. Hence, maybe contrary to expectation, adding a task-assignment problem does not increase the complexity of the coordination problem in an essential way.

3.3 Variant III: Complete Coordination

Obviously, the next step is to extend the notion of being coordinated from having *some* coordinated assignment to having *all* task assignments being coordinated. The associated coordination recognition problem can be stated as follows:

Complete Coordination Recognition (CCR)
INSTANCE: Given a free task instance $(T, \rho, \prec, \mathcal{A}, \boldsymbol{c}(\mathcal{A}), \boldsymbol{c}(T))$.
QUESTION: Is it true that for all assignment sets $\{T_i\}_{i=1}^n$ the resulting fixed task instance $(\{T_i\}_{i=1}^n, \prec)$ is coordinated?

It turns out that CCR is Π_2^p-complete [14].

Again we are interested in checking whether a bounded coordination set exists for this problem. This problem can be seen as guaranteeing that every possible assignment of tasks to agents results in a coordinated fixed task instance by

[6] This can be shown by reduction from PARTITION.

adding a limited number of additional constraints. We now define the most general variant of the coordination problem as

Complete Coordination (CC)
INSTANCE: Given a free task instance $(T, \rho, \prec, \mathcal{A}, \boldsymbol{c}(\mathcal{A}), \boldsymbol{c}(T))$ and a K.
QUESTION: Is it true that for all assignment sets $\{T_i\}_{i=1}^n$ there exists a coordination set $\Delta \subseteq \bigcup_{i=1}^n (T_i \times T_i)$ with $|\Delta| \leq K$ such that the resulting fixed task instance $(\{T_i\}_{i=1}^n, \prec \cup \Delta)$ is coordinated?

By guessing an assignment and using a Σ_2^p-oracle for the resulting PC problem, we can verify a counter-example in polynomial time. Hardness for this class can be proven by reducing a quantified version of PWFP to it, which is Π_3^p-complete. It turns out that the CC-problem is Π_3^p-complete.

Table 3. Complexity of three variants of the coordination problem

	Coordination Recognition	Coordination
Pure Coordination	coNP -complete	Σ_2^p-complete
Coordinated Assignment	Σ_2^p-complete	Σ_2^p-complete
Complete Coordination	Π_2^p-complete	Π_3^p-complete

To sum up, in Table 3, the complexity results of the discussed three variants of the coordination problem are given. It is clear that the general problems are intractable. Therefore, we have to rely on approximation algorithms for finding practical solutions to these problems. We refer the reader to [14] for some practical applications using such approximation algorithms for coordination.

4 Concluding Remarks

We discussed a general framework capturing the basic aspects of task-based planning and coordination for non-cooperative agents. One of the advantages of this framework is that it allows us to study several factors, such as task-assignment procedures and capabilities of agents that might affect the complexity of the plan-coordination problem. The general plan-coordination problems turned out to be intractable. Studying the change in complexity when bounding the number of tasks per agent or the number of agents, we showed that these subclasses are much easier to solve, especially when the number of agents is kept constant.

These results are not only of theoretical interest, but also have some practical implications. First, because we assumed self-interested non-cooperative agents, the proposed solution to the plan-coordination problem allows the agents to plan independently. This enables the (re)use of single-agent planners in a multi-agent planning setting. After the addition of the constraints, the moderately-coupled planning problem has been reduced to a loosely coupled one. It turns out that in this way we can solve the multi-agent planning problem more efficiently by decomposing it into smaller subproblems that can be solved independently.

Secondly, these results show that we can only hope for approximation algorithms to solve these problems. In fact, in [14], such approximations have successfully been applied to solve multi-modal logistic planning problems.

Finally, we note that one of the shortcomings of the current framework is that it lacks the notion of time. Currently, we are extending our framework to represent time intervals and time constraints on tasks to apply our plan-coordination methods on. This will enable us to generalize similar decoupling methods, like the temporal decoupling method [17], and to use them as part of the coordination of temporal planners.

References

1. Dias, M.B., Zlot, R.M., Kalra, N., Stentz, A.: Market-based multirobot coordination: A survey and analysis. Technical Report CMU-RI-TR-05-13, Robotics Institute, Carnegie Mellon University (2005).
2. Kalra, N., Stentz, A., Ferguson, D.: Hoplites: A market framework for complex tight coordination in multi-agent teams. Technical Report CMU-RI-TR-04-41, Robotics Institute, Carnegie Mellon University (2004).
3. Decker, K.S., Lesser, V.R.: Designing a family of coordination algorithms. In: Proc. of DAI. (1994) 65–84.
4. Durfee, E.H., Lesser, V.R.: Partial global planning: A coordination framework for distributed hypothesis formation. IEEE Transactions on Systems, Man, and Cybernetics **21**(5) (1991) 1167–1183.
5. Ephrati, E., Rosenschein, J.S.: Multi-agent planning as the process of merging distributed sub-plans. In: Proc. of DAI. (1993) 115–129.
6. Cox, J.S., Durfee, E.H.: Discovering and exploiting synergy between hierarchical planning agents. In: Proc. of AAMAS. (2003) 281–288.
7. Foulser, D.E., Li, M., Yang, Q.: Theory and algorithms for plan merging. Artificial Intelligence Journal **57**(2-3) (1992) 143–182.
8. von Martial, F.: Coordinating Plans of Autonomous Agents. Springer (1992).
9. Moses, Y., Tennenholtz, M.: Artificial social systems. Computers and AI **14**(6) (1995) 533–562.
10. Shoham, Y., Tennenholtz, M.: On social laws for artificial agent societies: Off-line design. Artificial Intelligence **73**(1–2) (1995) 231–252.
11. Jennings, N.R.: Commitments and conventions: The foundation of coordination in multi-agent systems. The Knowledge Engineering Review **8**(3) (1993) 223–250.
12. Zlot, R.M., Stentz, A.: Market-based multirobot coordination for complex tasks. International Journal of Robotics Research, Special Issue on the 4th International Conference on Field and Service Robotics **25**(1) (2006) 73–101.
13. Zlot, R.M., Stentz, A.: Market-based multirobot coordination using task abstraction. In: Proc. of FSR. (2003).
14. Buzing, P., ter Mors, A.W., Valk, J.M., Witteveen, C.: Coordinating self-interested planning agents. Autonomous Agents and Multi-Agent Systems **12**(2) (2006).
15. Valk, J.M.: Coordination among Autonomous Planners. PhD thesis, Delft University of Technology (2005).
16. Steenhuisen, J.R., Witteveen, C.: Complexity studies in coordinating non-cooperative planning agents. Technical report, Delft University of Technology (2006) Forthcoming.
17. Hunsberger, L.: Group Decision Making and Temporal Reasoning. PhD thesis, Harvard University (2002).

A Model Driven Approach to Agent-Based Service-Oriented Architectures

Ingo Zinnikus[1], Gorka Benguria[2], Brian Elvesæter[3], Klaus Fischer[1], and Julien Vayssière[4]

[1] DFKI GmbH, Stuhlsatzenhausweg 3 (Bau 43), D-66123 Saarbruecken, Germany.
[2] European Software Institute (ESI) - Corporacion Tecnologica Tecnalia - Parque Tecnológico de Zamudio, # 204 E-48170 Zamudio Bizkaia - Spain.
[3] SINTEF ICT, P.O. Box 124 Blindern, N-0314 Oslo, Norway
[4] SAP Research - Level 12 - 133 Mary Street - Brisbane QLD 4000 - Australia
Ingo.Zinnikus@dfki.de, Gorka.Benguria@esi.es, Brian.Elvesater@sintef.no, Klaus.Fischer@dfki.de, Julien.Vayssiere@sap.com

Abstract. Business process management has been identified as an interesting application area for agent technologies. Current developments in Web technologies support the execution of business processes in a networked environment. In this context, the flexible composition and usage of services in a service-oriented environment is a key feature. Additionally, the model-driven architecture (MDA) idea of transforming models on different abstraction levels, from highly abstract design-oriented views to an executable program, is a current trend in business process modeling. BDI agents provide a framework for both aspects by employing a planning from second principles approach, which uses a predefined library of plans and instantiates and adapts these plans. From this perspective, plans are design-time models for agent task execution and for Web Service composition. This paper presents a Rapid Prototyping framework for SOAs built around a Model-Driven Development methodology which we use for transforming high-level specifications of an SOA into executable artefacts, both for Web Services (WSDL files) and for BDI agents. The framework was designed to handle a mix of new and existing services and provides facilities for simulating, logging, analysing and debugging. Our framework was validated on a real industrial electronic procurement scenario in the furniture manufacturing industry. Once input from business experts had been collected, creating the high-level PIM4SOA (Platform Independent Model for SOA) model, deriving the Web service description and incorporating existing Web services took less than a day for a person already familiar with the techniques and tools involved. We show that rapid prototyping of SOAs is possible without sacrificing the alignment of the prototype with high-level architectural constraints.

1 Introduction

Service-oriented architectures (SOAs) have the potential to increase significantly the interoperability of information and communications technology (ICT) applications. However, business applications ask for planned and customizable services, which basically

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 110–122, 2006.

comes down to the requirement for a methodology to do service composition in a flexible and efficient manner. The flexible combination and usage of services in such a service-oriented environment is a key feature that we believe can be best supported by agent technologies. Regarding service composition for an individual agent, we have two extreme options. On the one hand, we can try to adopt a general purpose planner and try to map service descriptions to individual actions which are then composed by the planner. In a naive approach, this will most likely result in a linear sequence of actions, i.e. service calls, that make up the newly composed service. It is quite unlikely that a general purpose planner will come-up with more complex structures. Business Process Modellers are often reluctant when confronted with the idea of choosing services or even processes at run-time. Control of the actual services invoked and the concrete processes performed is preferred over the possibility of an unperceived program decision. On the other hand, we can of course program the composition of services directly in some object-oriented programming language. In this case we are not restricted regarding the structures which we want to use. However, almost every change to a system which adopts such an approach is likely to end-up in painstaking programming sessions.

Composition languages such as e.g. BPEL4WS [1] address some of the problems arising from this approach, but still have limitations regarding flexibility and adaptability (especially during run-time). BPEL4WS does not *prevent* ways for choosing services at run-time, but a service endpoint discovered at run-time must adhere to e.g. a declared partner link and port type. Fault handlers allow the compensation of failures, but the concrete action to be performed in case of a failure has to be modeled in advance.

A planning from second principles approach which uses a predefined library of plans and instantiates and adapts these plans for the task at hand, when the system is at work, seems to bring together the best of two extreme approaches just described. The plan library can be maintained and incrementally updated and the agent will automatically take advantage of the knowledge which the plan library provides. How complex the plan structures in the plan library can be depends on the language which we use for specifying these plans.

SOAs as an architectural style for distributed systems have steadily been gaining momentum over the last few years and are now considered as mainstream in enterprise computing. Compared to earlier middleware products, SOAs put a stronger emphasis on loose coupling between the participating entities in a distributed system. The four fundamental tenets of Service Orientation [2] capture the essence of SOAs: explicit boundaries, autonomy of services, declarative interfaces, data formats and policy-based service description.

Web Services are the technology that is most often used for implementing SOAs. Web Services are a standards-based stack of specifications that enable interoperable interactions between applications that use the Web as a technical foundation [3]. The emphasis on loose coupling also means that the same degree of independence can be found between the organisations that build the different parts of an SOA. The teams involved only have to agree on service descriptions and policies at the level of abstraction prescribed by the different Web Service standards.

Our approach relies on a model-based approach to SOA prototyping that allows us to take existing services into account at a fairly high level of abstraction while keeping the

development of new components aligned with existing ones at each step of the process, from early modelling all the way down to execution and monitoring.

Section 2 introduces the framework for rapid prototyping for SOA. Section 3 details the model-driven development framework. Section 4 details the service enactment and monitoring platform. Section 5 presents how an autonomous agents framework can be used for performing the tasks of composition, mediation and brokering between Web Services. Section 6 introduces a detailed example based on a real industry scenario. Section 7 concludes and proposes avenues for future work.

2 A Framework for Rapid Prototyping of Service-Oriented Architectures

The framework for Rapid Prototyping of SOAs presented here is composed of three parts: a modelling part, a service part and an autonomous agent part. The modelling part is concerned with applying Model-Driven Development (MDD) techniques and tools to the design of SOAs. It defines models and transformations that are specific to the concepts used for SOAs, such as Web Service descriptions and plans for autonomous agents. The service part provides a highly flexible communication platform for Web services. The autonomous agent part deals both with designing and enacting service compositions as well as performing mediation, negotiation and brokering in SOAs.

Each of these three parts leverages the others in various ways. For example, the service part invokes the autonomous agents framework for starting the execution of a service composition described by a plan. The reverse also applies: autonomous agents may invoke Web Services through the tools from the services part. In turn, a description of a service composition at a platform-independent level can be transformed into a plan for autonomous agents, especially BDI agents (see Section 5). High-level service models can also be transformed into standards-based documents such as WSDL files.

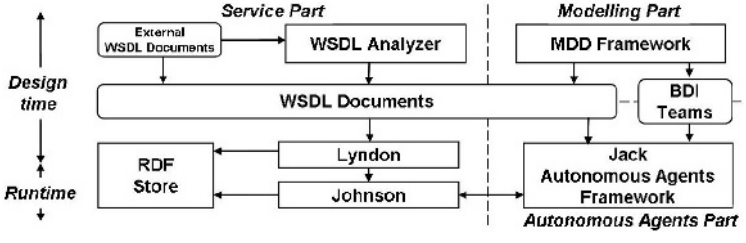


Fig. 1. High-level view of the SOA framework

Figure 1 gives a high-level overview of our framework, illustrating the main components as well as the flow of existing and generated artefacts such as WSDL files and BDI agent plans. The components of the framework are:

- The ***MDD framework*** defines the metamodels used to specify SOAs. It also provides modelling guidelines, model transformation and generation support for execution artefacts such as WSDL files and BDI plans. Importantly, it also supports importing existing WSDL files into the SOA models.
- The ***WSDL Analyzer*** is a tool for detecting similarities at a structural level between WSDL descriptions of Web services and generating the corresponding mappings. It supports flexible service invocation in a dynamic environment and the integration of external services.
- The ***Johnson*** tool is responsible for invoking Web services and receiving calls issued by Web service clients. The Lyndon tool takes WSDL files as input and configures Johnson for playing either the role of service provider, service consumer or service proxy for the service described by the WSDL file analyzed.
- The ***Jack*** [4] tool is used for specifying plans for autonomous agents which form teams that can invoke or receive calls from Web services. The plans used may be created as the result of an MDD process.
- The ***RDF store*** stores as RDF files both design-time information (WSDL files) and runtime information (SOAP messages) for the purpose of monitoring.

3 Model-Driven Development Framework for SOA

Designing SOAs at the enterprise level involves several different stakeholders within the enterprise. In order to support the various views pertinent to these stakeholders, we have defined an MDD framework. The MDD framework partitions the architecture of a system into several visual models at different abstraction levels subject to the concerns of the stakeholders. This allows important decisions regarding integration and interoperability to be made at the most appropriate level and by the best suited and knowledgeable people. The models are also subject for semi-automatic model transformations and code generation to alleviate the software development and integration processes.

Figure 2 details the model-driven development framework. It follows the OMG Model Driven Architecture (MDA) [5] approach and defines a Platform Independent Model (PIM) for SOA (PIM4SOA) and Platform Specific Models (PSMs) for describing Web services (XSD and WSDL), Jack BDI agents and BPEL [1] processes. The PIM4SOA is a visual PIM which specifies services in a technology independent manner. It represents an integrated view of the SOA in which different components can be deployed on different execution platforms. The PIM4SOA model helps us to align relevant aspects of enterprise and technical IT models, such as process, organisation and products models. This model allows us to raise the abstraction level at which we can talk about and reason on the architecture we design. The PIM4SOA metamodel defines modelling concepts that can be used to model four different aspects or views of a SOA:

1. **Service:** Services are an abstraction and an encapsulation of the functionality provided by an autonomous entity.
2. **Information:** Information is related to the messages or structures exchanged, processed and stored by software systems and components.
3. **Process:** Processes describe sequencing of work in terms of actions, control flows, information flows, interactions, protocols, etc.

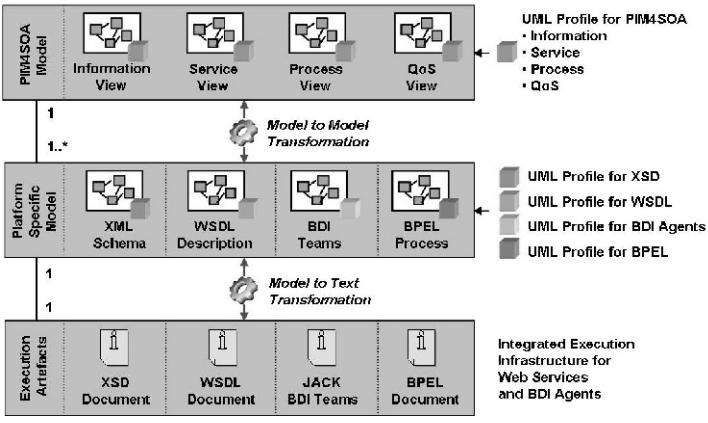


Fig. 2. Model-driven development framework

4. **Quality of service (QoS):** Extra-functional qualities that can be applied to services, information and processes.

The MDD framework provides model-to-model transformation services which allow us to transform PIM4SOA models into underlying PSMs such as XSD, WSDL, JACK BDI agents or BPEL. The PSMs depicted in Figure 2 are also visual models which IT developers can further refine by adding platform-specific modelling constructs such as deployment properties. PSMs typically represent a one-to-one mapping to an execution artefact. Dependencies between the various components are modelled at the PIM-level and two-way model transformations help us to ensure interoperability at the technical level and consistency at the PIM-level. Tool support for the MDD framework has been developed as a set of plugins for Rational Software Modeller (RSM) (IBM Rational Software). RSM is a UML 2.0 compliant modelling tool from IBM based on the Eclipse modelling environment. All models and metamodels were implemented using the EMF Core (Ecore) metamodel. Model transformations have been implemented using the model transformation capabilities of the RSM/Eclipse platform.

4 A Lightweight Web Services Enactment Framework

The part of our SOA Rapid Prototyping framework that deals with the enactment of Web services is composed of three tools which are arranged along a value chain: the WSDL Analyzer, the Lyndon tool and the Johnson tool.

4.1 The WSDL Analyzer

The WSDL Analyzer is a tool for detecting similarities between Web service descriptions. The tool can be used to find a list of similar services r to support the integration

of external services by producing a mapping between messages, thereby enabling brokering and mediation of services. The algorithm of the WSDL Analyzer improves over an algorithm for finding structural similarities proposed by Wang and Stroulia ([6], [7]) by taking into account additional features of the WSDL structure. More specifically, we make use of the tree-edit distance measure [8] and the concept of a weak subsumption relation introduced by Nagano et al. [9]. The idea of the tree-edit distance is that a similarity between two XML structures can be measured by stepwise transforming a tree representation of the first structure into the other. The steps necessary for that transformation provide the measure for their similarity, and, at the same time, induce the mapping between the schemas. Possible steps are basic edit operations such as node inserts, deletes and relabels. The algorithm of Wang and Stroulia considers only node matching without editing, or simple renaming operations such as changing a data type from string to int. Nagano et al. give three different types of weak subsumption: replacing labels, pruning edges and removing intermediate nodes. These operations can be correlated to specific tree-edit operations, namely relabeling and deleting nodes. A possible scenario for using the WSDL Analyzer is that the user already knows a service which provides the correct format. The WSDL of this service can be used as requirement for a similarity search. The WSDL Analyzer allows browsing the original WSDL and the candidate files. The algorithm detects common structures in port types, operations, messages and data type definitions. WordNet is integrated to improve the matching result. Mappings are assessed with a score which is used to establish a ranking between candidate service descriptions. The result is a ranking of the candidates according to their matching score. Based on the similarities, a mapping is generated between two WSDL descriptions which can be used to transform SOAP messages exchanged between similar services at runtime. The translation can be done automatically, if there is a one-to-one correspondence between elements. However, if there exist several possible corresponding elements, translation requires intervention from a user in order to unambiguously transform parameters. The latter case shows the limitation of the structural approach. There are possible mismatches which can be detected with the help of the WSDL Analyzer, but not automatically corrected.

4.2 The Johnson and Lyndon Tools

Johnson is a runtime tool that enables users to enact most of the roles typically found in an SOA, thereby enacting complex SOA scenarios by sending real SOAP messages between Web services without having to write a single line of code. Johnson features a Web-based user interface designed to closely resemble Web-based email applications, with the only difference that SOAP messages and Web Services endpoints are used in place of email messages and email addresses. The user can see incoming SOAP messages in the Inbox and create outgoing SOAP messages in the Outbox that will be sent to external Web services. A powerful user-interface generator relieves the user from having to deal with XML documents by generating forms for displaying and editing any XML-based data type. When playing the role of a Web service consumer, for example, a user would create a message in the Outbox, send that message to a remote Web service, and later see the response message appear in the Inbox. On the reverse, a user enacting a Web service provider would read incoming requests in the Inbox and reply to them

by creating response messages in the Outbox. Central to the architecture of Johnson are the concepts of endpoint, processing modules and processing chains. An endpoint is an abstraction for the address of a service. To each endpoint is attached a processing chain, which implements the processing of messages for that endpoint. Each processing chain is composed of a number of processing modules which are called in sequence. Each processing module is responsible for implementing one aspect of the processing of the message. A processing module may be responsible, for example, for adding routing information to the message, or dealing with reliability of the message exchange, or performing a transformation of the message content, or appending the message to an audit trail of messages sent. Assembling existing processing modules into new processing chains can be done through the user interface. Creating new processing modules, though, requires writing code.

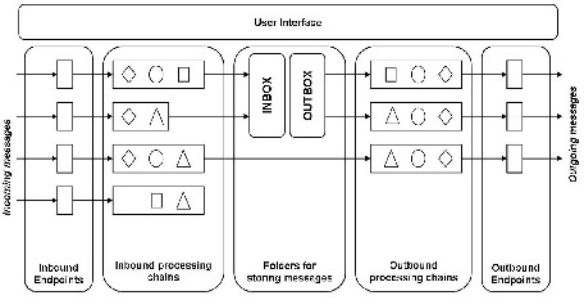


Fig. 3. Architecture of the Johnson tool

Figure 3 shows a sample instance of Johnson where all the aforementioned concepts are illustrated. The different processing modules are represented as different shapes such as circles, lozenges, squares and triangles. Messages received on the third inbound endpoint from the top do not end up in the inbox but are directly sent out using one of the outbound endpoints. This is possible because this logic is coded in the last processing module of the inbound processing chain. Being able to specialise message processing for each endpoint basis allows us to play the role of Web services that would implement different subsets of the Web Services stack of specifications, which proved very useful for studying the possible interoperability issues raised by the use of unrelated specifications together.

