Proceedings of The Third International Workshop on

INFRASTRUCTURES AND TOOLS FOR MULTIAGENT SYSTEMS

ITMAS 2012

June 5, 2012
Valencia, Spain

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

EDITORIAL

First edition 2012

ISBN: 978-84-8363-850-7
Publisher reference number: 2701

Printed in Spain

Workshop Organization

Programme Chairs

Juan M. Alberola (Universitat Politècnica de València, Spain)
Vicent Botti (Universitat Politècnica de València, Spain)
Ana Garcia-Fornes (Universitat Politècnica de València, Spain)
Michal Pěchouček (Czech Technical University in Prague, Czech Republic)
Alessandro Ricci (Alma Mater Studiorum-Università di Bologna, Italy)
Jose M. Such (Universitat Politècnica de València, Spain)
Danny Weyns (Katholieke Universiteit Leuven, Belgium)

Programme Committee

Juan M. Alberola (Universitat Politècnica de València, Spain)
Makoto Amamiya (Kyushu University, Japan)
Matteo Baldoni (Università degli Studi di Torino, Italy)
Fabio Bellifemine (Telecom Italia, Italy)
Juan A. Botía (University of Murcia, Spain)
Vicent Botti (Universitat Politècnica de València, Spain)
Juan M. Corchado (University of Salamanca, Spain)
Yves Demazeau (Laboratoire d'Informatique de Grenoble, France)
Nadia Erdogan (İstanbul Teknik Üniversitesi, Turkey)
Agustin Espinosa (Universitat Politècnica de València, Spain)
Ana Garcia-Fornes (Universitat Politècnica de València, Spain)
Dominic Greenwood (Whitestein Technologies, Switzerland)
Jomi F. Hübner (Federal University of Santa Catarina, Brazil)
Kamalakar Karlapalem (Int. Institute of Information Technology, India)
Michal Laclavik (Slovak Academy of Sciences, Slovak Republic)
Tim Miller (University of Melbourne, Australia)
Pavlos Moraitis (Paris Descartes University, France)
Andrea Omicini (Alma Mater Studiorum-Università di Bologna, Italy)
Michal Pechoucek (Agent Technology Center, Czech Republic)
Martin Purvis (University of Otago, New Zealand)
Alessandro Ricci (Alma Mater Studiorum-Università di Bologna, Italy)
Juan A. Rodriguez-Aguilar (IIIA-CSIC, Spain)
Michael Rovatsos (The University of Edinburgh, UK)
Murat Sensoy (University of Aberdeen, UK)
Michael Shumacher (University of Applied Sciences Western, Switzerland)
Carles Sierra (IIIA-CSIC, Spain)
Jose M. Such (Universitat Politècnica de València, Spain)
Pavel Vrba (Rockwell Automation Research Center, Czech Republic)
Danny Weyns (Katholieke Universiteit Leuven, Belgium)

External Reviewers

Manuel Atencia (IIIA-CSIC, Spain)
Christopher Frantz (University of Otago, New Zealand)

Preface

ITMAS 2012 follows the success of its predecessors ITMAS 2010 and ITMAS 2011. ITMAS 2010 and ITMAS 2011 were held in conjunction with AAMAS 2010 in Toronto (Canada) and Taipei (Taiwan) respectively. ITMAS 2012 is to be held standalone this time in Valencia (Spain). This year we had 17 submissions from which 13 were finally accepted. This clearly confirms the growing interest in the workshop as well as its relevance for the whole comunity working on infrastructures and tools for Multi-agent Systems.

ITMAS aims at bringing together leading researchers from both academia and industry to discuss issues on the design and implementation of infrastructures and tools for Multiagent Systems. When developing applications based on Multiagent Systems, developers and users demand infrastructures and tools which support essential features in Multiagent Systems (such as agent organizations, mobility, etc.) and facilitate the system design, management, execution and evaluation. Agent infrastructures are usually built using other technologies such as grid systems, service-oriented architectures, P2P networks, etc. In this sense, the integration and interoperability of such technologies in Multiagent Systems is also a challenging issue in the area of both tools and infrastructures for Multiagent Systems. A long term goal is the industrial development of infrastructures for building highly scalable applications comprising pre-existing agents that must be organized or orchestrated.