A processing module was also developed for keeping an audit trail of messages, which forms the basis for troubleshooting and performance measurement. The headers of SOAP messages are turned into RDF and stored in an RDF store. The Lyndon tool can be seen as the design-time counterpart of the Johnson tool. It analyses WSDL files and automatically configures Johnson for playing either the role of consumer or provider of the service described. Lyndon parses a WSDL file and determines which endpoints need to be created, and which processing chains need to be assigned to them. Determining which processing modules to include in the processing chain takes into account information extracted from the WSDL file as well as options set by the user.

The user may, for example, specify whether Johnson should be configured as a service consumer or a service provider, or whether messages sent to or from the service should be logged. Some configuration information can be extracted from the WSDL file, such as the need for implementing the WS-Addressing specification, which is specified as part of the description of the bindings of a Web service. Lyndon also generates an RDF representation of WSDL files and stores it into the same RDF store used for logging SOAP messages. Having both design-time and runtime artifacts in the same store is critical to monitoring the SOA and detecting services that do not behave accordingly to the service description they published.

5 An Agents-Based Execution Platform

The aim of the extended JACK agent framework for Web Services is to provide a goal-oriented service composition and execution module within an SOA. Following the Belief Desire Intention (BDI) model [10], agents are autonomous software components that have explicit goals to achieve or events to handle (desires). Agents are programmed with a set of plans to describe how to go about achieving desires. Each plan describes how to achieve a goal under varying circumstances. Set to work, the agent pursues its given goals (desires), adopting the appropriate plans (intentions) according to its current set of data (beliefs) about the state of the world. The combination of desires and beliefs initiating context-sensitive intended behaviour is part of what characterises a BDI agent. BDI agents exhibit reasoning behaviour under both pro-active (goal directed) and reactive (event driven) stimuli. Adaptive execution is introduced by flexible plan choice, in which the current situation in the environment is taken into account.

A BDI agent has the ability to react flexibly to failure in plan execution. Agents cooperate by forming teams in order to achieve a common goal. The JACK agent platform is not inherently ready for interaction within a Web service environment. Additional steps are necessary for enabling interactions between the agent platform and Web services, especially when the agents themselves offer services. In this case, some tools are needed for generating the server and client-side code for using JACK inside a Web server. Figure 4 is an overview of the extended JACK architecture for Web service composition and plan execution, with at its core the JACK agent framework with plan library and knowledge base. Following the MDA approach, a modeller specifies at design time a set of plans (PSM level) that constitute the workflow library of the agents. Web service calls are integrated as steps into plans. Workflows are modelled graphically and most of the common workflow patterns are supported.

In order to prepare the ground for a transformation from a PIM4SOA model to the JACK PSM, following the MDA approach, the service providers are mapped to Jack agents/teams. The parts of the PIM which define the processes involved are mapped to agent/team plans and correlated events, whereas the parts which define the interfaces are mapped to the modules which provide the client- and server-side code for the JACK agent platform. Hence, the idea is to apply a methodology which separates the Web service adapter from the core plans.

Accordingly, the steps of the methodology consist in

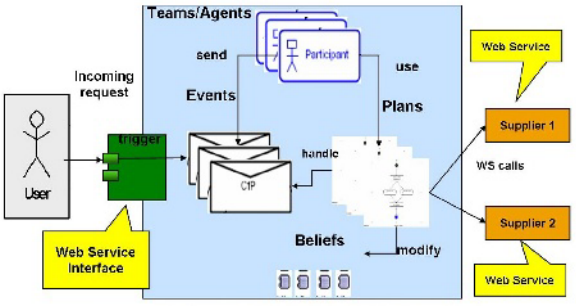


Fig. 4. Extended JACK framework for service composition and execution

- Constructing the agent/team plans for coordinating the internal activities and composing Web service calls based on the generated PSM models.
- Generate adapter code (client and server-side) with the help of a suitable toolkit (e.g. Apaches Axis framework [11]), based on a WSDL description of the service
- Adjust an agent's belief set so that it complies with the data structure provided by the Web services used and offered
- Specify events which correlate to Web service calls (in case the agents offer a Web service)
- Adjust the plans which handle these events
- Insert plan steps which trigger Web service calls (in case the agent calls a Web service) and call the adapter code from the plans directly

In order to compare an agent-based approach with other standards for Web service composition, the following distinction is useful [12]:

- Fixed composition: a fixed composite service requires the component services to be integrated in a fixed way. The composition structure and the component services are statically bound.
- Semi-fixed composition: in this situation, a composition definition which indicates the structure of the composition is generated. However, the actual service binding needs to be decided at run time.
- Explorative composition: this type of service composition is specified on the fly and requires dynamically structuring the composition service and binding component services.

Fixed composition can be done with e.g. BPEL4WS, but also by applying BDI agents. Semi-fixed composition might also be specified with BPEL4WS: partner links are defined at design time, but the actual service endpoint for a partner might be fixed at run-time, as long as the service complies with the structure defined at design time. Late binding can also be done with the JACK4WS framework. The service endpoint needs to be set (at the latest) when the actual call to the service is done. Explorative composition is beyond of what BPEL4WS and a BDI agent approach offer (at least if they are used in a 'normal' way). To enable explorative composition, a general purpose

planner might be applied which generates, based on the service descriptions stored in a registry, a plan which tries to achieve the objective specified by the consumer [13].

It might seem as if BPEL4WS and JACK4WS offer the same features. However, there are several advantages of a BDI agent approach which become evident by looking more closely at the definition of a semi-fixed composition. An important question is how the availability of a service is detected. This might be checked only by actually calling the service. If the service is not available or does not return the expected output, an exception will be raised. BPEL4WS provides a fault handler which allows specifying what to do in case of an exception. Similarly, a JACK agent plan will fail if a WS call raises an exception, and execute some activities specified for the failure case. However, the difference is that a plan is executed in a context which specifies conditions for plan instances and also other applicable plans. The context is implicitly given by the beliefs of an agent and can be made explicit. This means that in a given context, several plan instances might be executed, e.g. for all known services of a specific type, the services are called (one after another), until one of the services provides the desired result. An exception in one plan instance then leads to the execution of another plan instance for the next known service. Additionally, JACK provides the possibility of 'meta-level reasoning' which allows choosing the most feasible plan according to specified criteria. Similarly, if for a specific goal several plan types are feasible, the JACK execution engine executes one of these plans and, in case of a failure, immediately executes the next feasible plan to achieve the desired goal. The BDI agent approach supports this adaptive behaviour in a natural way, whereas a BPEL4WS process specification which attempts to provide the same behaviour would require awkward coding such as nested fault handlers etc. Another advantage is that *extending* the behaviour by adding a new, alternative plan for a specific task is straightforward. The new plan is simply added to the plan types and will be executed at the next opportunity. Similarly, *customizing* the composition is facilitated since the different plans clearly structure the alternatives of possible actions. Since the control structure is implicit, changes in a plan do not have impact on the control structure, reducing the danger of errors in the code.

6 An SOA Rapid Prototyping Case Study from the Furniture Industry

We have tested our approach against a real industry scenario, namely an electronic procurement process that spans the furniture manufacturing industry and the interior decoration retailers. The procurement process, traditionally, covers the activities that one organization performs to derive the goods to be purchased from the providers to build the products requested by the customer. We started by creating a PIM4SOA model based on the input we gathered from business experts at AIDIMA, a Spanish technology advisory body for the furniture industry. This PIM4SOA model details the interactions between the different roles involved in the e-procurement end-to-end process, from the initial customer order to the final acknowledgement of the delivery of the goods. To describe these interactions the PIM4SOA model identifies the different roles, the collaborations between those roles, the information exchanged, the internal behaviour of those collaborations and the expected quality that should be provided by the roles.

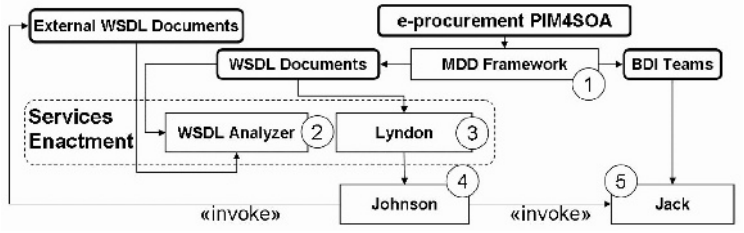


Fig. 5. Approach for the validation of the Rapid Prototyping framework

The following approach was followed for the validation of the Rapid Prototyping framework (see Figure 5). First we used the MDD framework (1) to derive the WSDL files and BDI models from the e-procurement PIM4SOA model. The next objective was to enact the services identified for the e-procurement scenario using the WSDL Analyser (2) and the Johnson and Lyndon (3) tool. Because some of the pieces of the SOA already existed, we used the WSDL Analyser to locate services similar to those required in the e-procurement scenario. For those that do not exist we have used the Lyndon tool to configure the Johnson platform to simulate them. The next step was to configure Johnson (4) to act as a service proxy; this allowed us to change the final service endpoints without affecting the process execution. Finally the PSM model for Jack (5) was implemented and tested with the enacted services.

- The ***MDD framework*** uses model-to-model transformations to derive the platform specific models for XML schemas, WSDL descriptions, and JACK Model from the PIM4SOA model as stated in the Section 3. These models are then completed by the platform experts to make them ready for the generation of the execution artefacts through the use of model-to-text transformations.
- The ***WSDL Analyzer*** compares the types of the parameters of the services required with the available services and returns a ranked set of candidate service. The technical experts then select the services that will be used and the tool provides the appropriate mappings to transform the messages at runtime.
- The ***Lyndon tool*** configures the Johnson tool for enacting some of the services and for logging all appropriate information in the RDF store for later analysing and debugging of the SOA.
- The ***Johnson tool*** is also configured to incorporate the endpoints of the mappings services generated by the WSDL Analyzer.
- Finally the ***Jack tool*** is loaded with the PSM-level model (agents/teams, plans, events, beliefs etc) for the e-procurement scenario.

Once we have implemented the prototype we can execute it together with the client to check that it achieves the stated requirements. If we need to analyse the details of the message exchange we can use the Johnson platform for doing so. Besides, the Johnson platform also allows us to simulate other situations in a flexible and agile way. Situations such as service delays, service shutdown or service errors can be simulated, logged and analysed.

7 Conclusion and Future Work

This paper presented a rapid prototyping framework for SOAs built around a Model-Driven Development methodology which is used for transforming high-level specifications of an SOA into executable artefacts, both in the realm of Web Services and in that of BDI agents. The framework can handle a mix of new and existing services and provides facilities for simulating, logging, analysing and debugging.

BDI agents, especially the JACK agents platform, provide the framework for a model-driven transformation of SOA specifications into executable processes and services. JACK agents execute the process descriptions and act as services within the SOA.

The framework was validated on a real industrial electronic procurement scenario from the furniture manufacturing industry. Once input from business expert had been collected, creating the high-level PIM4SOA model, deriving the Web service description and incorporating existing Web services took less than a day for a person already familiar with all the tools involved. After having run a few variants of the SOA, it became clear that the model-based approach we followed delivers significant value in keeping all the pieces of the SOA aligned with high-level business objectives throughout rounds of prototyping.

Acknowledgments

The work published in this paper is partly funded by the European Commission through the ATHENA IP (Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications Integrated Project) (IST-507849). The work does not represent the view of the European Commission or the ATHENA consortium, and the authors are solely responsible for the papers content.

References

1. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1. Technical report (May 2003)
2. Box, D.: A guide to developing and running connected systems with indigo. MSDN Magazine (January 2004)
3. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web Services Architecture. Technical report, W3C Working Group (February 2004)
4. The Agent Oriented Software Group: JACK Development Environment. http://www.agent-software.com (2004)
5. Soley, R.: Model Driven Architecture. OMG (November 2000)
6. Wang, Y., Stroulia, E.: Flexible Interface Matching for Web-Service Discovery. In: 4th Int'l Conf. on Web Information Systems Engineering (WISE 2003). (2003)
7. Wang, Y., Stroulia, E.: Semantic Structure Matching for Assessing Web-Service Similarity. In: 1st Intl Conf. on Service Oriented Computing (ICSOC 2003). Volume 2910 of Lecture Notes in Computer Science, Springer-Verlag (2003) pp. 197 – 207
8. Shasha, D., Zhang, K.: Approximate Tree Pattern Matching. In: Pattern Matching Algorithms. Oxford University Press (1997) 341 –371

9. Nagano, S., Hasegawa, T., Ohsuga, A., Honiden, S.: Dynamic Invocation Model of Web Services Using Subsumption Relations. In: IEEE International Conference on Web Services (ICWS). (2004) 150 – 156
10. Rao, A.S., Georgeff, M.P.: Modeling Rational Agents within a BDI-Architecture. In Allen, J., Fikes, R., Sandewall, E., eds.: Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91), Morgan Kaufmann publishers Inc.: San Mateo, CA, USA (1991) 473–484
11. Axis: (http://ws.apache.org/axis/)
12. Yang, J., Heuvel, W., Papazoglou, M.: Tackling the Challenges of Service Composition in e-Marketplaces. In: Proceedings of the 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems (RIDE-2EC 2002), San Jose, CA, USA. (2002)
13. Sirin, E., Parsia, B., Wu, D., Hendler, J.A., Nau, D.S.: HTN planning for Web Service composition using SHOP2. J. Web Sem. **1** (2004) 377–396

Meta-models, Models, and Model Transformations: Towards Interoperable Agents

Christian Hahn[1], Cristián Madrigal-Mora[1], Klaus Fischer[1], Brian Elvesæter[2], Arne-Jørgen Berre[2], and Ingo Zinnikus[1]

[1] DFKI GmbH, Stuhlsatzenhausweg 3 (Building D 3-2), D-66123 Saarbrücken, Germany
{christian.hahn, cristian.madrigal, klaus.fischer, ingo.zinnikus}@dfki.de
[2] SINTEF ICT, P. O. Box 124 Blindern, N-0314 Oslo, Norway
{brian.elvesater, arne.j.berre}@sintef.no

Abstract. Services provide an universal basis for the integration of applications and processes that are distributed among entities, both within an organization and across organizational borders: This paper presents a model-driven approach to design interoperable agents in service-oriented architectures (SOA). The approach provides a foundation for how to incorporate autonomous agents into a SOA using principles of model-driven development (MDD). It presents a metamodel (AgentMM) for a BDI-agent architecture and relates AgentMM to a platform-independent model for SOAs (PIM4SOA). In this paper we mainly concentrate our discussions on the service and process aspects of SOA and how transformations to agent technology would look like. We argue that this mapping allows the design of generic agent systems in the context of SOAs that are executable in an adaptive and flexible manner.

Keywords: Modeling Agents, Model-Driven Development, Service-Oriented Architectures, Metamodels.

1 Introduction

Model-driven development (MDD) is emerging as the standard practice for developing modern enterprise applications and software systems. MDD frameworks define a model-driven approach to software development in which visual modeling languages are used to integrate the huge diversity of technologies used in the development of software systems. As such the MDD paradigm, i.e., to develop (i) metamodels that describe the concepts and their relationships and (ii) model transformations that map those concepts and relationships from metamodel to metamodel, provides us with a better way of addressing and solving interoperability issues compared to earlier non-modeling approaches [1].

The current state of the art in MDD is much influenced by the ongoing standardization activities around the OMG Model Driven Architecture (MDA) [2]. MDA defines three main abstraction levels to software development: From a top-down perspective, it starts with a computation independent model (CIM), describing the application context and requirements, that is refined to a platform independent model (PIM), which specifies software services and interfaces independent of software technology platforms.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 123–134, 2006.

The PIM is further refined to a set of platform specific models (PSMs) that describes the realization of the software systems with respect to the chosen software technology platforms.

The focus of this paper is on PIM to PSM transformation development, i.e., the basic idea is to define the transformation from a PIM to an agent PSM. For this purpose, a PIM for Service Oriented Architectures (PIM4SOA) [3] and a metamodel for agent technologies (AgentMM) are presented. If PIM4SOA models can actually be transformed into and executed by agent models, agent systems can be built, so that they can really interoperate with competing technologies in SOAs.

The paper is structured as follows: In Section 2, we present the PIM4SOA metamodel, followed by the metamodel for a specific agent architecture (Section 3). In Section 4, we compare the two metamodels and discuss feasible transformations. Section 5 illustrates related work. Finally, Section 6 presents some conclusions.

2 A Metamodel for Service-Oriented Architectures

Services are loosely coupled, dynamically locatable software pieces, which provide a common platform-independent framework that simplifies heterogeneous application integration. An approach based on agent technologies can be an interesting opportunity when executing services, because:

- agents are self-aware and they acquire the awareness of other agents and their attitudes,
- agents are proactive, whereas services are passive until invoked,
- in contrast to services, agents act in an autonomous manner that is required by many Internet applications,
- agents are cooperative and, by forming organizations, they can provide higher-level and more comprehensive services.

Current standards in the domain of services do not provide any of those functionalities [4]. The metamodel for SOAs addresses the conceptual and technological interoperability barrier and aims at defining platform independent modeling language constructs that can be used to design, re-architect and integrate technologies supporting SOA. The introduction of agents will enable us to design SOAs that are more adaptable and flexible, and, thus, better able to cope with changes over time—which is important for supporting interoperability.

In order to support an evolution of the metamodel for SOAs (PIM4SOA) we have defined a small metamodel core and structured it into groups, each focusing on a specific aspect of a SOA. The current version of the PIM4SOA defines modeling concepts that can be used to model four different aspects of SOAs: Services, information, processes and non-functional aspects. In this paper, we mainly concentrate on the service and process aspects, where services are an abstraction and an encapsulation of the functionality provided by an autonomous entity, and processes describe sequencing of work in terms of actions, control flows, information flows, interactions, protocols, etc. In general, SOAs are formed by components provided by a system or a set of systems to achieve a shared goal.

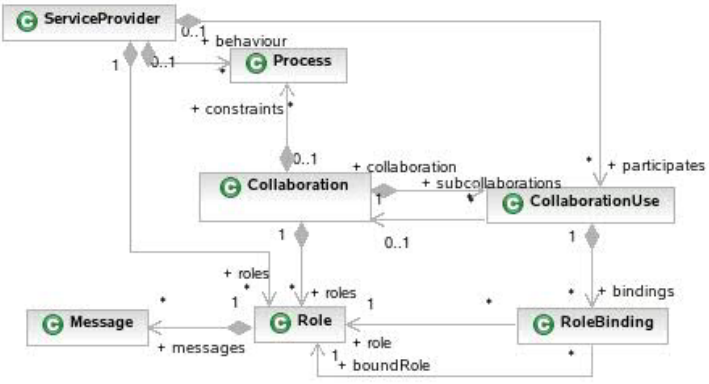


Fig. 1. Service concepts of the PIM4SOA metamodel

The service aspect of the PIM4SOA presents services modeled as collaborations that specify a pattern of interaction between the participating roles. A subset of the metamodel for this aspect is presented in Figure 1. The *Collaboration* specifies the involved roles and their responsibilities. Additionally, a *CollaborationUse* specifies the application of a *Collaboration* in a specific context and includes the *RoleBindings* to entities in that context. Collaborations are composable and the responsibilities of a role in a composite *Collaboration* are defined through *CollaborationUses* by binding roles from the composite to roles of its subcollaborations. The simplest form of *Collaboration* is the binary collaboration, which has no subcollaborations and only two roles; A requester that provides the input, and a provider that produces the output parameters. Therefore, a *Role* represents how a partner participates in the *Collaboration* by providing services and parameters and using services provided by other partners in its service collaboration.

Furthermore, collaborations can have a *Process* that specifies the constraints that define how the involved roles interact. We refer to this as the collaboration protocol. The collaboration protocol is specified from a global view point. Constraints on the expected behavior of a role can be deduced.

The process elements of the PIM4SOA metamodel are shown in Figure 2. This process aspect is closely linked to the service aspect. The primary link is described by the *Process* that belongs to a *ServiceProvider*. The *Process* contains a set of steps (generally tasks), representing the actions to be carried out, and *Interactions/Flows* linking the tasks together. In addition, the *Process* also contains a set of *Flows* between these actions which may indicate the transfer of specific data.

3 A Metamodel for BDI Agents

There are several alternatives when it comes to transforming PIM4SOA models into models that can actually be executed. Agent technologies can provide various

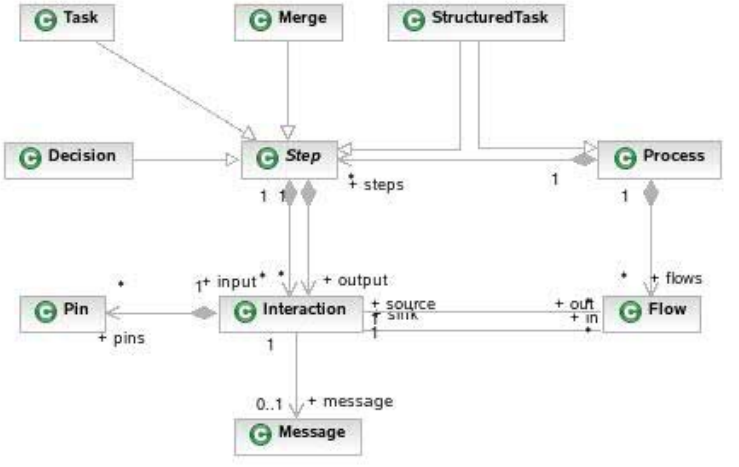


Fig. 2. Process concepts of the PIM4SOA metamodel

contributions to SOAs. For instance, agent technologies allow a flexible and adaptive execution. Also, well-known agent protocols, such as the contract net protocol [5], could be adopted for the service supplier selection, when the number of suppliers is not know at design time. This kind of service selection can be nicely described with agent technologies, while a straightforward model for the PIM4SOA is not available. The integration of agent technologies with PIM4SOAs is therefore a worthwhile enterprise.

For the design of agents with rational and flexible problem solving behavior, the BDI agent architecture has proven to be appropriate during the last decade [6,7]. Three mental attitudes (beliefs, desires, and intentions) allow an agent to act in and to reason about its environment in an effective manner. A vast number of tools and methodologies have been developed to foster the software-based development of BDI agents and multi-agent systems (MAS) [8,9,10,11,12]. Rather than inventing our own agent metamodel, we took a bottom-up approach, by extracting the metamodel (AgentMM) from one of the most sophisticated tools to design BDI agents, namely JACK^{TM} Intelligent Agents[3] [13]. Figure 3 presents the most interesting part of this metamodel.

Table 1 summarizes the most important high-level BDI concepts. From these, the most relevant one in AgentMM is the concept of *Team*, which can be either atomic, in which case we can refer to it simply as an *Agent*, or a set of required roles—subteams—that all together form the *Team*. It is important to note that it is not necessary for all members of the *Team* to be involved in all the tasks it performs. Rather, for each individual task, a subset of the available roles is selected to actually work on it. The tasks a given *Team* is able to work on are defined by the roles that it is able to fulfill.

[3] JACK^{TM} is the trademark of an agent oriented model developed by Agent Oriented Software Group. A free evaluation package for JACK Intelligent Agents is available for download.

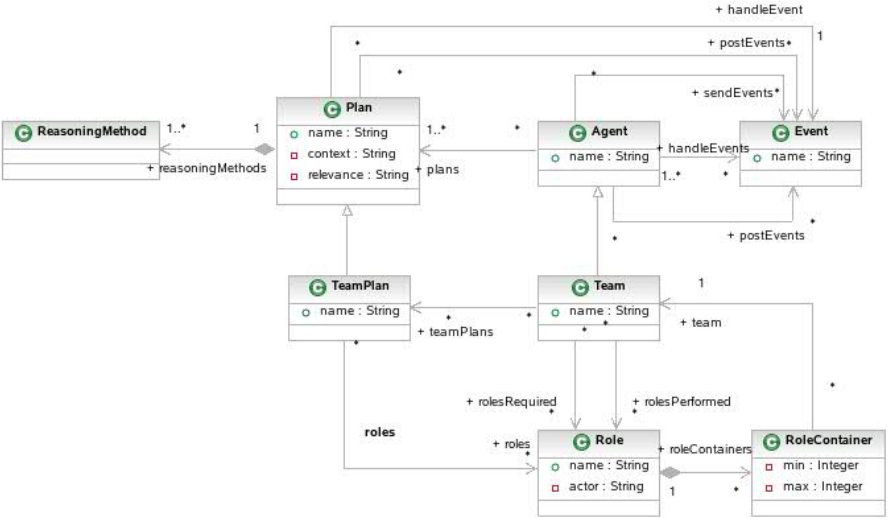


Fig. 3. Partial metamodel for BDI agents

Table 1. High-Level BDI Concepts

High-Level BDI Concepts	
Team	Specifies the structure of one or more entities (Teams/Agents) that is formed to achieve a set of desired objectives
Role	Specifies a role as a type by listing the types of the events the role can deal with
Named Role	An instance of a specific role type. This concepts allows multiple roles with the same type in teams and team plans
Events	The type of stimuli a team, role, or team plan reacts to or posts
Team Plan	Specifies the behavior of a team in reaction to a specific event. In general, a team plan is a set of steps specifying how a particular task is achieved by particular roles
Named Data	Allows the team to store information/beliefs

Table 2. The JACK^{TM} Plan Structure

Structure of Plans	
Triggering Condition	Specification of the event the plan reacts to
Relevance Condition	Additional constraints on the triggering event
Context Condition	Constraints on the state of affairs the plan is designed for to deal with. Usually these constraints access the agents beliefs
Plan Body	Actions that should be taken when the plan becomes active. Can include pure Java code but allows the use of special concepts to drive BDI reasoning

Role definitions are the second most important concept to define teams because a role specifies which messages—in JACK^{TM} those are rather events—the role fillers are able to react to and which messages they are likely to send.

How a team actually reacts to an incoming request is specified by a set of team plans. The structure of a plan can be seen in Table 2. Due to space restrictions, we concentrate in this paper on the concepts of the *Plan Body*—the core part of a plan. Each team plan has an explicitly defined objective (incoming message or internal event) for which this team plan is responsible. When the so-called triggering event is raised and all additional criteria are valid (i.e. *Relevance* and *Context Condition*), a specific team plan is executed by creating an instance of this team plan. As a consequence, a concrete team (already known or newly established) to actually execute the team plan is established.

The process aspect of AgentMM is presented in Figure 4 and it provides the constructs to model the body of the Plans and other, so called, reasoning methods (see Figure 3). The whole package is based on two abstract classes—the *ProcessBase* and *NodeBase*. *NodeBase* represents the basic node in the plan body graph, it provides a reference to its default flow node, the next node in the execution order, and references to the incoming flows, the previous nodes in the execution. *ProcessBase* is the graph container that includes a list of all the nodes and a reference to the starting node. The *Process* class extends *ProcessBase* and represents the main component of the reasoning method body. It is also a process node in itself, which permits modeling nested processes.

Further specialization of the nodes is performed through the *ForkNode* abstract class, which adds an alternative output flow to the *NodeBase*. As shown in Figure 4, the *Node* classes are separated in two groups: The ones that inherit directly from *NodeBase*—only one output flow, and the ones that inherit from *ForkNode*—with the alternative output flow. The semantics of the alternative flow varies depending of the particular node being modeled, for example, in a *DecisionNode* the alternative flow represents the *else* path of the decision, while in a *SubgraphNode* it represents the *fail* path. For detailed semantics of the plan constructs in JACK^{TM} please refer to the JACK^{TM} Documentation [13].

4 Comparison of the Metamodels for PIM4SOA and BDI Agents

In this section, we bring together the metamodel concepts from the two previous sections and relate them to one another in a mapping and a transformation derived from it. Although the concepts from PIM4SOA and AgentMM differ quite significantly the mapping of PIM4SOA models to AgentMM models seems to be feasible as the AgentMM is more expressive.

The first element we inspect is the *ServiceProvider* (presented in Figure 1). At first glance an agent seems to be the best match, but since a *ServiceProvider* references roles, it is recommended to assign it to a team. The name of the *ServiceProvider* coincides with the name of the team and its roles are the roles the team performs.

While a *ServiceProvider* is supposed to represent an atomic team, a *Collaboration* is mapped onto a team that may consist of any number of agents. However, since collaborations do not specify any cardinalities for roles, we can assume that a collaboration asks for exactly one filler for each of the required roles. Correspondingly, we suggest

Table 3. Partial transformations between PIM4SOA and AgentMM concepts

PIM4SOA		**AgentMM**		**Notes**
Concepts	Attributes	Concepts	Attributes	
ServiceProvider	→	***Team***		
	name		name	*name of the service provider is used as team identifier*
	roles		rolesPerformed	*each role is mapped to a role performed by the team*
	behaviour		usesPlans	*for each behavior/process a new teamplan is instantiated*
	participates bindings boundRole		rolesRequired	*roles a team makes use of are specified in the role bindings the service provider participates*
Message	→	***Event***		
	name		name	*name of message is used as event identifier*
Role	→	***Role***		
	name		name	*name of the role (PIM4SOA) is used as role (AgentMM) identifier*
	messages		handleEvents roleContainers min = 1 max = 1	*required roles have a min/max requirement of one filler*
		Team		
	name		name + 'Team'	*name of the role plus the extension "Team" is used as team identifier*
	self		rolesPerformed	*role itself is mapped to the team's performed role*
	→	***TeamPlan***		*TeamPlan responsible for handling service requests*
	name		'Receive' + *name* reasoningMethods body establish pass fail handleEvent postEvents	*bases on Role (PIM4S0A)* *name of role acts as identifier* *body contains send method* *these reasoning methods are automatically generated* *handles service requests* *posts corresponding asynchronous message*
	→	***TeamPlan***		*TeamPlan responsible for invoking services*
	name		'Invoke' + *name* reasoningMethods body establish pass fail handleEvent	*bases on Role (PIM4S0A)* *name of role acts as identifier* *body contains service call* *these reasoning methods are automatically generated* *handles service requests*

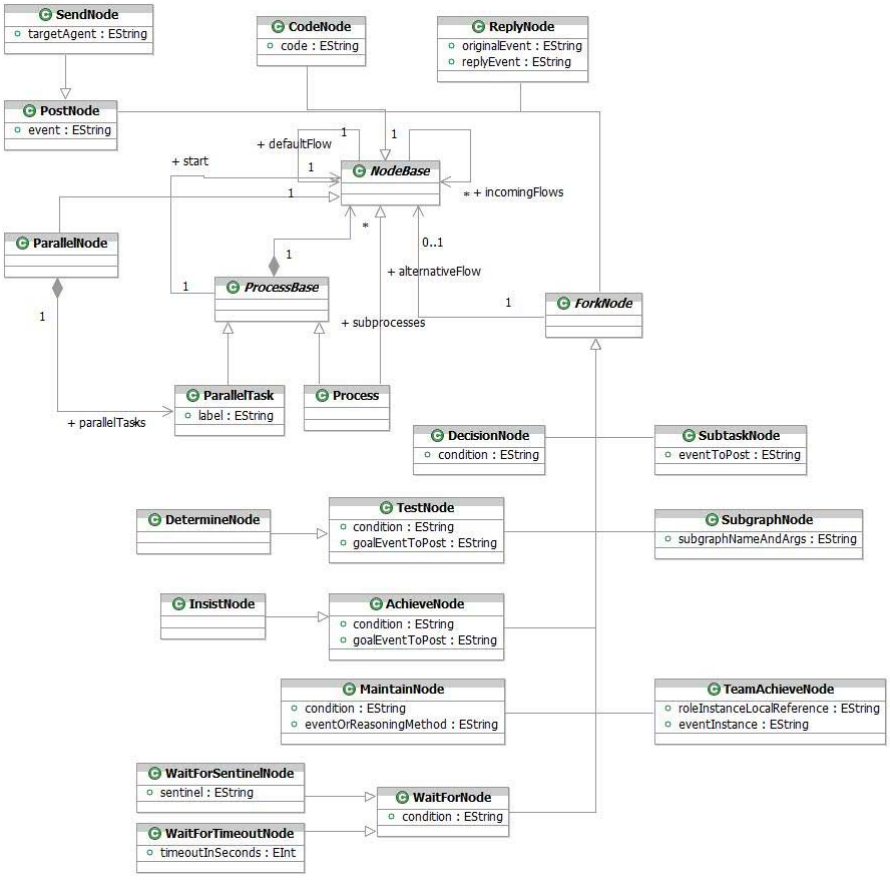


Fig. 4. Process Metamodel of AgentMM (Plan Body)

to map collaborations to team structures where the required roles have a cardinality requirement of exactly one role filler. Although the metamodel for PIM4SOA allows to specify constraints on the behavior of the participating collaborations and their roles, up to now it is unclear how these constraints might look like.

In practical cases, it turns out that a *Process* is provided for *ServiceProviders* and requester roles only. This allows to use the PIM4SOA model in a manner where the information on *Collaborations* and *CollaborationUses* is mapped to a set of teams, where each of the teams consists of a service requester—represented by the team itself—and a set of roles this specific service requester makes use of. Due to the fact that non-composed collaborations in the PIM4SOA are binary, it is always clear who requests and who provides the corresponding service. Interactions other than pure service requests-provisions do not exist. This matches nicely with the fact that, in JACK$^{\text{TM}}$, the interaction of a team as a whole with its attached roles is easy to describe. The behavior of the service requester is therefore mapped to a team plan where each request of a service from a *ServiceProvider* is represented by the construct *team achieve* (see

Table 4. Partial transformations between the process metamodels

PIM4SOA		AgentMM		Notes
Concepts	Attributes	Concepts	Attributes	
Process	→	***TeamPlan***		*Filter: corresponding Team bases on Service Provider*
			name + 'TeamPlan'	*name := name of service provider*
			reasoningMethods	
	self		body	*the process itself is mapped on the plan body*
	Task		TeamAchieveNode	*transform all tasks with an 'in' Interaction to a TeamAchieveNode*
	Decision		DecisionNode	*one-to-one mapping*
	Merge		ForkNode	*one-to-one mapping*
	in		incomingFlows	*one-to-one mapping*
	out		defaultFlow	*one-to-one mapping*
			establish	*these reasoning methods*
			pass	*are automatically*
			fail	*generated*
	Task		handleEvent	*handle all messages inside a task with an 'out' but no 'in' Interaction*
			postEvents	*post all messages inside a task* with an 'in' Interaction
			roles	*roles a team makes use of are bound in the team plan*

Table 4), that is sent to the respective role in the collaboration the service is involved in (which defines which roles are actually attached to the team that represents the service provider). Thus, the transformation of behavior for service requesters and providers is stereotyped and can be done automatically.

The transformation between the PIM4SOA and AgentMM is finally done using the Atlas Transformation Language (ATL) [14,15] that is a hybrid language designed to express model transformations as described by the MDA$^{\mathrm{TM}}$ approach. The transformation model in ATL is expressed as a set of transformation rules. Tables 3 and 4 illustrate some transformation rules by abstracting from the ATL syntax. The core transformation is described as $ServiceProvider \rightarrow Team$ in Table 3. In this case, for each *ServiceProvider* a *Team* is instantiated that has the same name, performs the same roles and makes use of the roles specified in the *CollaborationUse*, in which it participates, as bound roles. Beside introducing a role in the AgentMM for each role specified in the PIM4SOA, we define a team and two team plans for every role service providers make use of (see $Role \rightarrow Team$ and $Role \rightarrow TeamPlan$ in Table 3). These plans specify how the requested service is invoked and how the corresponding team reacts on a service request.

As discussed before, the process of an PIM4SOA can easily be transformed into team plans (cf. Table 4). As a first approach, we translated the sequential process structure of an PIM4SOA into a sequential team plan.

The final transformation step involves serializing the AgentMM model instance into JACK^{TM} Gcode. This serialization is implemented using MOFScript language [16], which is currently a candidate in the OMG RFP process on MOF Model to Text Transformation. In MOFScript, a set of serialization rules is created following the structure of the source MOF-based metamodel—AgentMM in our case—and the language constructs allow a straight-forward definition of the desired output—Gcode for our purposes.

5 Related Work

This section presents some related contributions in Agent Oriented Modeling, particularly, and Agent Oriented Software Engineering (AOSE) in general.

With regard to modeling languages, the authors of *AgentUML* [11] argue that UML is inappropriate for modeling MAS, so *Agent Interaction Protocols* and *Agent Class Diagrams* are proposed as extensions to UML in order to model these basic features of multiagent systems. Additionally, in [17], the use of *Z* is recommended to overcome the absence of formal semantics of AgentUML.

An additional modeling language approach, the *Agent Modeling Language* (AML) [10], is presented as a semi-formal visual modeling language for the specification of agent systems. However, *AML* does not cover operational semantics since they are considered, by the authors, as a dependency of the specific execution platform.

With respect to methodologies, *Tropos* [9] is an agent-oriented methodology based on the concepts of actor and goal and strongly focused on early requirements. It proposes the use of AgentUML for detailed design and JACK Intelligent Agent^{TM} as implementation platform. In [18], a MDA approach to the transformations in the Tropos methodology is presented.

Prometheus [8] is another known agent-oriented methodology that starts its modeling process by determining the systems percepts and actions. It follows with the definition of *functionalities*. In Prometheus the interaction protocols are modeled using AgentUML and the target implementation platform is BDI-style platforms.

The *Malaca Agent Model* [19] is an approach to Agent Oriented Design using MDA. The *Malaca UML Profile* provides the stereotypes and constraints necessary to create Malaca models on UML modeling tools. In their MDA approach, the transformation is realized from a TROPOS Design Model—as PIM—to a Malaca Model—as PSM.

Related AOSE topics are presented in [12,20,21]. The concept of *Goal-Oriented Interactions* [12,20] presents an interesting way of representing the behavior of agents and provides some additional robustness to the interactions. Cabri et al. propose the *BRAIN Framework* [21] to deal with agent interactions based on the concept of role. In the framework, the description of roles and interactions is realized in an XML notation.

All the mentioned contributions, make valuable points for the specification and modeling tasks in agent systems; however, interoperability among varied agent systems and other technologies is not addressed in these works. Our MDA approach to interoperable agents demonstrates that this objective can be achieved.

6 Conclusions

The paper presented the transformation from PIM to an agent PSM, by explaining how the concepts of a metamodel for SOAs can be transformed to agent related concepts. Therefore an overview of the PIM4SOA, along with its service and process models, was given. Moreover, AgentMM, a metamodel for a specific class of agents, was covered along with its corresponding process model. This transformation that is derived from the mapping of PIM4SOA to AgentMM allows models (i.e., concrete scenarios) that are defined according to the PIM4SOA metamodel to be executed in a flexible, adaptive and generic manner using agent technology.

On one hand, the difference in describing interactions is a challenge when transforming models from PIM4SOA to AgentMM. On the other hand, it also provides chances to actually improve the AgentMM with additional models that are possibly more compact and easier to read.

Our approach shows that interoperability between MAS and other application technologies can be obtained. Further development of the AgentMM could lead to a PIM for agent technologies (PIM4Agents). Since this PIM4Agents would incorporate all relevant high level concepts of the target agent platforms, the interoperability of the generated agent systems would be guaranteed. Moreover, the clear definition of what high level concepts should make part of the PIM4Agents for each particular type of agent technology could prove the greatest contribution of this MDA approach to MAS.

Acknowledgments

The work published in this paper is partly funded by the European Commission through the ATHENA IP (Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications Integrated Project) (IST-507849). The work does not represent the view of the European Commission or the ATHENA consortium, and the authors are solely responsible for the paper's content.

References

1. D'Souza, D.: Model-Driven Architecture and Integration - Opportunities and Challenges, Version 1.1, Kineticum. (2001)
2. Object Management Group (OMG): MDA Guide Version 1.0.1, Document omg/03-06-01, June 2003, http://www.omg.org/docs/omg/03-06-01.pdf (2003)
3. Benguria, G., Larrucea, X., Elvesæter, B., Neple, T., Beardsmore, A., Friess, M.: A platform independent model for service oriented architectures. In: Proceedings of I-ESA Conference. (2006)
4. Singh, M., Huhns, M.: Service Oriented Computing: Semantics, Processes, Agents. John Wiley & Sons, Chichster, West Sussex, UK (2005)
5. Davis, R., Smith, R.G.: Negotiation as a metaphor for distributed problem solving. Artificial Intelligence **20** (1983) 63 – 109
6. Rao, A.S., Georgeff, M.P.: Modeling agents within a BDI-architecture. In Fikes, R., Sandewall, E., eds.: KR'91, Cambridge, Mass., Morgan Kaufmann (1991) 473–484

7. Rao, A.S., Georgeff, M.P.: BDI-agents: from theory to practice. In Lesser, V., ed.: Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco, AAAI Press/The MIT Press (1995)
8. Padgham, L., Winikoff, M.: Prometheus: A Methodology for Developing Intelligent Agents. In Giunchiglia, F., Odell, J., Weiß, G., eds.: AOSE. Volume 2585 of Lecture Notes in Computer Science., Springer (2002) 174–185
9. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: TROPOS: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multiagent Systems **8**(3) (2004)
10. Cervenka, R., Trencanský, I., Calisti, M., Greenwood, D.A.P.: AML: Agent Modeling Language Toward Industry-Grade Agent-Based Modeling. In: AOSE. (2004) 31–46
11. Bauer, B., Müller, J.P., Odell, J.: Agent UML: A Formalism for Specifying Multiagent Software Systems. In: AOSE 2000, Springer-Verlag New York, Inc. (2001) 91–103
12. Cheong, C., Winikoff, M.: Hermes: a methodology for goal oriented agent interactions. In Dignum, F., Dignum, V., Koenig, S., Kraus, S., Singh, M.P., Wooldridge, M., eds.: AAMAS, ACM (2005) 1121–1122
13. AOS: JACK Intelligent Agents, The Agent Oriented Software Group (AOS), http://www.agent-software.com/shared/home/ (2006)
14. ATLAS Group, INRIA & LINA, University of Nantes: INRIA, ATL - The Atlas Transformation Language Home Page, http://www.sciences.univ-nantes.fr/lina/atl/ (2006)
15. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: MoDELS 2005, Montego Bay, Jamaica. (2005)
16. SINTEF ICT: MOFScript, http://www.eclipse.org/gmt/mofscript (2006)
17. Huget, M.P.: Modeling Languages for Multiagent Systems. In: AOSE. (2005)
18. Perini, A., Susi, A.: Automating Model Transformations in Agent-Oriented modelling. In: AOSE. (2005)
19. Amor, M., Fuentes, L., Vallecillo, A.: Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA. In: AOSE. (2004) 93–108
20. Cheong, C., Winikoff, M.: Hermes: Designing Goal-Oriented Agent Interactions. In: AOSE. (2005)
21. Cabri, G., Ferrari, L., Leonardi, L.: Supporting the Development of Multi-Agent Interactions via Roles. In: AOSE. (2005)

Formation of Virtual Organizations Through Negotiation

Mark Hoogendoorn[1] and Catholijn M. Jonker[2]

[1] Vrije Universiteit Amsterdam,
Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
mhoogen@cs.vu.nl
[2] Radboud University Nijmegen,
Nijmegen Institute of Cognition and Information
Montessorilaan 3, 6525 HR Nijmegen, The Netherlands
C.Jonker@nici.ru.nl

Abstract. In this paper negotiation is presented as a solution to the formation of virtual organization in domains with many parties having (partially) unknown constraints and profiles and in which the environment is dynamic by nature. The solution presented is based on the MAGNET negotiation system, for which an extension is presented, that allows for last minute changes and failure management. An efficient algorithm is presented for supplier agents, incorporating preferences, and other constraints related to existing individual plans). Combining the algorithms for supplier agents, with a simple customer agent specification, and the ability to iterate the bidding, MAGNET is extended to deal with domains as described above. A case study in logistics using real data from a logistics company shows the validity of the approach.

1 Introduction

Virtual organizations have been defined as organizations where "complementary resources existing in a number of cooperating companies are left in place, but are integrated to support a particular product effort for as long as it is viable to do so" [7]. Nowadays, companies tend to outsource many non-core operations to upstream and downstream partner firms whose capabilities complement their own [9]. The relationship between such firms precisely complies to the definition of a virtual organization, making it an interesting type of organization to investigate given the current trends in organization theory.

Existence of a virtual organization can be long term or short term, where in the latter case the organization might only be formed to perform a few tasks. Especially for the cases where only a small number of tasks is involved in formation of a virtual organization, the overhead of the formation itself might be relatively large, possibly even causing more time than the task itself. One crucial aspect that for instance needs to be addressed in the formation process is what agents to allocate to what tasks. In order to cope with this problem, techniques from AI are being used to reduce the effort accompanying formation of a virtual organization.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 135 – 146, 2006.

This paper presents the application of one AI technique, namely automated negotiation between agents, to formation of virtual organizations. More in particular, the paper presents a system which enables automated allocation of agents to particular tasks that need to be performed within the virtual organization. The system tries to find a suitable allocation of tasks from two perspectives: (1) that of the agent looking for an agent to perform the task, and (2) that of the agent who can perform the task. Since both can have different, possibly partially conflicting interests, negotiation is most suitable to get to a solution for both parties. Besides the initial formation, the system also has facilities to cope with failure of agents to perform their allotted tasks and to redistribute tasks.

Section 2 presents MAGNET as the negotiation platform the supplier and consumer agents can use to find a solution to their needs. The techniques and extensions needed to be able to use the MAGNET for the dynamic formation of virtual organizations are presented in Section 3. Special attention is paid to obtain robustness with respect to failures in task performance and changes in the environment warranting the change of existing virtual organizations and the formation of new virtual organizations capable to cope with the situation at hand. The system was tested using real data from a logistic company. The test results are presented in Section 4. Section 5 discusses alternative approaches in literature and presents the conclusions.

2 The MAGNET System

This Section describes the negotiation system used as a basis for the development of the system supporting the formation of virtual organizations. The negotiation system used is the MAGNET (for Multi-AGent NEgotiation Testbed) system [4]. In [1] the MAGNET system is described as follows: the MAGNET architecture provides a framework for secure and reliable commerce among self-interested agents. What makes MAGNET particularly suitable is its ability to support negotiation of contracts for tasks that have temporal and precedence constraints [4]. MAGNET shifts much of the burden of market exploration, auction handling, and preliminary decision analysis from human decision-makers to a network of heterogeneous agents. Two types of agent are distinguished within such a network: The *supplier* agent and the *customer* agent. The main interactions between the two agent types are as follows:

- A customer agent issues a *Request for Quotes* (RFQ) which specifies tasks, their precedence relations, and a time line for the bidding process. For each task, a time window is specified giving the earliest time the task can start and the latest time the task can end.
- Supplier agents submit bids. A bid includes a set of tasks, a price, a portion of the price to be paid as a non-refundable deposit, and estimated duration and time window data that reflect supplier resource availability and constrain the customer's scheduling process.

- The customer agent decides which bids to accept. Each task needs to be mapped to exactly one bid (i.e. no free disposal [11]), and the constraints of all awarded bids must be satisfied in the final work schedule. In MAGNET the customer can chose from a collection of winner-determination algorithms (A*, IDA*, simulated annealing, and integer programming).

The customer agent awards bids and specifies the work schedule.

3 Formation of a Virtual Organization

An overview of the activities accompanying the formation of a virtual organization supported by the system introduced in this paper is presented in this Section. Note that for evaluation and communication concerning the negotiation the MAGNET system can be used whereas more specific implementations for the *customer* and *supplier* agent are needed for specific domains such as the formation of virtual organizations.

3.1 High-Level System Overview

A high-level activity diagram of the system is shown in Figure 1. At the starting point the tasks to be fulfilled by the virtual organization come in, which are bundled in an RFQ and sent via the MAGNET system. The RFQ is sent to all *supplier* agents that might want to participate in the virtual organization. These *supplier* agents bid on the tasks they are able to perform and prefer and send a bid including these tasks back via the MAGNET system. After receiving all the bids, the MAGNET system evaluates these and selects the best set of bids possible. In case this set does not fully cover the tasks, an RFQ is sent again. For the bids that are in the set of optimal bids, an award is sent. The *supplier* agent that receives such a reward takes place in the virtual organization and starts executing the tasks, possibly reporting trouble requiring sending another RFQ for the task. Finally, after all tasks have been performed, the virtual organization is terminated.

3.2 Customer Agent

The *customer* part of the system mainly includes the formation of Request for Quotes (RFQs), the sending of awards for bids, and reassignment of tasks which are not properly performed. Tasks in the system include the following elements: intake time, early start time, late start time, deadline, and a task description, including details on the task and constraints. After an RFQ is sent, the *customer* eventually gets a set of bids to be awarded from the MAGNET system. In case there is no bid for a particular task, a new RFQ is sent concerning the particular task. After a task is assigned by means of awarding a bid, the *supplier* agent is placed in the virtual organization and starts to perform the task, which can result in an error report. In case such a report is received, a new RFQ with the task is sent to ensure that the task is eventually performed.

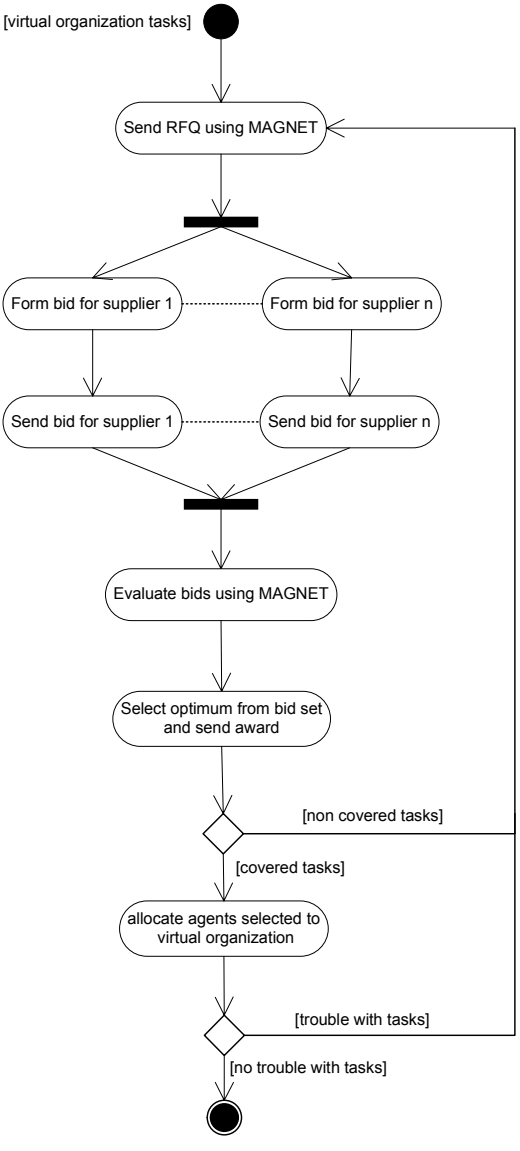


Fig. 1. UML Activity Diagram for the System

3.3 Supplier Agent

The *supplier* agents in the system are assumed to have one particular resource available during a certain time interval. Furthermore, a *supplier* is attributed with a certain preference for particular tasks, for example using the resource for a short time or using it in the beginning of the availability interval. In order to be able to derive

which tasks a *supplier* is to bid on, this Section presents an algorithm which derives which tasks are included in the bid, determines the cost, and finally, determines the time windows to be inserted. The notation used for the algorithm is shown in Table 1.

Table 1. Language used in the pseudo code

Function	Explanation
first_task: RFQ → TASK	The function application provides the task with the earliest early start time in the RFQ.
next_task: RFQ x TASK → TASK	Results in the task with the first earliest start time in the RFQ later than the earliest start time of the specified task.
number_of_tasks: RFQ → INTEGER	The number of tasks in the RFQ.
earliest_start: TASK → TIME	Denoting the earliest start time for executing the task.
latest_start: TASK → TIME	The latest possible start time for executing the task.
expected_duration: TASK → DURATION	The expected duration of executing the task.
latest_finish: TASK → TIME	Denoting the latest possible finish time of the execution of the task.
preference: TASK → REAL	The preference value for the task, a value between 0 and 1.
last_task_before: SCHEDULE x TIME → TASK	The last task in the schedule with a latest finish before the specified time.
next_task: SCHEDULE x TASK → TASK	Specifying the next task in the schedule.
switch_time: TASK x TASK → DURATION	The time needed to switch from one task to another.
determine_risk: REAL → REAL	The risk factor taken, based on the preference for the task.

The algorithm is specified in pseudo code below, a current schedule s from the supplier's perspective with tasks already scheduled is assumed to be present in advance.

```
t = first_task(RFQ)
do{
  before   = last_task_before(s, earliest_start(t))
  after    = next_task(s, before)
  chi      = determine_risk(preference(t))
  duration = chi * (expected_duration(t) + switch_time(before, t) + switch_time(t, after))
  if (earliest_start(t) + duration ≤ latest_start(after))
    if (preference(t) > phi ||
        number_of_tasks(RFQ) == 1)){
      // Add the task to the bid and schedule, set the cost using a particular cost function
    }else{
        // Do not include the task
    }
}while(t = next_task(RFQ,t) && t != NULL)
```

As can be seen, the first task to be performed is taken out of the RFQ. Given the current schedule, the task just ending before the early start time of the current RFQ task is obtained as well as the task after that. Furthermore, based on the preference (a value between 0 and 1) for the current RFQ task, the amount of risk to be taken is determined (e.g. I like this task so much, I will be able to perform it faster than average) represented by χ. Now calculate the *expected* duration for performing the task, which includes switching from the previous task, performing the task itself, and switching to the next task in the schedule. The *assumed* duration to be used in the calculation is obtained by multiplying this with the χ factor. In case the duration added to the earliest start time for task t is before the latest start time of the next task, then the task can in principle fit within the schedule. There is however still the preference of the supplier, which is specified by means of ϕ. ϕ is the threshold for the preference value above which a task is preferred. In case a task is preferred and fits within the current schedule, add the task to the bid. Do the same in case the RFQ contains one single task. This reflects the understanding by the *supplier* that this is a task that really needs to be performed and for which it is hard to get somebody. The global result is that unpopular tasks also will be performed. Once a task is added to the bid, the cost for performing the task are added by means of a cost function. Two cost functions are used in this paper, where the first is simply the *assumed* duration for the task. The second cost function used is the *assumed* duration divided by the preference value, which means a higher price for less preferred tasks. One element not addressed in the algorithm is determination of time windows to be included in the bid, which is specified in pseudo code below.

```
if (chi ≤ 1){
    earliest_start = latest_finish(before) + chi * switch_time(before, t)
    if (earliest_start < earliest_start(t)){
        earliest_start = earliest_start(t)
    }
    latest_start = latest_finish(before) + switch_time(before, t)
    if (latest_start < earliest_start(t)){
       latest_start = earliest_start(t)
    }
}else{
    earliest_start = latest_finish(before) + switch_time(before, t)
    if (earliest_start < earliest_start(t)){
        earliest_start = earliest_start(t)
    }
    latest_start = latest_finish(before) + chi * switch_time(before, t)
    if (latest_start < earliest_start(t)){
       latest_start = earliest_start(t)
    }
}
duration = expected_duration(t) * chi
```

In case the χ value is less than or equal to 1 (i.e. the task is *assumed* to go faster than *expected*), then set the earliest start time to either the earliest start time specified for the task or, in case this is not feasible, to the latest finish time of the task before in the schedule plus the *assumed* switching time. The latest start is set to the latest finish

time of the task before plus the *expected* switch time or, in case before the specified earliest start time, the earliest start time specified in the RFQ. If the value of χ is greater than 1, the earliest and latest start times are calculated just opposite from less than or equal to 1. Finally, the duration is set to the assumed duration.

After having sent a bid, a bid award is possibly received, resulting in the task actually being executed. The schedule is therefore replaced by a schedule including the tasks that have been awarded.

In the execution phase, incidents can occur that require replanning by the *customer* agent (or in similar domains, leading to the supplier agent becoming a customer agent that is seeking another supplier agent to solve his task). Three types of incidents are distinguished: (1) A simple task delay, that requires no replanning; (2) A task failure, the task needs to be performed by another *supplier*; (3) A day failure, all tasks for the day need to be re-planned.

4 Case Study

This section presents the results of a case study performed in order to validate the virtual organization formation approach presented in this paper. First, the domain in which the case study has been performed is described, thereafter the results regarding system performance are presented.

4.1 Case Study Description

In order to obtain experimental results, a choice has been made to use real company data instead of randomly generated data. Using company data has as the advantage that it can be determined how well such a system would work in a real environment instead of an artificially created one. The data has been obtained from a company within the field of logistics. This area is particularly interesting for application of the system due to the movement of several companies to so called Fourth Party Logistics (4PL), see e.g., [2]. A 4PL logistics company is an intermediate link within the chain of transporting goods, it closes contracts with large parties to arrange the logistics across the entire supply chain of the organization. 4PL companies have a limit amount, or possible even no trucks of their own (see e.g. [6]). They therefore have contracts with a number of trucking companies which they can call in case they need a truck for a particular order. The price for such a trip is negotiated over the phone. In the case study, the 4PL does not negotiate with the trucking companies through a scheduling officer, but directly with the truck drivers of that company. In this way the truck drivers get a higher responsibility for creating a revenue for the company they work for and they get the opportunity to guard their own preferences. Hence, the 4PL company is the *customer* in the system described in the previous section, and the trucks are the *supplier* agents, where a formation of a virtual organization for the transportation of certain goods is the goal of the negotiations.

The data used for the experiments concerns transportation of containers, of which only one can be carried at the same time by a truck. As a result, trucks can only perform tasks in sequential order and not in parallel. Furthermore, there are different

types of containers: 20 feet and 40 feet containers, both of them can only be carried by a truck suitable for that particular type. Each of the tasks contain an intake time (around which the order to transport the container comes in, and thus the time at which an RFQ can already be sent), an early start time (when the container becomes available), a deadline (when the container needs to be delivered), and a start and end location. Precedence constraints are present as well in case a container has to be transported along several locations. The data obtained from the company mainly concerns container transports from one of the container terminals at the port of Rotterdam (there are several such terminals in the port) to a particular customer, after which the container needs to be returned to a certain location. Typically, about 20 orders are received each day, most of which require a pickup early in the morning. For the usage of the system presented in Sections 2 and 3, each truck is seen as a separate *supplier* where the resource is in this case the ability to transport a particular type of container. On average, about 10 trucks are available as *suppliers* per day. Trucks have a start location at which they are located at the beginning of the day (typically close to the port of Rotterdam), and have a start and end time (e.g. the trucks starts at 9 am and stops at 5 pm). Preferences of trucks are found in the different pickup and destination locations, the length of the trip required to perform the task, and the start and end times of the tasks. As a result of interviews with personnel from the data providing company, these preferences have been determined for each truck, based on the driver assigned to it. The real cost for performing a task is set to the travel time in minutes to perform the task (i.e. driving to the pickup location, performing the task, and returning from the destination location). Note that this can differ from the price actually put in the bid for the task.

4.2 Case Study Results

In order to evaluate the effectiveness of the system, simulation runs have been performed using the real life data from the trucking domain as described before. For this purpose, the logs of the order system of one representative week has been used. Using this data, the system is evaluated from two perspectives. First, the time needed to evaluate the bids is measured, to see whether this evaluation process itself is not a bottleneck within the virtual organization formation process. The algorithm for the supplier agents can be run in parallel, which is not the case for the customer agent. Another perspective from which the system is evaluated is to see how different cost functions and preference thresholds influence the overall satisfaction of the *supplier* agents within the system.

Algorithm Performance. First, the performance of the evaluation algorithm during the simulation runs is presented. Note that these results are specific results for the characteristics of the data. For more generic results on algorithm performance and a comparison between different algorithms, see [3]. The experiments have been run on a Sun UltraSPARC IIIi 1062 MHz CPU with 2 GB memory. Figure 2 shows the results of the IDA* algorithm used for the case study for RFQs with varying amount of tasks.

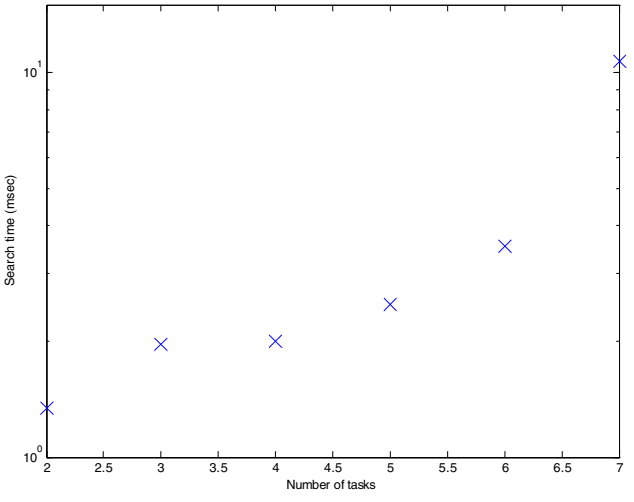


Fig. 2. IDA* search time for different number of tasks

Furthermore, Table 2 shows more detailed characteristics for the evaluation process.

Table 2. Evaluation characteristics

Number of tasks	Average number of bids	Average number of tasks in bid	Search time IDA* (msec)
2	3.59	1.00	1.35
3	6.05	1.07	1.98
4	6.00	1.00	2.00
5	7.25	1.08	2.50
6	8.00	1.25	3.54
7	7.63	1.01	10.69

As can be seen in the table, the average amount of tasks per bid is always close to one, which is due to the fact that an RFQ in the trucking domain typically specifies several tasks which need to be performed in parallel and, as already stated in the introduction of the case study, the trucks cannot execute tasks in parallel. Since only full bids can be awarded, they therefore often only bid for one task. As the graph shows, also for the RFQ's with the largest amount of tasks observed in the data (i.e. seven tasks in one RFQ) the evaluation algorithm generates a solution in just over 10 milliseconds. For a more extensive discussion on the scalability of the IDA* algorithm within the MAGNET system, see [3].

Supplier Satisfaction. Besides the evaluation time, the satisfaction of *suppliers* is another element which has been investigated. The satisfaction of the suppliers is measured in the average preference for the tasks they get awarded. Two parameters can be varied regarding this satisfaction, namely the threshold value ϕ and the

function for cost to be included in the bid (i.e. assumed duration or assumed duration divided by the preference). Figure 2 shows the satisfaction of the different agents for both cost functions for varying ϕ values. Note that despite the threshold for bidding on tasks, tasks can still be bid upon in case only one task is included in the RFQ. As a result, the satisfaction can be below the threshold value set.

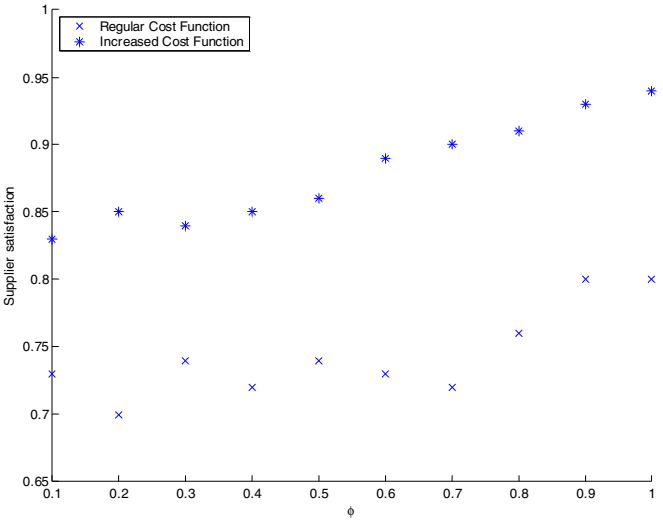


Fig. 3. Driver satisfaction for varying ϕ values

As can be seen in Figure 3, the increase of the price in case a task is not preferred is shown to be effective in the simulation runs of the case study. Having such a preference requires a less strict setting of the preference threshold ϕ for bidding on a task still obtaining a reasonable satisfaction rate. When looking at the regular price option, satisfaction is much lower once the value for ϕ decreases. An additional performance measure is of course the efficiency of the solution found, which in the trucking domain can be measured by means of the amount of effective driving (the amount of driving for a task divided by the total amount of driving). In the simulation runs, no correlation was found between the setting for the preference and the effectiveness of driving. On average, 62% of all driving was effective.

5 Discussion

This paper presents an approach for the formation of virtual organizations in highly dynamic environments which require a low overhead for the formation process of the organization. The approach allows for the formation of such an organization without the different parties needing to have knowledge about each others constraints and profiles. The approach is based upon an existing negotiation system called the

MAGNET system which is extended with specific implementations for the *supplier* and *customer* agents for the formation of virtual organizations. The implementation of the *supplier* agent incorporates preferences for tasks as well as schedules specifying the tasks to be performed. In a case study in the trucking domain, the paper shows that the evaluation algorithms incorporated in the MAGNET system scale well, requiring a minimal time for the evaluation process. Furthermore, reflecting the preference of a task in the price bid for that task in the algorithm increases the overall satisfaction of the *supplier* agents.

In the field of virtual organizations, negotiation systems have been introduced and used as well. In [8] a virtual office system is mentioned called *SmartProcurement* which is said to initiate the formation of a virtual organization by means of an electronic or human request for quotation (RFQ). Thereafter, a purchasing agent acquires a list of agents which are known vendors of the requested item and sends the RFQ to the vendors. Subsequently, the bids are evaluated and a bid is selected, informing the vendor agent upon acceptance. The approach is however more meant as a framework to support such negotiation, similar to the MAGNET system, not as a specific implementation of the agents themselves.

Besides the MAGNET system, more negotiation systems have been developed. The advantage of the MAGNET system is the market infrastructure in between the *supplier* and the *customer* agent whereas most other negotiation systems focus on direct agent to agent negotiation [12, 5] (from [1]). Based on the MAGNET system more extensive *supplier* agents have been developed [1], however these agents have not been tested with real life data. Furthermore, [1] does not focus on the formation of virtual organizations.

Team or coalition formation is another related field. Different protocols for the formation of coalitions are compared in [13]. Variations of such protocols go from local to social utility based negotiation systems. The authors show that increased social context can improve system performance. The agents are however required to share meta-level information before they allocate resources. In the trucking domain, however, agents do not want to share such meta-level information, as they might be competitors. Therefore the approach presented in [13] is not feasible in domains in which the agents represent competitors.

Different role-allocation and reallocation algorithms are compared in [10] The comparison is based on for the framework developed for the Role-based Markov Team Decision Problem. In the future the same framework could be applied to compare the approach presented in this paper with other role-allocation algorithms with respect to the corporate data for the trucking domain.

Acknowledgements

The authors would like to thank Maria Gini from the Department of Computer Science and Engineering at the University of Minnesota for the fruitful discussions. Furthermore, the authors wish to thank the anonymous reviewers for their useful comments that helped to further improve the paper.

References

1. Botman, S., Hoogendoorn, M., Bud, V., Jaiswal, A., Hawkins, S., Kryzhnyaya, Y., Pearce, J., Schoolcraft, A., Sigvartsen, E., Collins, J., and Gini, M., Design of supplier agents for an auction-based market. In: Giorgini, P., Giorgini, P. Lesperance, Y., Wagner, G., Yu, E. (eds.), Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS 2002 @ AAMAS-02), July 2002.
2. Briggs, P., The hand-off: the future of outsourced logistics may be found in the latest buzzword [Fourth Party Logistics], Canadian Transportation Logistics 102(5), pp. 18, 1999.
3. Collins, J., Solving Combinatorial Auctions with Temporal Constraints in Economic Agents. PhD thesis, University of Minnesota, June 2002.
4. Collins, J., Gini, M., and Mobasher, B., Multi-agent negotiation using combinatorial auctions with precedence constraints. Technical Report 02-009, University of Minnesota, Department of Computer Science and Engineering, Minneapolis, Minnesota, February 2002.
5. Fatima, S.S. and Wooldridge, M.. Adaptive task resources allocation in multi-agent systems. In Proc. of the Fifth Int'l Conf. on Autonomous Agents, pp. 537-544, 2001.
6. Foster, T., 4PLs: The next generation of supply chain outsourcing? Logistics Management and Distribution Report 38(4), pp. 35, 1999.
7. Goldman, S., Nagel, R., Preiss, K., Agile Competitors and Virtual Organizations, Van Nostrand Reinhold, New York, 1995.
8. O'Leary, D.E., Kuokka, D., and Plant, R., Artificial Intelligence and Virtual Organizations, Communications of the ACM 40(1), pp. 52-59, 1997.
9. Miles, E.R., Snow, C.C., Mathews, J.A., Miles, G., and Coleman, H.J., Organizing in the knowledge age: Anticipating the cellular form, Academy of Management Executive 11(4), pp. 7-20, 1997.
10. Nair, R., Tambe, M., Marsella, S., Role Allocation and Reallocation in Multiagent Teams : Towards a Practical Analysis, In: Proceedings of the Second Conference on Autonomous Agent and Multi-Agent Systems (AAMAS 2003), pp. 552-559, ACM Press, 2003.
11. Nisan, N., Bidding and allocation in combinatorial auctions. In *1999* NWU Microeconomics Workshop, 1999.
12. Sandholm, T.W., Negotiation Among Self-Interested Computationally Limited Agents PhD thesis, Department of Computer Science, University of Massachusetts at Amherst, 1996.
13. Sims, M., Goldman, C.V., and Lesser, V., Self-Organization through Bottom-up Coalition Formation, In: Proceedings of the Second Conference on Autonomous Agent and Multi-Agent Systems (AAMAS 2003), pp. ACM Press, 2003.

Continuations and Behavior Components Engineering in Multi-Agent Systems

Denis Jouvin

LIRIS, université Claude Bernard Lyon 1
denis.jouvin@liris.cnrs.fr

Abstract. Continuations are a well established programming concept, allowing to explicitly capture and resume the current program state. They are present in several functional programming languages (such as Scheme), in concurrent models (such as process calculi or Hewitt actor model), and more recently in dynamic programming languages (such as Ruby, Smalltalk, Python, and even Javascript or Java). They have been applied to automaton programming, cooperative threads, compilation techniques, and have lastly raised interest in web application programming. This paper shows how this concept happens to be especially useful and elegant to program agent behaviors (or behavioral components), while increasing code readability and ease of writing. The proposed approach especially facilitates modular interaction protocol implementation, one of the main difficulties in conversational agents engineering.

Keywords: Continuations; conversational multi-agent systems; agent oriented software engineering; behavioral component; continuation-based automatons.

1 Introduction

Multi-agent systems (MAS for short) are a programming paradigm especially well suited to model complex systems. In MAS, agents usually interact by means of an elaborate, normalized interaction model. The choice of this interaction model will condition numerous prized MAS properties, such as agent autonomy, interoperability, robustness, and self-organization, as mentioned by Luck et al. in [7].

However, these elaborate interaction models are also a significant source of difficulties in conversational MAS implementations, as shown in former research [6]. To address these difficulties, various multi-agent platforms propose componential approaches, where agents are provided with reusable behavioral components managing partially or completely their behavior with respect to a given conversation type.

In this paper we show how a well established programming concept, continuations, can significantly facilitate the writing of such behavioral components, while bringing better flexibility and performance in their implementation.

Section 2 introduces continuations, their applications and variants. Section 3 identifies implementation issues in conversational MAS, and the classical solutions proposed by multi-agent platforms. We present our continuation based approach in section 4, followed in section 5 by the results of an example prototype. We conclude on our approach benefits, limitations, and integration into existing MAS platforms.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 147 – 158, 2006.

2 Introduction to Continuations

Continuation are a well established programming concept, described for example by Strachey et al. [13] in the early seventies, which were originally introduced in functional programming languages such as Scheme or ML, in the actor model of Hewitt [4], and in various process calculi.

The principle consists in capturing the currently executing program state into a variable or artifact that can be manipulated programmatically, called the *continuation*, in order to be able to resume later on the program from this state by *activating* this continuation. More precisely, we designate by *execution context* the program state, in reference to the execution context of a thread or process, and by *continuation* the object or artifact allowing to activate (i.e. switch to) this execution context.

Some informal definitions of continuations are found in the literature, such as "the rest of the program", or "`goto` with parameters", however none of them are really satisfactory: the first implies a linear execution of the program, that should be unwind beforehand, and the second does not express that local variables and call stack are captured in a continuation.

Although there exist variants and restrictions of continuations, such as coroutines and generators (see section 2.3); continuations remain the most general form, as shown by Haynes et al. in [5] and Allison in [1].

Continuations are somehow related to threads, since they capture all, or part of, the stack, including local variables, in addition to the "program counter"; however their activation is done programmatically rather than by an operating system scheduler.

2.1 Languages Supporting Continuations Natively

For illustrative purpose, we will borrow examples from various languages that support continuations natively, such as Scheme or Ruby, as well as a restricted form of continuations to start with: Python's generators.

```
def simple_generator(max):
    i = 1
    yield "let's count.."
    while i < max:
        yield "Odd %d" % i
        i = i+1
        yield "Even %d" % i
        i = i+1
    yield "the end"
```

```
for s in simple_generator(4):
    print s

let's count..
Odd 1
Even 2
Odd 3
Even 4
The end
```

Fig. 1. Simple Python generator example: on the left the generator definition; on the right the code to display the successive values returned by the generator, and the console output in italic

As figure 1 suggests, a generator behaves the same way as an iterator, written however as a function that returns successive values, using the `yield` keyword, while memorizing its current execution state between each call (see Mertz [8]).

The advantage of generators is that the programmer does not have to explicitly manage the iterator state using object attributes. Generators make it possible to use the native flow control structures of the language (if, loops, etc.) to manage transitions between the states of the underlying automaton, as shown by Mertz in [9].

```
callcc {|cont|
  for i in 0..4
    print "\n#{i}: "
    for j in i*5...(i+1)*5
      cont.call() if j == 17
      printf "%3d", j
    end
  end
}
print "\n"
```

```
0:   0  1  2  3  4
1:   5  6  7  8  9
2:  10 11 12 13 14
3:  15 16
```

Fig. 2. Continuation example in Ruby, with the console output on the right (in italic)

The `callcc` (call with current continuation) primitive, in figure 2, calls the anonymous closure defined by the next bloc between brackets, and passes the continuation as the parameter `cont`. The continuation activation, `cont.call()`, results in the premature exit of the two `for` loops, by taking the control flow just after the bloc.

Note that this program could be translated in Scheme very easily, by using the Scheme primitive `call-with-current-continuation`. Smalltalk, Haskel, and Perl 6 are examples of other programming languages that support continuation natively in their most complete implementations.

2.2 Continuations as Extensions of Dynamic Object-Based Languages

More recently, continuations have been added as libraries or extensions to common object based (imperative) languages like Javascript or Java. Since this type of extension is related to flow control and low level stack manipulation, it requires either a modification of the interpreter, class instrumentation and manipulation techniques, or source code transformations, as shown by Pettyjohn et al. in [11].

In such languages, a continuation is relative to a callable object or "objectized" function (see figure 3), on the contrary to Ruby or functional languages where it is "global" to the program. Examples of continuation frameworks or extensions are:

- Flowscript[1], a modified version of the Rhino Javascript interpreter implementing continuations, part of the apache Cocoon framework since version 2;
- The RIFE web application framework[2], which allows limited continuations;
- And the Javaflow framework[3], which implements continuations in Java, using a dynamic class instrumentation technique, and which we will use hereafter.

[1] http://cocoon.apache.org/2.1/
[2] http://rifers.org/
[3] http://jakarta.apache.org/commons/sandbox/javaflow/

```
public abstract class Generator<T>
       implements Runnable, Iterator<T>, Iterable<T> {

   private transient T result;
   private Continuation current = Continuation.startWith(this);

   public T next() {
      if(!hasNext())
         throw new NoSuchElementException();
      T retval = result;
      current = Continuation.continueWith(current);
      return retval;
   }

   protected void yield(T param) {
      result = param;
      Continuation.suspend();
   }

   public boolean hasNext() { return current != null; }
   public Iterator<T> iterator() { return this; }
   public void remove() { throw new UnsupportedOperationException(); }
}
```

Fig. 3. Basic implementation of Python style generators in Java, using Javaflow continuations

Figure 3 gives a simple implementation of Python style generators in Java, using Javaflow. On the contrary to Ruby or Scheme, the continuation captures here the execution state *inside* the `Runnable` object passed to the continuation capture routine, and not the execution state with respect to the calling function.

In this example, it is necessary to temporarily store the return values of the generator in the `result` attribute, since Javaflow does not handle continuation parameters and return values.

```
public class SimpleGenerator
       extends Generator<String> {

   private final int max;

   public SimpleGenerator(int max) {
      this.max = max;
   }

   public static void main(String... a) {
      for(String s:new SimpleGenerator(5))
         System.out.println(s);
   }
```

```
   public void run() {
      yield("let's count..");
      for(int i=1; i<max; i++)
      {
         yield("Odd " + i++);
         yield("Even "+ i);
      }
      yield("the end");
   }
}
```

Fig. 4. Simple generator example in Java, instrumented by Javaflow

Figure 4 reuses the example of figure 1, translated in Java, and using the generator class introduced in figure 3. The console output is identical. To work properly, these java classes need to be instrumented by Javaflow at compile or class loading time.

2.3 Variants and Applications

The possibilities offered by continuations and closures are numerous: for example, one can rewrite existing or custom flow control structures, like simulating a `goto` in Scheme, etc. However the most useful applications of continuations are:

- The implementation and composition of automatons (for example in syntactic parsers and validators), that we will further develop in section 4.1;
- Coroutines and cooperative threads. Coroutines are functions that retain their execution state when calling another coroutine. On the contrary to generators, a coroutine never returns after capturing its state, but explicitly calls another coroutine that will in turn be activated. Coroutines thus allow defining cooperative threads, with explicit context switching. The main advantage is performance: cooperative threads are lightweight and fast compared to real preemptive threads. This technique is not suited when preemption is necessary, but is well suited for non preemptive concurrent models and event programming;
- Continuation based Web application programming, as described by Tate et al. in [14], which has recently drawn a lot of attention. Historically introduced by the Web application framework Seaside, this technique has been followed in other frameworks like RIFE or Cocoon. The reason is simple: the interaction between a Web server and a web browser usually takes the form of a stateful conversation, for which it is necessary to store the state on the server. The routines controlling the page flow can use continuations to store this state implicitly, relieving the programmer from this tedious and error prone task. Page flow is then described as a simple instruction flow in a script. This principle being simple and elegant, it has quickly been adopted by many Web application developers.

Remark. Such a behavior may be obtained using threads; however this is usually not feasible in practice, since it would require one thread for each possible conversational state during a session, which would result in too many threads, especially if the server implements the web browser back button behavior.

3 Conversational Agent Engineering

We define here a conversational agent as an agent whose behavior requires memorizing the local context of the ongoing interactions, which we refer to as *conversation*. This context may take various forms. In this work we focus on agents communicating by asynchronous messages, organized into inter-related message sequences, the conversations, between several participants, in the context of a collective task.

3.1 Problem Definition

Experience shows that, though bringing many useful properties to the system that they compose, conversational agents are difficult to implement. In particular, we can identify the following difficulties related to conversation management:

- *Multi-party conversation parallelism*. In a multi-party conversation, it is necessary to combine several behaviors simultaneously, corresponding to the bilateral sub-conversations with each participant separately, and possibly synchronize them;
- *Internal parallelism*. A bilateral conversation with a single participant can itself comprise some form of parallelism: it may fork into several parallel or interlaced branches of the conversation;
- *Protocol error management*. A great part of the complexity implied by conversation management comes from the asynchronous error management: protocol errors, timeouts, etc. This aspect being transverse, it is difficult to modularize and reuse;

Remark. The programming languages used in MAS platform are not concurrent languages, and, thus, are not designed to handle elaborate parallel processing. Concurrent languages, such as Erlang for example, could address some of these issues; however they suffer from other limitations, and are unfortunately not popular enough to have been chosen in existing MAS platforms.

It is interesting to observe that most MAS platforms propose componential approaches in order to promote the reusing of behavioral components, which confirms the importance of the difficulties identified above in MAS design and implementation. The problem then becomes: how to define behavioral components in a modular way, that is to say, decoupled and encapsulated in reusable components; and how to combine and synchronize several behavioral components consistently in agents.

Considering for example FIPA interaction protocols, a FIPA compliant platform will typically propose abstract behavioral components implementing partially these protocols, and bound to the agent specific code using a platform specific composition technique (callbacks, inheritance, or event model).

Behavior component libraries help the programmer to deal with delicate aspects of conversation management, such as: timeouts handling, protocol error handling, managing a group of participants in parallel, etc. These libraries are largely used in MAS platforms. Two strategies are possible to implement such behavioral components:

- Either a thread is assigned to each parallelizable branch of each behavioral component. A waiting state is then implemented as a blocking method on this thread, and the conversational state is defined by all the threads execution contexts. This option is not always feasible since, as we discussed it in the case of web applications, it may involve too many threads. Not only is the number of available threads limited by memory and operating system, but too many threads may result in a performance loss due to an excessive scheduling with respect to the actual processing;
- Or the conversational state of the behavioral component is stored explicitly. This implies that the corresponding code must be fragmented according to the transitions and structure of the underlying automaton, so that it may be executed step by step, by interrupting and resuming its execution at each state. Typically, distinct methods or objects will represent the various transitions, themselves associated to objects representing the states. This technique is the most widely used, because it gives more control on the activation of the behavior components, and on the number of threads allocated. The FIPA-OS platform, described by Poslad et al. in [12], uses a global thread pool to activate the behavioral components of agents (called *task* in FIPA-OS), whereas Jade, a MAS platform described by Bellifemine et al. [2], assigns by default one thread per agent.

3.2 Automaton Behavioral Components

In Jade, behavioral components are named *behaviors*, and are all inherited from the abstract `Behavior` class. Figure 5 shows a simple 3 states behavior example, illustrating the explicit state management, and resulting code fragmentation, in the `switch` statement. Although not obvious at first, this behavior passes three times on the three states before termination.

```
class my3StepBehaviour extends SimpleBehaviour {
   private final int FIRST = 1, SECOND = 2, THIRD = 3;
   private int state = FIRST, numberOfExecutions = 3;

   public void action() {
      switch (state) {
      case FIRST: {op1(); state = SECOND; break;}
      case SECOND:{op2(); state = THIRD;  break;}
      case THIRD: {op3(); state = FIRST;
         numberOfExecutions--; break;}
      }
   }

   public boolean done(){
      return (!(numberOfExecutions>0));
   }

   private void op1(){ System.out.println("Step 1"); }
   private void op2(){ System.out.println("Step 2"); }
   private void op3(){ System.out.println("Step 3"); }
}
```

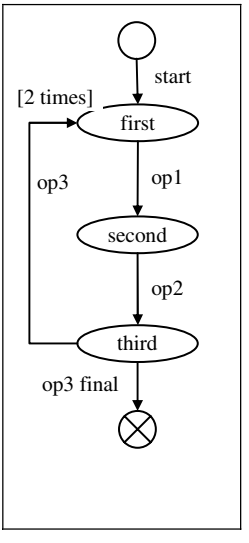


Fig. 5. Simple Jade behavior example: 3 states automaton, with the corresponding finite state automaton diagram on the right

In this example the `SimpleBehavior` class does not contribute much to the behavior; however, other available behaviors, like `ContractNetInitiator`, or `AchieveREInitiator`, define abstract implementations of common interaction protocols, bound to the agent specific code by inheritance or callback. Some composite behaviors allow various types of composition of children behavior, for example `SequenceBehavior` or `ParallelBehavior`. Inter-thread synchronization is usually not necessary in Jade since transitions are executed sequentially.

Comparable techniques can be found in other MAS platforms. Let us mention for example the multi-plan automatons of the Bond platform, by Bölöni et al. [3]; the finite state automatons of the Zeus platform, by Nwana et al. [10]; the FIPA-OS tasks, by Poslad et al. [12]; or the automatons designed with SEdit in the MadKIT platform[4].

The multi-plan automatons of Bond, in particular, handle the parallelism using a simplified parallel sub-automatons composition model; whereas FIPA-OS tasks are by default executed in parallel: they may launch other sub-tasks, but this requires careful multithread synchronization.

[4] http://www.madkit.org/

4 Continuation-Based Approach

As we stated above, the need to manage parallelism and asynchrony without requiring dedicated threads implicates that behavioral components be manipulated like automatons, without blocking on threads. Besides, since existing MAS platforms comply with this strategy, our approach should also comply with it to facilitate integration.

On the other hand, being able to write stateful behaviors as simple routines, leveraging native control flow structures of the hosting language, without scattering the code, would also make MAS programmers' life easier.

With respect to these issues, continuations offer a particularly elegant solution that combines the best of both worlds. Indeed, using continuations in MAS behavioral components for conversation management allows to:

- Capture the conversation or conversation branch state implicitly in the continuation, including call stack and local variables, so that the component may be used as an automaton, without having to explicitly manipulate its state;
- Get rid of inter-thread synchronization problems, since this model is not preemptive and transitions are executed sequentially.

4.1 Continuation-Based Automaton Behavioral Component

A continuation represents a program states, usually as an immutable object in object based language, but is not itself an automaton: it requires an encapsulating automaton object, similar to the generator implementation presented in section 2.1. Figure 6 shows a minimal automaton abstract base class, containing:

- a `current` attribute, storing a single current continuation for this automaton;
- the `activate()` method, the automaton step by step activation point, that updates the current continuation with a new one, corresponding to the new captured state;
- a continuation capturing method, `yield()`, meant to be used in implementations of the abstract `run()` method, or other methods called by `run()`, that interrupts the normal control flow and jumps to the `activate()` method;

```
public abstract class Automaton implements Runnable {
   private Continuation current = Continuation.startSuspendedWith(this);

   protected void yield() { Continuation.suspend(); }

   public void activate() {
      if(current != null)
         current = Continuation.continueWith(current);
   }
}
```

Fig. 6. Simple continuation-based behavioral automaton abstract class, in Java, using Javaflow. Each time `activate()` is executed, the automaton advances one step. Each step terminates either by invoking `yield()`, or by the normal end of the `run()` method. In both cases this gives back control to `activate()`, which updates the continuation and returns normally.

In order to be usable in an actual agent implementation, however, these primitives are not sufficient: a way to consume events, like messages or timeouts, in the `run()` method is necessary. In the case of a conversational agent platform, an agent is in principle provided with a message queue. A behavioral component specific message queue may also be considered, associated to a message dispatch mechanism at the agent level. Such dispatching will typically be based on message response or conversation identification parameters, such as FIPA `conversation-id` message parameter.

4.2 Example and Comparison

In comparison with the Jade component of figure 5, the version shown on figure 7 is much easier to read: it does not contain any object attribute to store and manage state information, nor does it require code fragmentation. The loop is materialized by a normal java `for` statement, increasing readability and reducing error proneness.

```
public class My3Step extends Automaton {
   public void run() {
      for(int i=0; i<3; i++) {
         System.out.println("Step 1"); yield();
         System.out.println("Step 2"); yield();
         System.out.println("Step 3"); yield();
      }
   }
}
```

Fig. 7. Continuation-based version of the 3 states automaton behavior of section 3.2, figure 5

4.3 Pseudo Parallelism and Synchronization Primitives

In order to address the parallelism requirements mentioned in section 3.1, it is necessary to introduce synchronization and pseudo-parallelism mechanisms in our automaton abstract classes. We use the term "pseudo-parallelism" since transitions will *in fine* always be executed sequentially. However, sub-automaton states corresponding to parallel branch of the conversation need to be stored and updated separately.

The case of multi-bilateral conversations (i.e. conversations involving an initiator and a group of participants), in particular, is quite common in protocols such as negotiations or auctions. This case requires interacting in parallel with *n* participants, possibly synchronizing at key steps of the protocol, even if the various participants all play the same role and are thus governed by the same rules in the protocol.

We introduce two primitives to handle this kind of parallelism:

- `parallelize()`, a primitive allowing to switch to parallel mode, duplicating the current automaton in *n* sub-automatons, one per participant;
- and `join()`, the reverse primitive, usable only in parallel mode, that waits for all sub-automatons to reach this point (note that this does *not* entail blocking on a thread), before switching back to synchronous mode;

5 Testing Framework

In the context of this work, we have realized a testing prototype and framework based on Javaflow. Two abstract classes, `BilateralRole` and `MultiBilateralRole` (see table 1) implement the primitives mentioned in section 4.1 and 4.3. [5]

Table 1. Primitives provided by `BilateralRole` and `MultiBilateralRole`

Method signature	Description and side effect
`void activate()`	Automaton activation point
`void yield()`	Continuation capture, interrupts control flow
`Message receive()` `Message receive(Message.Type... types)`	Read the next event or message (possibly typed). Calls `yield()` if no event is available
`void parallelize()`	Switch to parallel mode, duplicate automaton
`void join()`	Switch back to synchronous mode

5.1 Auction and Handshake Protocol Example

In order to represent both forms of parallelism related to conversation management, we have defined four behavioral components, corresponding respectively to: the initiator and participant roles of a simple multi-bilateral handshake protocol, consisting in a linear sequence of *inform* message exchanges; and the initiator and participant roles of an English auction protocol, comparable to FIPA-English-Auction.

```
public void run() {
   send("auction start", inform);
   bestOffer = min;
   do {
      send(bestOffer, cfp);
      nbAnswered = 0;
      parallelize();
      Message msg = receive(propose, inform, refuse);
      if(msg.getType() != propose)
         return;
      nbAnswered++;
      int offer = (Integer) msg.getContent();
      if(offer > bestOffer) {
         bestOffer = offer;
         winner = getRunningAgent();
      }
      join();
      for(Agent a:getInterlocutors())
         send(bestOffer, a == winner? accept: reject, a);
   } while(nbAnswered > 1);
}
```

`propose`, `inform`, `refuse`, `cfp`, `accept` and `reject` are values of the enumerated type `Message.Type`

Parallel section: here the automaton is duplicated in n sub-automatons, managed by *n* continuations, until all of them reach `join()`, which switches back to synchronous mode

Fig. 8. *`run()`* method of the behavior component `EnglishAuctionInitiator`

[5] By lack of space, the algorithm used to implement `parallelize()` and `join()` are not detailed here. The corresponding Java code may be downloaded from the author's web page.

In figure 8 the whole management of this auction protocol role is handled in a few simple lines of code, which demonstrates the applicability and elegance of this approach. An equivalent behavior, implemented using classical explicit finite state automaton, would require numerous states, conditions and switches, and would result in a fragmented code, difficult to read and debug.

Note that the variables `bestOffer`, `nbAnswered` and winner are here attributes of the `EnglishAuctionInitiator` class. These attributes are necessary here to communicate between sub-automatons in parallel mode, since Javaflow does not support parameter passing (see the generator example in section 2.1).

The participant roles, quite symmetrical to the initiator roles, of course do not require the use of `parallelize()` and `join()`, since they don't comprise parallelism.

5.2 Performance

Similarly to cooperative thread with respect to preemptive threads, a non preemptive activation of agents' behavior components, using continuation, give better performances. To evaluate this gain, we have compared the execution time of two versions of our prototype: a preemptive version with one thread per behavior component and per agent, and a non preemptive one, using round-robin activation and continuations. In this test the initiator and participant agents combine the handshake and auction behaviors. The results exhibit an average gain of 50% using Javaflow continuations.

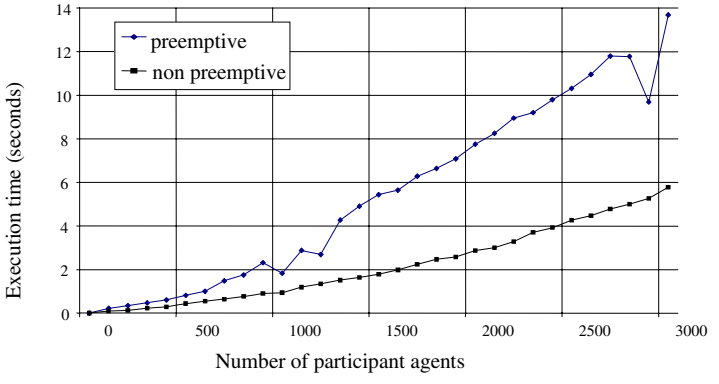


Fig. 9. Execution time, function of the number of participant agents. The upper curve corresponds to the preemptive version, the lower curve to the non preemptive version.

6 Conclusion and Future Works

In this paper we have shown how to use continuations to facilitate conversational agents' behavioral components engineering. This approach allows an elegant, concise and intuitive coding of agents behavior dynamics, in the form of automatons written as "resumable" functions similar to coroutines or generators, while relieving the programmer from the explicit management of the automaton state and transitions. This approach also allows benefiting from the host language native flow control structures.

Its integration into existing platforms only depends on the support of continuations in the host language, and is quite straightforward. It gives to the designer a great flexibility in the mode of activation of agents and their behavior components.

In the general case, it allows to get rid of multithreading synchronization problems, but can still be combined with multithreading if necessary.

The main perspective to this work is the integration to the Jade platform, by defining a `ContinuationBehavior`. If the adoption of continuations in the MAS community follows its adoption in the Web application programming community, they will represent a promising new technology for agents, and become a common practice in agent behavioral components engineering.

References

1. Allison, L.: Continuations Implement Generators and Streams. In The Computer Journal, volume 33(5), 1990. Oxford Journals, 460-465
2. Bellifemine, F., Poggi, A., Rimassa, G.: JADE – A FIPA-compliant agent framework. In proceedings of the International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 1999), London. 97-108.
3. Bölöni, L., Marinescu, D.: A Multi-Plane State Machine Agent Model. Acts of the International Conference on Autonomous Agents (AA 2000). Barcelona, Spain. ACM.
4. Hewitt, C.: Viewing control structures as patterns of passing messages. In Artificial Intelligence, volume 8(3), 1977. Elsevier. 323-364.
5. Haynes, C.T., Friedman, D. P., Wand, M.: Obtaining coroutines with continuations. In Computer Languages, volume 11(3), 1986. 143-153.
6. Jouvin, D., Hassas, S.: Role Delegation as Multi-Agent Oriented Dynamic Composition. In proceedings of the Intl. Workshop on Agent Technology and Software Engineering (AgeS 2002, collocated with NOD 2002), Erfurt, Germany.
7. Luck, M., McBurney, P., Preist, C.: Agent Technology: Enabling Next Generation Computing, A Roadmap for Agent Based Computing. Agentlink II report, 2003.
8. Mertz, D., Charming Python: Iterators and simple generators. IBM technical report, 2001. http://www-128.ibm.com/developerworks/library/l-pycon.html
9. Mertz, D., Charming Python: Generator-based state machines. IBM technical report, 2002 http://www-128.ibm.com/developerworks/library/l-pygen.html
10. Nwana, H., Ndumu, D., Lee, L., Collis, J.: ZEUS: A Toolkit and Approach for Building Distributed Multi-Agent Systems. In proceedings of the International Conference on Autonomous Agents (AA 1999), Seattle, USA. ACM press.
11. Pettyjohn, G, Clements, J., Marshall, J., Krishnamurthi, S., Felleisen, M.: Continuations from Generalized Stack Inspection. In proceedings of the ACM SIGPLAN International Conference on Functional Programming (ICFP 2005), Tallinn, Estonia. ACM press.
12. Poslad, S., Buckle, P., Hadingham, R., The FIPA-OS Agent Platform Open Source for Open Standards. In proc. of International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000), Manchester, UK.
13. Strachey, C., Wadsworth, C.: Continuations: A mathematical semantics for handling full jumps. In Programming Research Group Technical Monograph PRG-11, Oxford, 1974. Reedited in Higher-Order and Symbolic Computation, volume 13(1/2), 2000. 135-152.
14. Tate, B.: Crossing borders: Continuations, Web development, and Java programming, A stateful model for programmers, a stateless experience for users. IBM technical report, 2006 http://www-128.ibm.com/developerworks/java/library/j-cb03216/

Evaluating Mobile Agent Platform Security

Axel Bürkle, Alice Hertel, Wilmuth Müller, and Martin Wieser

Fraunhofer Institute for Information and Data Processing,
Fraunhoferstraße 1, 76131 Karlsruhe, Germany
{axel.buerkle, alice.hertel, wilmuth.mueller,
martin.wieser}@iitb.fraunhofer.de

Abstract. Agent mobility requires additional security standards. While the theoretical aspects of mobile agent security have been widely studied, there are few studies about the security standards of current agent platforms. In this paper, test cases are proposed to assess agent platform security. These tests focus on malicious agents trying to attack other agents or the agency. Currently, they have been carried out for two agent platforms: JADE and SeMoA. These tests show which of the known theoretical security problems are relevant in practice. Furthermore, they reveal how these problems were addressed by the respective platform and what security flaws are present.

1 Introduction

Over the past years software agents in general and mobile agents in particular have become more and more important in many areas of computer science such as distributed systems, autonomous systems and robotics as well as artificial intelligence.

Mobile agents are a special sort of software agents that possess the ability to migrate to other hosts. In contrast to classical distributed systems, where processes are bound to the host they were launched on, mobile agents can transfer their code and context to another host where their execution continues.

Especially the technical advantages of mobile agents such as delegation of tasks, asynchronous processing, adaptable service interfaces, and code shipping versus data shipping provide an interesting approach to distributed systems [1].

Application domains where mobile agents have proven to be valuable are telecommunication applications [2], IP-network configuration and management [3], sensor networks [4], electronic commerce, information retrieval [5], [6], mobile computing, and dynamic deployment of software, just to mention a few.

Despite these proven demonstrations of valuable contributions for building systems and applications in different domains, the number of commercial applications built with mobile agents is still rather small. One of the principle reasons for this is security concerns. Companies and individuals are skeptical of allowing an uncontrollable piece of code to appear on their machines and execute, which is basically the same as what a virus does [7]. Agent users are worried about the confidentiality and integrity of sensitive data carried by the agent, e.g. an

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 159–171, 2006.

electronic purse in an e-commerce application, when the agent is running on a remote and potentially malicious platform.

All application domains mentioned above demand guarantees of agent behavior and safe interaction with the underlying operating system and other involved legacy systems.

In the last decade a number of publications focused on security aspects of mobile code in general and mobile agents in particular. Some of them analyze theoretical security problems [8], [9], [10] while others propose mechanisms and architectures to overcome these problems [11], [12], [13], [14].

While the theoretical aspects have been widely studied, there is a lack of studies about the practical realization of security concepts in available agent platforms. This paper introduces test cases for the evaluation of security mechanisms of mobile agent platforms. They are based on the results of theoretical studies on mobile agent security and designed with respect to practical relevance.

The agent plattforms covered in this paper are JADE (Java Agent DEvelopment Framework) [15], an open source software distributed by Telecom Italia, and SeMoA (Secure Mobile Agents) [16], a freely available agent platform developed and distributed by Fraunhofer IGD. SeMoA was specially designed with focus on security aspects of mobile agents.

The paper is organized as follows: in Sect. 2, a taxonomy of possible attacks is introduced. Sects. 3 and 4 present test cases and results for JADE and SeMoA. Finally, Sect. 5 discusses the findings of our study.

2 Taxonomy of Possible Attacks

With mobile agents there are four different sorts of attacks according to whether the agent or the agency is malicious and whether the agent or the agency is attacked [1], [9].

2.1 Malicious Agents Attacking the Hosting Agency

- Denial of Service (DoS) attacks: The agent excessively consumes the resources of the agency such as memory, CPU cycles or bandwidth so that the agency is not able to provide its services to other agents.
- Unauthorized access to the agency's data: The agent tries to access confidential data, e.g. keystores or policy files (usually, these are locally stored on the agency's hard disk(s)), or tries to manipulate the agency's management mechanisms.
- Masquerading: The agent masquerades as another agent with more permissions and gains access to sensitive data or services.
- Complex attacks through cooperation with other agents [17]

2.2 Malicious Agents Attacking Other Agents (Located on the Same Agency)

- Changing the other agent's state or task
- Reading or manipulating the other agents' data

- Masking its identity to deceive other agents and gain sensitive information from them or using services on behalf of other agents without paying
- Retarding another agent or detaining it from fulfilling its task [17]
- Denial of Service attacks on other agents by sending spam messages

2.3 Malicious Agencies Attacking Other Agencies

- Eaves-dropping on the communication between two agencies and capturing agents to extract useful information from the agents' state or code
- Traffic analysis attempting to find patterns in the communication between two agencies to derive assumed behaviors based on these patterns
- Sending an agent to attack the agency - this could be either a malicious agent or an agent manipulated to act maliciously

2.4 Malicious Agencies Attacking Agents

- Accessing the agent's data: Reading confidential data, e.g. private keys, or manipulating the collected data
- Accessing the agent's code and / or workflow: Reading the migration path or the algorithms; permanently or temporarily changing the agent's behavior for the benefit of the malicious agency or to damage of other agencies [1].
- Delaying or even denying the agent's execution.
- Cut-and-Paste attack: The agency cuts data items from the agent and pastes it into a new agent. If this data is encrypted, the agent can migrate to the agency where a decryption is possible and come back with the decrypted data.

2.5 Security Solutions

There are multiple mechanisms for providing security against the above mentioned attacks. However, to list them all is out of the scope of this paper, so we refer to [1], [14], [18] and present only the most important security solutions:

- Encryption: The data the agent is carrying, or even the whole agent, is encrypted to prevent unauthorized access to data.
- Digital signatures and certificates: Using PKI (Public Key Infrastructure) is the most common way for communication partners to authenticate against one another, i.e. agent against agency and vice versa.
- Central management of security mechanisms and access authorizations: Each agency has its own security management where authentication and authorization of incoming agents and the monitoring of their actions during their stay is managed.
- Sandbox: Every agent is executed in a secure environment and any attempt to access anything outside this environment is strictly controlled by a security manager.
- Secure Socket Layer (SSL): This technique provides a secure way for mobile agents to migrate from agency to agency.

2.6 Test Purposes

From the above mentioned attacks we chose to consider only those initiated by malicious agents, as we did not intend to modify any code of the platforms. Consequently we disregarded the case of malicious agencies. Furthermore, we did not analyze the security mechanisms of the underlying operating system, the JVM (Java Virtual Machine) or the network. Neither did we examine the security of encryption algorithms or signatures. Our purpose was to test existing platforms for mobile agents on their ability to ensure security in practice.

So far, our tests were carried out for the platforms JADE and SeMoA. JADE has been chosen since it is probably the most widely used agent platform today. SeMoA was considered because of its focus on security. It was specially designed with respect to security and seems to provide the most elaborate security concept. It is interesting to see how these two platforms compare.

All test cases are based on the before described taxonomy of attacks. However, they were adapted to the individual platform in order to meet the peculiarity of each platform. E.g. JADE and SeMoA greatly differ in their architecture and their security concepts. Instead of developing a homogeneous test methodology for all platforms we considered it more appropriate to approach platform security from a practical point of view. The following sections present the test cases and test results for JADE and SeMoA.

3 JADE Security Test

JADE provides security features through an add-on [19]. It comprises the following services:

- **SecurityService**: Authentication using the corresponding Java functionality.
- **PermissionService**: Granting permissions to access Java libraries (using the Java Authentication and Authorization Service JAAS) and to perform agent-specific actions (e.g. start, kill or clone or send-to agents).
- **SignatureService**: Signing of messages to avoid falsification.
- **EncryptionService**: Encryption of messages to avoid unauthorized reading.

To test the JADE security add-on (we tested "Version 3[1].3" with JADE Version 3.3), especially in conjunction with mobile agents, we implemented a specific test environment. This test environment uses the JADE test suite tool [20] to manage the execution of the test cases. Test cases are small pieces of code to test a specific behavior of the security add-on; a sequence of logically related test cases is called a test suite. The JADE test suite tool provides a graphical user interface for configuring and starting test suites and base classes for test suite and test case agents.

3.1 Test Agents

The JADE security test environment comprises the following agents:

- a test suite agent for each test suite to parameterize and start the test cases, and to clean up the platform after test case execution,
- a test case agent for every test case which executes the test procedure step by step,
- a database access agent, which represents the normal application part and provides a service to read metadata of an image from a database according to the read conditions contained in the read request, and
- a user agent, which is able to represent a regular user agent who retrieves image data using the service of the database access agent, as well as a malicious user agent who tries to disturb the regular users and to attack the platform. The user agent is generated and stimulated by the currently active test case agent to behave as regular or as malicious user.

3.2 Test System

The test system consists of three JADE containers: The test container, where the test suite and test case agents are located, the user container, where the user agents are generated, and the main container, where the default JADE agents (ams, df, rma) and the database access agent are located. The main and the test container on one side, and the user container on the other side, have different owners, so that different owner-specific permissions can be granted within the policy file. Fig. 1 shows the testbed architecture.

3.3 Test Cases

Denial of Service (DoS) Tests. Denial of Service test cases check if malicious mobile agents are able to disturb or even prevent the tasks of regular agents by overloading the CPU or by extensive use of operating system resources. All DoS test cases start with generating a regular user agent in the user container and migrating it to the main container. Then a malicious user agent is started in the user container and migrated to the main container as well. The regular user agent then requests image data from the database access agent and the execution time is measured. Then, if requested by the test case, the malicious user agent is cloned to produce many malicious user agents. The malicious user agent(s) start(s) its (their) work, while the regular user agent executes image requests again. The execution time is measured again and compared with the execution time in the undisturbed case to find out if the DoS attack succeeded. The following test cases for DoS tests were implemented:

- TC1: Recursively cloning malicious agents.
- TC2: Malicious agents try to overload the agency by flooding the ams (agent management system) agent with agent search requests.

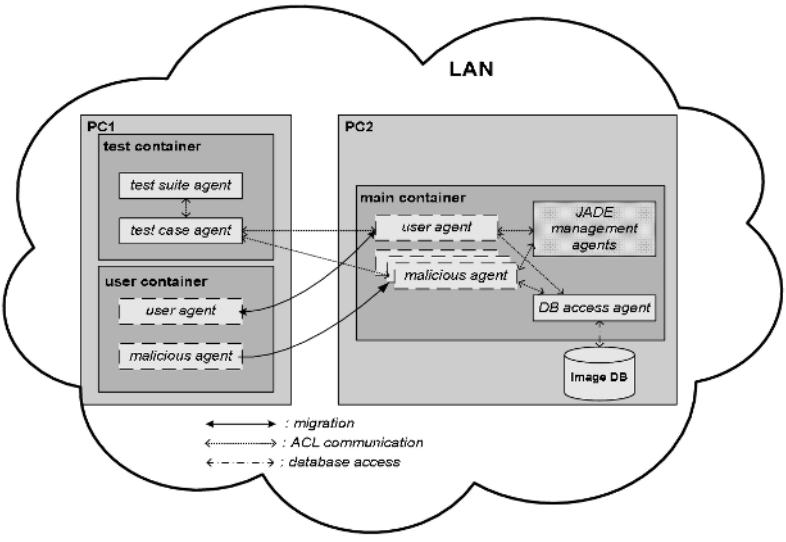


Fig. 1. JADE security testbed

- TC3: Malicious agents activate non-blocking behaviors, resulting in endless loops for many threads.
- TC4: Malicious agents send a message to many receivers, forcing message decoding for all receiving agents at the same time.

Unauthorized Access Tests. Unauthorized access test cases check if mobile agents can access vital functionality of the agent platform or of the runtime environment so that they can sabotage the platform operation or access confidential information. Unauthorized access test cases start with generating a regular user agent in the user container, migrate it to the main container, then a malicious user agent is started in the user container and migrated to the main container as well. After that the malicious user agent tries to attack the platform. The test case ends with removing all user agents from the main container. The following unauthorized access test cases were implemented:

- TC5: Try to modify the policy file.
- TC6: Try to replace the Java security manager.
- TC7: Try to kill another agent.
- TC8: Try to deregister another agent at the ams agent (i.e. deregister from white pages).
- TC9: Try to deregister another agent at the df agent (i.e. deregister from yellow pages).
- TC10: Try to create a new container.
- TC11: Try to kill the JADE platform.

Agent Attack Tests. Agent attack test cases check if malicious mobile agents are able to attack regular agents operating in the same container. The test case procedure is similar to the one used for DoS test cases. The following agent attack test cases are currently implemented:

- TC12: A malicious agent sends a lot of dummy requests to a regular user agent, trying to prevent it from doing its work.
- TC13: A malicious agent sends a lot of spam messages (*inform* messages with no useful content) to a regular user agent, trying to prevent it from doing its work.
- TC14: A malicious agent tries to suspend a regular agent.
- TC15: A malicious agent tries to send a signed message with a fake sender ID.

3.4 Test Results

All test cases have been carried out in four different environments to take into account and study hardware and operating system specific effects. The four test setups were (cf. Fig. 1):

- PC2 was a laptop computer (Pentium M 760, 2.0 GHz, 1 GB RAM) runnig Windows XP Professional
- PC2 was a PC (Pentium D 830, 3.0 GHz, 2 GB RAM) running Windows XP Professional 64 Bit
- PC2 was a PC (Pentium D 830, 3.0 GHz, 2 GB RAM) running SuSE Linux 10.0 (64 Bit)
- PC2 was a HP workstation (PA-8800, 900 MHz, 4 GB RAM) running HP-UX 11.0

The effectiveness of the DoS attacks (TC1 - TC4) and the attacks against other agents (TC12 and TC13) was measured by comparing the execution time of the regular agent in the undisturbed scenario to the execution time when under attack. Fig. 2 shows the mean execution times in milliseconds for 100 iterations of each test case in the respective environment. In case of the spamming attack, the user agent was completely blocked on the laptop. On the workstation, the mean execution time was beyond 30 seconds.

To build the JADE permission service, the JADE security add-on uses the Java security system and adds some agent-specific permission checks. Assuming that the Java security system and the JADE-specific add-ons cannot be corrupted (for example by falsifying the identity of an agent), and if the policy file is edited carefully, most of the described attacks can be prevented, except the following:

- Recursively cloning and non-blocking behaviors cannot be prevented.
- Registering at and deregistering from yellow pages can only be allowed for all agents or for none.

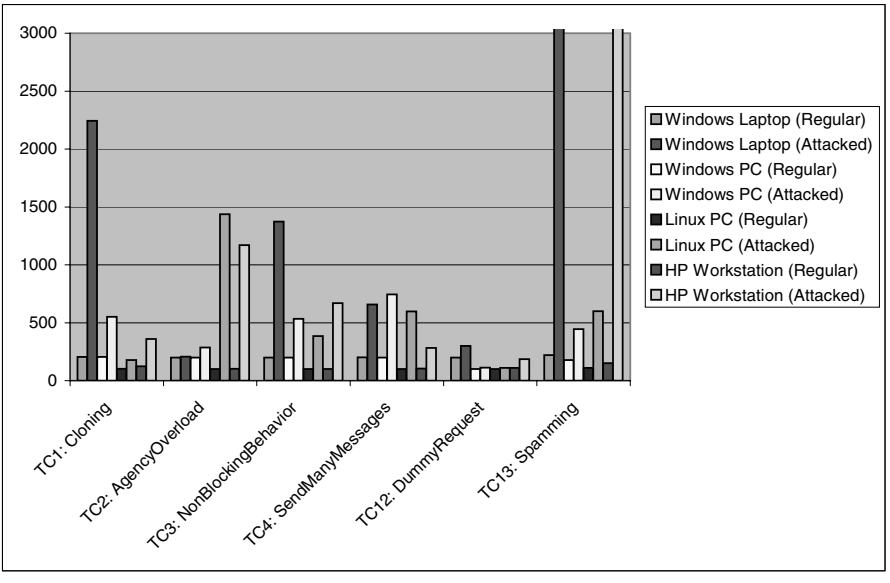


Fig. 2. JADE DoS test results (execution time im msecs)

- Spam and dummy requests cannot be avoided.
- During migration, the ownership of the agent is lost (non-persistent data) and can be easily replaced by an arbitrary ownership, which is used to validate the send-to permission. With the send-to permission obtained by fraud it is possible to deregister any other agent at the df.
- It is possible for an agent to generate a new pair of keys using a fake agent name in order to sign messages with a fake sender ID.

4 SeMoA Security Test

SeMoA provides several mechanisms in order to prevent attacks in terms of a layered security architecture. The first layer is the transport layer, where a protocol such as TLS or SSL can be used for agent transport. In the second layer an incoming or outgoing agent has to pass a pipeline consisting of several security filters. Each filter can accept or reject an agent. Furthermore this layer checks an incoming agent's signatures, decrypts it and finally assigns permissions to it [21]. In this context, Java's permission classes as well as SeMoA-specific classes can be used. An important example for the latter is the EnvironmentPermission. It can be granted for a specific path in an agency's environment and can allow an agent to lookup, publish or retract information or services. The assignment of permissions is done according to a configurable role-based security policy. If an agent tries to execute an action without the corresponding permission, access is denied and an exception is thrown.

After passing the filter pipeline successfully, every agent gets its own class loader. The class loader supports loading classes bundled with the agent and those that are specified as URLs in the agent's static resources. The classes to be loaded are verified against a list of cryptographic hash values signed by the agent's owner. This verification represents layer three. As a fourth security layer, a so-called sandbox is created for each agent: it gets its own threadgroup and is though strictly separated from other agents [21].

Every agent's static part is signed by its owner and the whole agent is signed again by every server that forwards it. Moreover, each agent gets a globally unique "implicit" name which is generated by means of its owner's signature [21].

4.1 Test System

For our tests we used the SeMoA distribution "semoa_complete_050812.zip", released on August 12, 2005. We created a simple agent (the user agent) that migrates from its home agency (AgencyA) to another agency (AgencyB) and requests a database lookup service there (a search for some images in an image database with the images' URLs as a result). After getting the results the agent migrates back to AgencyA and displays the URLs within a GUI. A malicious agent now migrates from a third agency (AgencyC) to AgencyB and attacks either the agency or the reporting agent (Fig. 3). In order to detect the impact of the attacks we launch the user agent in an endless loop and each time quantify the duration it needs to fulfill its task.

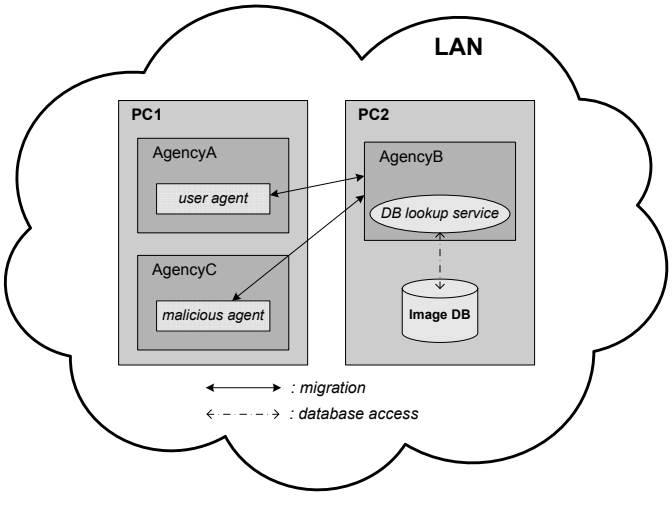


Fig. 3. SeMoA testbed

4.2 Test Cases

Denial of Service (DoS) Tests. In the majority of cases, an agent needs special permissions to be able to attack an agency. We assume that the

agency is not vulnerable unless these permissions are granted to the malicious agent.

Possible targets for DoS attacks are the agency's environment, the agency's computing resources and the Java Virtual Machine.

- TC1: Consuming the agency's computing resources (the malicious agent launches agents from AgencyC to AgencyB; each of these agents starts an endless loop where it just increments a counter).
- TC2: Consuming the agency's memory (nesting AWT-Threads).
- TC3: Spamming the agency with agents (as SeMoA does not provide a cloning mechanism we simulate it in terms of the malicious agent endlessly launching agents from AgencyC to AgencyB).
- TC4: Overloading the agency by too many database accesses (invoking the database access service in an endless loop)
- TC5: Spamming the agency by publishing too many objects (requires an *EnvironmentPermission* with a wildcard for publishing objects or services on AgencyB).
- TC6: Incorrect code (synchronization on the *Thread*-class).
- TC7: Shutdown of the JVM (executing *System.exit(0)*)

Unauthorized Access Tests. The malicious agent migrates to AgencyB and tries to directly access files in the agency's file system using Java's *FileInputStream*. This is possible with a corresponding *FilePermission* which can be granted for read and/or write for individual files or directories.

To access the database on AgencyB the malicious agent can use JDBC or invoke the database lookup service. Either way at least the permissions to access the data and the database driver (again *FilePermissions*) are required.

Another target for unauthorized access are the agency's management mechanisms. SeMoA has several management mechanisms which are located in the agency's environment under specific paths and are therefore only accessible if the *EnvironmentPermissions* for these paths exist. An exception is the policy file which can be manipulated like any file if and only if a corresponding *FilePermission* exists. The most important management mechanisms are:

- The *policy file* which contains all roles and permissions.
- The *vicinity service* which shows all available SeMoA servers in the LAN.
- The *FarSight service* which shows all available SeMoA servers within the network.
- The *ATLAS* (Agent Tracking and Location Service) which serves to track agents; an *ATLAS* client is integrated in each SeMoA server.
- The *security filters*
- The */agents/active* path which contains the contexts of all agents that reside on an agency.

For these management mechanisms we decided to implement the following attacks:

– TC8: Modifying the policy file.
– TC9: Replacing the policy filter.
– TC10: Replacing the Java security manager.
– TC11: Deregistering an agent from the */agents/active* path.

Agent Attack Tests. Attacking an agent in SeMoA is only possible if its implicit name is known. This name can be found either through accessing the agent's context with the *EnvironmentPermission* for the */agents/active* path or if the agent itself publishes its name via a service (if an agents wants to receive messages from other agents it must let these agents know its implicit name, which serves as the agent's address).

– TC12: Spamming the agent with messages / blocking the agent.
– TC13: Manipulating the agent's resources (changing the result of the database lookup).

4.3 Test Results

Fig. 4 shows the effectiveness of the DoS attacks (TC1 - TC5) in four test setups analogous to Sect. 3.4. Note, that for these test cases all necessary permissions have been granted.

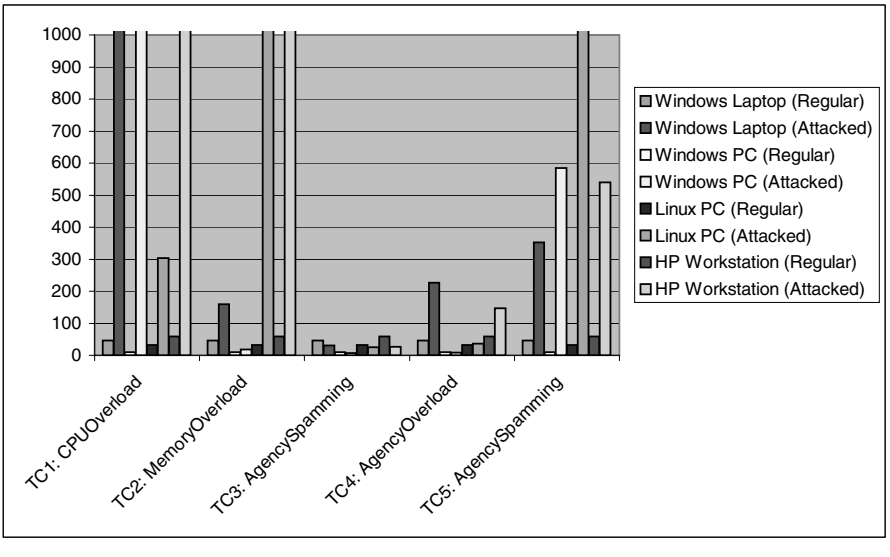


Fig. 4. SeMoA DoS test results (execution time im msecs)

Most attacks can be prevented by correctly configuring the security policy. Critical permissions like *EnvironmentPermission* for the security and management

mechanisms should only be granted to an agency's administrator and permissions for publishing objects in an agency's environment should not be granted with wildcards. Nevertheless, some attacks cannot be avoided:

- Too many agents on one agency are not critical if they terminate soon, but if they execute an endless loop, this heavily burdens the agency.
- The synchronization on the *Thread*-class is a widely known drawback of Java and cannot be prevented.
- The nesting of AWT-Threads is a crucial attack which crashes not only the agency but also the whole operating system. This attack cannot be prevented by the security policy.
- Spamming an agent cannot be prevented if the agent regularly reads its messages (otherwise the sender gets the message "recipient not available"). While the spamming itself does not cause much harm, it can result in blocking the agent, because all messages are stored in the agent's resources. So the agent can become too big to migrate. This can be avoided by emptying the mailbox before migrating.

5 Conclusion

The tests showed that the implemented security mechanisms of the evaluated agent platforms can prevent several of the theoretically possible attacks provided that the access permissions are configured appropriately. Nevertheless, DoS-attacks and spamming cannot be prevented completely. Granting permissions in a very restrictive way avoids some of the DoS-attacks on the SeMoA agent platform. It depends on the application scenario and system environment at hand, as to whether those permissions are necessary for the agents to perform their intended task.

The tests revealed some critical flaws in the security mechanisms of JADE if used in combination with agent mobility. Especially the possibility to fake the owner of an agent when migrating enables several severe attacks and undermines the available security measures. The SeMoA agent platform is well protected against most of the tested attacks, but is inferior with respect to agent communication and cooperation mechanisms.

The next steps will be to examine further agent platforms and define test cases for malicious agency scenarios.

References

1. Braun, P., Rossak, W.: Mobile Agents. Basic Concepts, Mobility Models & the Tracy Toolkit. dpunkt.verlag (2005)
2. Karmouch, A., Magedanz, T., Delgado, J., eds.: Proc. of the 4th Int. Workshop on Mobile Agents for Telecommunication Applications. Volume 2521 of LNCS., Springer (2002)

3. Yang, K., Galis, A., Guo, X., Liu, D.: Rule-Driven Mobile Intelligent Agents for Real-Time Configuration of IP Networks. In: Knowledge-Based Intelligent Information and Engineering Systems: 7th Int. Conf., KES 2003. Volume 2773 of LNCS., Oxford, UK, Springer (2003) 921 – 928
4. Fok, C., Roman, G., Lu, C.: Mobile Agent Middleware for Sensor Networks: An Application Case Study. In: Proc. of Fourth Int. Symposium on Information Processing in Sensor Networks, IEEE CNF 2005 (2005) 382 – 387
5. Brewington, B., Gray, R., Moizumi, K., Kotz, D., Cybenko, G., Rus, D.: Mobile agents in distributed information retrieval. Intelligent Information Agents, Springer (1999)
6. Thati, P., Chang, P., Agha, G.: Crawlets: Agents for high performance web search engine. In Picco, G.P., ed.: Mobile Agents, Proc. of the 5th Int. Conf. (MA 2001). Volume 2240 of LNCS., Atlanta, USA, Springer (2001) 119–134
7. Geirland, J.: The Feature: Mobile Intelligent Agents. `http://www.thefeature.com/article?articleid=26051` (2002)
8. Gray, R., Kotz, D., Cybenko, G., Rus, D.: D'Agents : Security in a Multiple-Language, Mobile-Agent System. In Vigna, G., ed.: Mobile Agents and Security. LNCS, Springer (1998) 154–187
9. Jansen, W., Karygiannis, T.: Mobile Agent Security. Special Publication 800-19, NIST (1999)
10. Roth, V.: Programming Satan's Agents. In Fischer, K., Hutter, D., eds.: Proc. of the 1st Int. Workshop on Secure Mobile Multi-Agent Systems, SEMAS 2001, Elsevier (2002)
11. Hohl, F.: Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In: Mobile Agents and Security. Volume 1419 of LNCS., Springer (1998) 92–113
12. Jansen, W.: A Privilege Management Scheme for Mobile Agent Systems. Electronic Notes in Theoretical Computer Science, SEMAS 2001, First International Workshop on Security of Mobile Multiagent Systems **63** (2002)
13. Tschudin, C.: Mobile Agent Security. In Klusch, M., ed.: Intelligent information agents: agent based information discovery and management in the Internet, Chapt. 18, Springer (1999)
14. Vigna, G.: Protecting Mobile Agents Through Tracing. In: Proc. of the 3rd ECOOP Workshop on Mobile Object Systems, Jyvalskyla, Finland (1997)
15. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.: JADE - A White Paper. `http://jade.tilab.com` (2003)
16. SeMoA. `http://www.semoa.org` (2006)
17. Santana Torrellas, G.: A Network Security Architectural Approach for Systems Integrity using Multi Agent Systems Engineering. Int. Symposium on Parallel Architectures, Algorithms and Networks (ISPAN) (2004)
18. Borselius, N.: Mobile Agent Security. Electronics & Communication Engineering Journal (2002)
19. JADE Board: JADE Security Guide. `http://jade.tilab.com` (2005)
20. Cortese, E., Caire, G., Bochicchio, R.: JADE Test Suite User Guide. `http://jade.tilab.com` (2004)
21. Roth, V., Jalali, M., Pinsdorf, U.: Secure Mobile Agents (SeMoA). `http://www.inigraphics.net/press/brochures/sec_broch/sec/Security.pdf` (2006)

A New Model for Trust and Reputation Management with an Ontology Based Approach for Similarity Between Tasks*

Alberto Caballero, Juan A. Botia, and Antonio F. Gomez-Skarmeta

Universidad de Murcia. Campus Espinardo. Murcia, Espana
albe_cu@yahoo.com, {juanbot, skarmeta}@um.es

Abstract. This paper proposes a new trust and reputation model to assist decision making process into agents in P2P environments, taking WSMO as the base for definition of tasks to contract. This work shows the integration of trust and reputation model and WSMO in two ways: 1) how agents use WSMO as ontology to define their requirements, responses, domain-dependent features and metrics; and 2) how the Web services discovery process in WSMO may be improved using trust and reputation criteria given by the model from data stored by consumer agents in previous interactions.

1 Introduction

In P2P environments, the peers interact in a decentralized manner trying to obtain the solution for a given problem. For instance, peers can be providers or consumers of resources [6,13]. Each node or agent may expose very different behaviors, for this reason it is possible that consumers would want to contract only providers with the best behaviors. For that, it is necessary that each node manages his own updated model about the rest of nodes in the system. Trust and reputation based models can help to separate good and bad nodes.

The paper proposes a new model to manage trust and reputation values that an agent has about the rest of agents associated to the realization of a given task. These values are obtained from the agent experience and the information interchanged between agents. The experience of each agent must represent the satisfaction that this agent obtained from others. The way to measure the satisfaction may be very different according to the application domain, and representations for tasks and responses. Many times, the task, that agents negotiate, is a service request; and the response, the description of the service that satisfies it. This way, task and response representations using WSMO [14] may be very useful in order to define the domain-dependent features of the model. The model can be used in environments where agents take WSMO as ontology of reference.

Moreover, the proposed trust and reputation model carries out the Web service discovery process in a more flexible and intelligent way, using the previous

* This paper is supported by the Programme Alban scholarship No.E05D049799CU and also by the Spanish Ministry of Education and Science by the Research Project TIN-2005-08501-C03-02.

K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 172–183, 2006.

knowledge of the system. Also, it identifies and uses some WSMO elements to obtain several measures such as satisfaction of the task given the response, and similarity between two tasks.

The rest of the paper is organized as follows: section 2 introduces the most important elements treated in previous trust and reputation works that are taken into account in our model; section 3 explains the role of WSMO as conceptual framework of the model. The proposed trust and reputation model is described in section 4. Section 5 shows, using an example, how WSMO elements may be used in order to determine the task satisfaction and similarity between tasks. In the final section, we draw conclusions and ongoing research work.

2 Antecedents on Trust and Reputation

A great amount of trust and reputation models considers trust as an emergent property of direct interactions between agents and assume that agents interact many times [8,14].Trust within an agent is calculated based on this performance in past interactions using expressions that use measurable quantities. REGRET system proposes a trust model based on direct experiences and reputations, providing measures of reliability for these concepts [10].

Reputation may be viewed as an aggregation of opinions of members of the community about one agent. Some authors propose to obtain some ratings from social networks and a procedure to aggregate them to obtain a unique reputation value. This way, it may be consider a community subset, through concepts like *groups* or *neighbors*, to take only closest agents for a specific link [11,10,17].

Our work only considers decision making based on interaction patterns, taking into account that a very simple trust model must be characterized by the following three features: (1) it is possible to calculate trust and reputation values to indicate who trusts who, (2) these values are based on the experience of the system taken from past interactions; (3) it is possible to refine this value based on new acquired knowledge added to the experience of the agent.

Generally, trust and reputation values are obtained as global values only associated to a peer [3,5,7,13,16], but it is logical to suppose that these values must be associated also to the specification of tasks that agents need to delegate. This way, Griffiths proposes a model to manage trust between agents with respect to a particular task [4]. But, in some cases, it is possible that an agent does not have enough information to produce a trust value for a given task, but he knows instead the previous partner behavior performing similar tasks. It may obtain an approximate trust value for the specified task using available trust information about similar tasks.The way to estimate trust using the information about similar tasks is one contribution of this work (please see section 4.3).

3 Representing Contracting Tasks with WSMO

WSMO offers a conceptual framework for ontologies and Web services descriptions. WSMO consists on four main elements: ontologies (that define a common

terminology, used by other elements, providing concepts and relationships between these concepts), Web services (that represent the computational entities providing access to services), goals (that represent the user desires) and mediators (that solve interoperability problems between the rest of elements) [14].

WSMO is a suitable framework to represent the knowledge structures needed by a trust model for P2P environments based on Web services. Service requests may be represented by tasks using the WSMO concept of "Goal". In other hand, the response (describing the Web service that satisfies the task) may be represented using the concept of "Web Service" given by WSMO.

We will use this conceptual framework because 1) it is a new standard proposal, enhancing existing standards in order to describe ontologies, that can be used to represent a broad range of situations where users need to find the most suitable resource in P2P environments based on Web services; 2) it offers great facilities to Web service representation and discovery from different variants according the application requirements; 3) Web services, goals and other elements have some non-functional properties that can be used to calculate trust and quality in Web service discovery process; and 4) the Web service discovery process in WSMO can be improved with the use of trust and reputation models taking into account the system previous experience and behavior exposed.

In line with reason 3, we identify some interesting non-functional properties of Web services and goals to manage trust, quality, costs, etc.:

- Accuracy - numbers of errors generated in a certain time interval.
- Network-Related QoS - network delay, delay variation and/or message loss.
- Performance - throughput, latency, execution time, and transaction time.
- Reliability - number of failures of the Web service in a certain time interval.
- Robustness - number of invalid inputs for which the Web service still function correctly.
- Scalability - number of solved requests in a certain time interval.
- Trust - the trust worthiness of a Web service.

The trust model, presented in the next section, uses service discovery based on simple semantic descriptions of services as a good and simple method to evaluate the quality of a given response. This needs that the agent stores its satisfaction degree for each initiated interaction. Stored information can be used to enhance the WSMO discovery process in later system interactions. This way, the proposed trust and reputation model allows to find the most suitable Web service, in an intelligent form, using knowledge about past experiences.

4 Trust/Reputation Model

The main goal of the proposed model is to offer mechanisms to support adaptive negotiations between agents. It enables mechanisms to decide which are the agents with which it is necessary to negotiate based on the calculation of a value of confidence, that is associated with the specification of the task that it is necessary to contract.

Given the decentralized nature of P2P environments, the model must follow a distributed approach. Each agent has its own bases of experiences to obtain trust values and to interact with its neighbors if it needs to calculate reputation.

When an interaction is finalized, the initiator agent (i.e. the agent that began the interaction) stores interaction data into a binnacle. If a_i is the initiator agent and a_j is the contracted agent to execute the task, the interaction data is represented as a tuple into the set (Initiator's Experience of Trust):

$$IET_i^{(t)} = \{(a_j, s_k, et_{i,j,k,l}) | a_j \in A, s_k \in S, et_{i,j,k,l} \in [0,1]\}$$

where $IET_i^{(t)}$ is the trust experience of agent a_i at time t, A is the set of agents in the system, S is the set of possible specifications of tasks that the agent needs to contract, $et_{i,j,k,l}$ is the satisfaction degree of agent a_i when agent a_j offers a solution to the task s_k for the l-th time.

Also, the initiator agent must store data about the reliability of other agents when they offer reputation information (Initiator's Experience of Reputation):

$$IER_i^{(t)} = \{(a_j, s_k, er_{i,j,k}) | a_j \in A, s_k \in S, er_{i,j,k} \in [0,1]\}$$

where $er_{i,j,k}$ is the satisfaction degree of agent a_i when a_j offers reputation values about other agents when they performed the task s_k.

In order to update the bases of experiences, at the end of each interaction t, the agent a_i evaluates the interaction, taking into account the solution w_j as response of the task s_k. The information about each particular interaction, that agent a_i carried out for a given task s_k, may be grouped in the set:

$$I^{(t)}(a_i, s_k) = \{(a_j, w_j) | a_j \in C^{(t)}(a_i, s_k),\ w_j \in W\},$$

where w_j is the response given in this interaction by agent a_j; W is the set of all possible responses, and $C^{(t)}(a_i, s_k)$ is the set of the most reliable agents to give solutions to task s_k according to the experience of agent a_i.

Updating process combines the interaction results $I^{(t)}(a_i, s_k)$ and stored experiences in $IET_i^{(t)}$ and $IER_i^{(t)}$ using quality and satisfaction measurements.

4.1 Arranging Agents for Asking Them About Trust and Reputation

At the beginning of each interaction, initiator agent needs to identify the more reliable partners for the required task in order to interacts with them depending on its previous behaviour. For each task s_k, it can create two neighbors lists according to the partner trust degree to give a response for the required task:

- Set of the most reliable agents to give a response ($CT_{sup}^{(t)}(a_i, s_k)$)
- Set of agents with a doubtful confidence to give a response ($CT_{dud}^{(t)}(a_i, s_k)$)

In the same way, we may stand out the group $CR_{sup}^{(t)}(a_i, s_k)$ for the most reliable agents giving reputation values, according to confidence to give reputation values for the specific task s_k.

Agents from $CT_{sup}^{(t)}(a_i, s_k)$ should be asked from responses for task s_k, because they are the most reliable agents. However, it is possible that some agents from $CT_{dud}^{(t)}(a_i, s_k)$ could come up with valuable information. The system is dynamic and this allow the system to adapt. It would be desirable that these agents were also asked for a response about s_k. We define a set of agents with a doubtful trust to give response, but with a high reputation:

$$C_{prom}^{(t)}(a_i, s_k) = \{a_j |\ a_j \in CT_{dud}^{(t)}(a_i, s_k), R(a_i, a_j, s_k, CR_{sup}^{(t)}(a_i, s_k)) \geq \gamma_{sup}\}$$

where the function $R(a_i, a_j, s_k, CR_{sup}^{(t)}(a_i, s_k))$ represents the reputation value assigned by a_i to agent a_j for task s_k according to experience of the most reliable agent given reputation information, and γ_{sup} is a threshold value.

Hence, we define

$$C^{(t)}(a_i, s_k) = CT_{sup}^{(t)}(a_i, s_k) \cup C_{prom}^{(t)}(a_i, s_k)$$

as the set of requested agents that agent a_i will ask for task s_k. This list is populated by agents with a high trust value to give response and others with doubtful trust value to give response but with a high reputation value according to the most reliable agents given reputation information.

4.2 Obtaining Trust and Reputation

The concept of trust as used in this model not only takes into account the partner in a given negotiation, but the trust value associated to the given task specification. Also, it combines direct trust experience with opinions of high-trusted neighbors. The global value of trust $f_{i,j,k}^{(t)}$ is obtained from the bases of experiences like in REGRET [10], using this function:

$$f_{i,j,k}^{(t)} \equiv T(a_i, a_j, s_k) = DTRL(a_i, a_j, s_k, IET_i^{(t)})\ DT(a_i, a_j, s_k, IET_i^{(t)}) +$$

$$(1 - DTRL(a_i, a_j, s_k, IET_i^{(t)}))R(a_i, a_j, s_k, CR_{sup}^{(t)}(a_i, s_k))$$

where $DT(a_i, a_j, s_k, IET_i^{(t)})$ represents the direct trust value that agent a_i assigns to agent a_j for task s_k according to the experience in his own base $IET_i^{(t)}$; $R(a_i, a_j, s_k, CR_{sup}^{(t)}(a_i, s_k))$ is the reputation value that agent a_i assigns to agent a_j for task s_k according to the experiences of the most reliable agents to give reputation.

Direct trust $DT(a_i, a_j, s_k, IET_i^{(t)})$ and its reliability $DTRL(a_i, a_j, s_k, IET_i^{(t)})$ are obtained using functions that query the base of experiences $IET_i^{(t)}$. Our model uses a discount approach taking into account that experiences lose relevance as they get older. The way to model the lose of relevance of experience is carried out different than REGRET. If $0 \leq \delta \leq 1$ is a time modulating parameter, that gives higher importance to experiences closer to t, trust can be calculated as follows:

$$DT(a_i, a_j, s_k, IET_i^{(t)}) = \sum_{l_p \in L} \delta^{|L|-p} et_{i,j,k,p}$$

where L is subset of different experiences that agent a_i has about the performance of agent a_j associated to task s_k ($L \subset IET_i^{(t)}$, $|L| \leq t$). Subindex p, in the new set L, indicates how old is the experience $et_{i,j,k,p}$: l_{p_2} is more recent that experience l_{p_1} only if $p_2 > p_1$. The $et_{i,j,k,0}$ represents the oldest experience that agent a_i has about the performance of agent a_j for task s_k.

To model how reliable the direct trust measure is, we follow the models given by SPORAS [18] and REGRET [10]. Reliability value is obtained from the amount and variability of experiences used to calculated the trust:

$$DTRL(a_i, a_j, s_k, IET_i^{(t)}) = N_o(a_i, a_j, s_k, IET_i^{(t)}) \cdot (1 - D_v(a_i, a_j, s_k, IET_i^{(t)}))$$

where

$$N_o(a_i, a_j, s_k, IET_i^{(t)}) = \begin{cases} sin(\frac{\pi \cdot |L|}{2 \cdot itm}) & : \quad |L| \leq itm \\ 1 & : \quad \text{otherwise} \end{cases}$$

and

$$D_v(a_i, a_j, s_k, IET_i^{(t)}) = \sum_{l_p \in L} \delta^{|L|-p}(|et_{i,j,k,p} - DT(a_i, a_j, s_k, IET_i^{(t)})|)$$

where itm is a domain-dependent parameter to control the maximum number of experiences taken into account to improve the reliability on the trust measurement. Values greater than itm do not improve the reliability of the metric.

The deviation of the experiences from direct trust ($D_v(a_i, a_j, s_k, IET_i^{(t)})$) is obtained following the same method to calculate direct trust. Differences between experience value and direct trust loses relevance thorugh time.

In other hand, $R(a_i, a_j, s_k, CR_{sup}^{(t)}(a_i, s_k))$ represents the reputation value that agent a_i assigns to agent a_j according to the experiences of the most reliable agents giving reputation information for task s_k.

Taking into account some trust and reputation models, given by Golbeck and Hedler [3,2], Zacharia [18] and Schillo [11], and considering that the reputation is a task-associated value, we propose a reputation function based on the propagation of reputation information from the most reliable agents:

$$R(a_i, a_j, s_k, CR_{sup}^{(t)}(a_i, s_k)) = \frac{\sum_{a_q \in CR_{sup}^{(t)}(a_i, s_k)} DT(a_q, a_j, s_k, IET_q^{(t)}) \cdot er_{i,q,k}^{(t)}}{\sum_{a_q \in CR_{sup}^{(t)}(a_i, s_k)} er_{i,q,k}^{(t)}}$$

where agent a_i, interested in obtaining reputation information, requests information to the reliable agents a_q to give reputation information for task s_k (grouped in $CR_{sup}^{(t)}(a_i, s_k)$) about trust on a_j.

The trust value to give reputation information $er_{i,q,k}^{(t)}$ is obtained directly from the base of experiences $IER_i^{(t)}$. The agent a_i stores, for each agent a_q, a unique trust value to give reputation information $er_{i,q,k}^{(t)}$, for each task s_k. This value is updated after each interaction is finalized (please, see section 4.4).

4.3 Obtaining Trust and Reputation from Similar Tasks

It is possible that an agent does not have information about performance of other agents for a given task. In this case, it needs to approximate the trust and reputation values using a similar task whose accomplishment has been previously done by known agents and requested by a_i. The model may obtain this approximation using a similarity degree between the most similar well-known task and the unknown one. In our model, the similarity between two tasks s_k and s_p is obtained from the comparison of the task attributes. This is a domain-dependent function.

This way, combining the trust or reputation in the most similar task (s_p) with the similarity degree between the two tasks $D(s_k, s_p)$, we define indirect trust (IT) or indirect reputation (IR) functions to approximate direct trust or reputation values, respectively:

$$IT(a_i, a_j, s_k, s_p, IET_i^{(t)})) = DT(a_i, a_j, s_p, IET_i^{(t)}) \cdot D(s_k, s_p),$$

$$IR(a_i, a_j, s_k, s_p, CR_{sup}^{(t)}(a_i, s_k)) = R(a_i, a_j, s_k, CR_{sup}^{(t)}(a_i, s_k)) \cdot D(s_k, s_p)$$

According to Rodriguez and Egenhofer [9] the similarity can be calculated using elements from the set theory and Tversky's measure [12] as indicator of the semantic similarity between entities described using the same ontologies, in this case, between two tasks described using WSMO.

Tversky [12] defines a similarity measure in terms of a matching process. This measure produces a similarity value that is not only the result of the common, but also the result of the different characteristics between objects.

In our model, similarity between tasks ($D(s_q, s_p)$) is a domain - dependent concept that takes into account the Tversky's measure and set theory [9,12] over WSMO [14]. According to the Tversky's model, the similarity between two concepts a and b can be determined in the following way:

$$D(a, b) = \frac{|A \cap B|}{|A \cap B| + \alpha(a, b)|A \setminus B| + (1 - \alpha(a, b))|B \setminus A|}$$

where $0 < \alpha < 1$; A and B are the set of properties of concepts a and b, respectively.

$D(a, b)$ is not necessarily symmetrical, unless a and b are equal or $\alpha(a, b) = (1 - \alpha(a, b))$, that is to say, $\alpha(a, b) = 0.5$. Rodriguez and Egenhofer [9] define the function α taking into account the depth of compared concepts in the ontology hierarchy. Using the same expression to obtain α, and comparing the same concept of the same ontology (equal depth for each task) we take that $\alpha = 0.5$ (symmetrical similarity measure $D(a, b) = D(b, a)$).

4.4 Updating the Bases of Experiences IET and IER

At the end of each interaction, the model must update the two bases of experiences with the information generated in that time step.

To update the base of experiences $IET_i^{(t)}$, for each agent a_j, that gives the solution w_j to the task s_k, the model can generate the new experience:

$$edt_{i,j,k} = (a_i, a_j, s_k, et_{i,j,k})$$

where

$$(a_j, w_j) \in I^{(t)}(a_i, s_k)$$

where the trust value $et_{i,j,k}$ is a measure obtained from the real quality of the solution (Q) and the fulfillment (P) of the promised satisfaction ($ec_{i,j,k}$):

$$et_{i,j,k} = Q(w_j, s_k) \cdot P(ec_{i,j,k}, Q(w_j, s_k))$$

This way, the proposed model avoids that an agent a_j, with a low promised satisfaction $ec_{i,j,k}$ and a medium-quality solution for task s_k, may obtain a high satisfaction degree $et_{i,j,k}$. The satisfaction degree must be the combination of real quality of the solution (Q) and the fulfillment of the promised quality (P). The definitions of functions P and Q are given in section 4.5.

Here, the model takes into account that is possible to add the new experience $edt_{i,j,k}$, without having to analyze how many experiences are in the base $IET_i^{(t)}$.

The base of experiences for reputation $IER_i^{(t)}$ has an unique value of reputation $er_{i,j,k}^{(t)}$ to indicate, according to experience of agent a_i, the reliability of agent a_j to give reputation information about other agents performing task s_k.

When agent a_j was requested by agent a_i about agents from $CT_{dud}^{(t)}(a_i, s_k)$, it may recommend some of them given their high reputation according to its experience. The recommended agents by a_j, to agent a_i for task s_k, can be grouped under the set:

$$M_j^{(t)}(a_i, s_k) = \{a_r | a_r \in CT_{dud}^{(t)}(a_i, s_k), f_{j,r,k}^{(t)} \geq \gamma_{sup}\},$$

where γ_{sup} is a thershold value.

This way, agent a_i must adjust $er_{i,j,k}^{(t)}$ reputation value on the requested agent a_j, taking into account the variation (produced during the interaction) on trust about recommended agents from $M_j^{(t)}(a_i, s_k)$.

For each requested agent $a_j \in CR_{sup}^{(t)}(a_i, s_k)$, for current task s_k, the model analyzes the cases of each agent $a_r \in M_j^{(t)}(a_i, s_k)$, taking into account the trust value that agent a_i had about a_r (denoted by $f_{i,r,k}^{(t)}$) at the beginning of the interaction and the new value (denoted by $f_{i,r,k}^{(t+1)}$) at the end.

The trust value to give reputation information $er_{i,j,k}^{(t)}$ is modified combining the mean of all differences between final and previous trust values for each agent a_r, about agent a_j. The value of the reputation $er_{i,j,k}^{(t+1)}$ will be better than $er_{i,j,k}^{(t)}$ when the trust on recommended agents from $M_j^{(t)}(a_i, s_k)$ is improved during the interaction:

$$er_{i,j,k}^{(t+1)} = sigmod(er_{i,j,k}^{(t)} + \frac{\sum_{a_r \in M_j^{(t)}(a_i,s_k)} f_{i,r,k}^{(t+1)} - f_{i,r,k}^{(t)}}{|M_j^{(t)}(a_i, s_k)|})$$

where

$$sigmod(x) = \frac{1}{1 + e^{-\rho(x-\frac{1}{2})}}$$

The parameter ρ controls the squared-like shape of the function (the higher the ρ value, the faster the function gets to its maximum).

4.5 Satisfaction: Fulfillment and Quality

As we treated in the previous section, our model needs two functions to evaluate the satisfaction of the initiator agent through the fulfillment of the promised satisfaction degree and the quality of the solution according to the task.

The fulfillment of the promised satisfaction indicates to what extent, the responder agent fulfills the promised quality $ec_{i,j,k}$. We understand this function as a comparison between the agreement quality value $ec_{i,j,k}$ and the real quality of the given solution, denoted by $Q(w_j, s_k)$. To determine the fulfillment of the satisfaction agreement, we may define a function P:

$$P(ec_{i,j,k}, Q(w_j, s_k)) = \begin{cases} 1 & : \quad Q(w_j, s_k) \geq ec_{i,j,k} \\ 1 - (ec_{i,j,k} - Q(w_j, s_k)) & : \quad Q(w_j, s_k) < ec_{i,j,k} \end{cases}$$

comparing the promised quality value ($ec_{i,j,k}$) with the quality of the solution w_j for task s_k. If the real satisfaction degree overcomes the promised value, the function returns 1, otherwise it is an indicator of the difference between values.

The quality of the solution, $Q(w_j, s_k)$, indicates how much the response w_j satisfies the requirements specified in the task s_k. Calculation of this value is based on the comparison of both concepts, it is a domain-dependent function.

To obtain the value of satisfaction degree, our model proposes to consider the Web service discovery process in WSMO [15]. In this case, tasks are represented by goals and responses by Web services descriptions, discovery process given by WSMO acts as a function that indicates the matching degree of the Web service (response w_j) and the desired goal (task s_k). Section 5 shows an example of the definition of the quality function using WSMO discovery process based on simple semantic descriptions of services.

5 How to Compute Satisfaction and Similarity

It is possible to apply this model into a simple provider - consumer P2P scenario. This way, we try to ilustrate how we can use this model in an scenario where a consumer agent requests tasks and obtains solutions from providers.

Each task request s_k or response w_j (represented by Goal or WebService, respectively) is described by the set of non-functional properties listed in section 3 (i.e. accuracy, performance, reliability, etc.). Also, according to this application domain, we may add two properties: speed, representing the download speed, and quality, representing the quality of downloaded resource.

For each property of Goal or WebService, the model must define a normalization function to make independent the domain of the real world values from

model-managed values. For that, the model uses values in the range [0,1] to represent the convenience of the property, independent of the original property domain: a value near to 0 indicates a non-desired value in the original property, and values near to 1 indicate high-desired values in the original properties. For instance, when download speed is very fast, the value of the property "speed" is near to 1, but when the number of errors generated in a certain time interval is high, the value of the attribute "accuracy" is near to 0 (please, see section 3).

In WSMO, the Web Service discovery process using simple semantic descriptions of services is based on set theory and exploits ontologies as formal, machine-processable representation of domain knowledge [14,15].

The set of elements of Goals and WebServices can be analyzed in different ways, given a non-unique semantic interpretation. For instance, using the same set of elements to describe a Goal, we can specify that the user wants to satisfy all properties or only some of them. The same situation occurs with WebServices concept. For this reason, it is necessary to specify the intention (universal or existential) of the description of Goal or WebService, in order to determine the type of coincidence between Goal and WebService in the discovery process. For instance, if the user wants to satisfy all request attributes, the intention of the goal is universal; in other hand, if the purpose is to satisfy only some of them, the intention is existential.

Following the discovery approach based on the simple description of Web services [15], for each goal (s_k) or Web service (w_j), we need to group the good-value attributes in the sets R_g and R_w, respectively.

R_g and R_w consist of the most prominent attributes for each concept, according to the value of each attribute. To construct these sets, we consider that the attribute b_i of s_k is a good-value attribute and hence $b_i \in R_g$ if $s_k.b_i \geq \lambda_i$ (λ_i is a domain-dependent threshold value). In the same way, an attribute b_i of w_j is a good-value and $b_i \in R_w$ if $w_j.b_i \geq \lambda_i$.

Considering universal intentions for goals and Web services $I_g = I_{s_k} = \forall$ and $I_w = I_{w_j} = \forall$ over the sets R_g and R_w (that contain good-values attributes of s_k and w_j, respectively), we may define the value of satisfaction degree:

$$Q(w_j, s_k) = \begin{cases} 1 & : \ R_g = R_w & Match \\ 0.75 & : \ R_g \subseteq R_w & Match \\ 0.5 & : \ R_g \supseteq R_w & PartialMatch \\ 0.5 & : \ R_g \cap R_w \neq \emptyset & PartialMatch \\ 0 & : \ R_g \cap R_w = \emptyset & NoMatch \end{cases}$$

According to this definition, maximum satisfaction degree is obtained when all important (good-value) attributes desired in goal s_k are important (good-value) attributes in Web services w_j. Contrary, the worst satisfaction is obtained when no prominent attributes of goal s_k are satisfied by important attributes of Web services w_j. Also, the satisfaction function considers intermediate cases.

To determine the similarity between two tasks s_q and s_p, using the definition of D defined in section 4.3, we consider the sets R_{gq} and R_{gp} of prominent non-functional and domain - dependent properties of tasks s_q and s_p, respectively:

$$D(s_q, s_p) = \frac{|R_{gq} \cap R_{gp}|}{|R_{gq} \cap R_{gp}| + 0.5|R_{gq} \setminus R_{gp}| + 0.5|R_{gp} \setminus R_{gq}|}$$

This way, we have a general method to obtain two needed domain-dependent measures in the proposed trust and reputation model: task satisfaction given a response and similarity between two tasks. It offers a very simple definition based on the set theory and WSMO elements.

When linking trust and reputation model to WSMO, the satisfaction and similarity measures use the concepts of Goals and WebServices in the definition of the tasks (the users' requirements) and the answers (services that satisfy the requirements), respectively. However, it can consider other domain-dependent characteristics like in this example: download speed and download quality. For this reason, the model can be adapted to different application domains where WSMO is the used ontology framework.

6 Conclusions and Future Work

This paper proposes a model to manage trust and reputation taking WSMO as conceptual framework in a P2P environment, where agents should be able to contract the Web service of best behavior. The combination of the trust model with the ontological representation offered by WSMO allows the service discovery process to take advantage of the previous knowledge of the system, taking into account the satisfaction degree of the previous tasks.

It is considered that the trust and the reputation in each agent can be different in dependence of the specified task or requirement. Nevertheless, if the model ignores the behavior of the service for a given task, the values of trust can be approximated using the similarity between this and a well known task.

For the description of the services and their requests, the model suggests the concepts given by WSMO: Web Service and Goal. This way, it facilitates the definition of some characteristics and functions that are dependent of the application domain, such as the satisfaction of a task given the answer or the similarity between two tasks.

We intend to implement and prove the trust and reputation model based on WSMO, comparing different configurations, trying to affirm experimentally that the quality of discovery process in WSMO is improved when the trust and reputation model is used. We will identify the parameters that affect the system performance and their high-recommended values.

Now, the model is being adapted to ART [1] trying to prove its operation in front of other models of trust and reputation, evaluating and adjusting its capacities of reactivity and representation of the behavior of other agents. We identify some common characteristics and match some related concepts between this model and ART. However, there are several concepts difficult to match, that require ingenious and hard work. Also, we expect that the initial partition of neighbors, proposed by our model, enhances the agents' profits because it reduces unnecessary message interchanging, taking into account that in ART each opinion request has associated a given cost.

References

1. K. Fullam, T. Klos, G. Muller, J. Sabater, Z. Topol, K. S. Barber, J. S. Rosenschein, and L. Vercouter. The Agent Reputation and Trust (ART) Testbed Architecture. In *Proc. of Trust Workshop at AAMAS*, 2005.
2. J. Golbeck and J. Hendler. Accuracy of metrics for inferring trust and reputation in semantic web-based social networks. In *Proc. of EKAW'04*, 2004.
3. J. Golbeck and J. Hendler. Filmtrust: Movie recommendations using trust in web-based social networks. In *Proc. of the IEEE Consumer Communications and Networking Conference*, January 2005.
4. N. Griffiths. Task delegation using experience-based multi-dimensional trust. In *Proc. of Trust Workshop at AAMAS*, 2005.
5. S. Marti. *Trust and Reputation in Peer-to-Peer Networks*. PhD thesis, Stanford University, 2005.
6. D. Milojicic, V. Kalogeraki, and R. L. R. Peer-to-peer computing. Tech report: Hpl-2002-57, Hewlett Packard, 2002.
7. M. Montaner, B. Lopez, and J. L. de la Rosa. Developing trust in recommender agents. In *Proc. of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*. C. Castelfranchi and L. Johnson (eds), 2002.
8. S. Ramchurn, D. Huynh, and N. Jennings. Trust in multi-agent systems. *Knowledge Engineering Review*, 1(19):1–25, 2004.
9. M. A. Rodriguez and M. J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Transactions on knowledge and Data Engineering*, 15(2):442–456, 2003.
10. J. Sabater and C. Sierra. Regret: a reputation model for gregarious societies. In *Proc. of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 475–482. C. Castelfranchi and L. Johnson (eds), 2002.
11. M. Schillo, P. Funk, and M. Rovatsos. Using trust for detecting deceptive agents in artificial societies. *Applied Artificial Intelligence, Special Issue on Trust, Deception, and Fraud in Agent Societies*, 14:825–848, 2000.
12. A. Tversky. Features of similarity. *Psychological review*, 84(4):327–352, 1977.
13. Y. Wang and J. Vassileva. Trust and reputation model in peer-to-peer networks. In *Proc. of IEEE Conference on P2P Computing*. Sweden, September 2003.
14. WSMO Team. *Web Service Modeling Ontology (WSMO)*. W3C, http://www.w3.org/Submission/WSMO/, 2005.
15. WSMO Team. *WSMO Web Service Discovery. WSML Working Draft*. W3C, http://www.wsmo.org/2004/d5/d5.1/v0.1/, 2005.
16. T. Yamagishi, K. Cook, and M. Watabe. Uncertainty, trust, and commitment formation in the united states and japan. *American Journal of Sociology*, 104(1) 165–194, 1998.
17. B. Yu and M. Singh. Distributed reputation management for electronic commerce. *Computational Intelligence*, 18(4):535–549, 2002.
18. G. Zacharia. *Collaborative reputation mechanisms for communities*. PhD thesis, Massachusetts Institute of Technology, 1999.

Author Index