In order for Multiagent Systems to be included in real domains such as media and Internet, logistics, e-commerce and health care, infrastructures and tools for Multiagent Systems should provide efficiency, scalability, security, management, monitorization and other features related to building real applications.

Alessandro, Anna, Danny, Jose, Juanmi, Michal, and Vicent.

Table of Contents

A Dynamic Account Payment Method for Integrating Heterogeneous B2C Electronic Payment Systems

Jingzhi Guo and Chong-Wan Wong
Department of Computer and Information Science, University of Macau, Taipa, Macau
{jzguo, ma56565}@umac.mo

Abstract

While the supported e-payment methods of online shops are different from the applied e-payment methods of online consumers, a B2C e-payment mismatch problem occurs and severely affects the online sales. This paper proposes a new dynamic account payment (DAP) method to resolve the mismatch problem. The main idea of DAP is that consumer can request a temporary e-payment account, acceptable to the online shop s/he is shopping, from a third-party which has pools of accounts in most e-payment systems. In this paper, DAP method is technically designed in a framework of rules, events and agents, and is implemented in two alternative approaches. Experiments have been made for evaluating the performance of the two approaches. The experiment result signifies that DAP method is feasible and efficient.

Keywords: e-payment mismatch problem, e-payment, electronic payment, B2C, business-to-consumer, electronic commerce

1 Introduction

Electronic payment (e-payment) is now popularly adopted by both online shops and their customers ([5], [7], [12], [17], [18]). An online shop may often support one or more e-payment systems and an online consumer may also have one or several e-payment means. Nevertheless, we have observed that when an online consumer makes an e-payment in an online shop, his/her available e-payment means may not be supported by the e-payment systems that the online shop adopts. For example, when an online shop accepts payments in PayPal (paypal.com) and Visa credit card and a consumer only has the payment means of AliPay (alipay.com), the consumer is unable to make payment for a successful transaction. This is what we called a business-to-consumer (B2C) e-payment system mismatch problem and typically leads to an unsuccessful online transaction between online shop and online consumer.

Resolving B2C e-payment system mismatch problem is important since it can increase the confidence of consumers of online shopping and enlarge the volume of online sales. Existing methods of resolving this problem can be categorized into e-payment standards ([15], [2]) and e-payment system vendor-based integration (e.g. AliPay and YeePay in Table 1). An e-payment standard often defines a set of standard payment message formats, processes and commands for all adopters to follow. This approach is limited to only those online shops, which share a set of same standards. Differently, the vendor-based integration approach is often applied by specialized e-payment service providers, which integrate as many as the existing known payment means and make close cooperation with merchants and banks to clear and settle payments. This approach is often exclusive, that is, excluding

the other providers' e-payment systems in integration. This exclusive manner of integration can be illustrated in Table 1 in a comparison between e-payment service providers of PayPal (paypal.com), AliPay (alipay.com) and YeePay (yeepay.com), which are all large in service size.

Table 1: Comparison of Payment Methods between AliPay, PayPal and YeePay

AliPay	Mobile phone	Online banks	Payment cards	AliPay	Credit card	Escrow service	COD
PayPal				PayPal	Credit card		
YeePay	Mobile phone	Online banks	Payment cards	YeePay	Credit card	Phone payment	Debit payment

Source: The information is summarized from alipay.com, paypal.com and yeepay.com.

Table 1 shows that all AliPay, PayPal and YeePay do not accept payments from each other but they accept payments from other means such as online banks and various card payments. It implies that existing e-payment systems have vertical e-payment integration into existing online banks and card payments. It does not, however, have horizontal e-payment integration into other competitive e-payment systems. This phenomenon might be explained as a result of heated competition between existing e-payment systems [15]. Nevertheless, such exclusive integration solution provided by single e-payment service providers makes e-payment mismatch problem more severe, and it is thus worthwhile for us to make a further research.

To relieve the problem caused by the B2C e-payment mismatch, this paper proposes a new B2C e-payment method, which is called Dynamic Account Payment (DAP). The main idea is: we design and implement an integrated e-payment portal, which registers several e-payment accounts as an account pool from each existing e-payment system. A user-side agent is installed in the consumer's computer or shop's merchant server. It asks for a temporary account from the account pool for each payment transaction. This temporary account is time-constrained with user-proposed spending limit such that the temporary account will not be abused. By this new B2C e-payment method, we solve the B2C e-payment mismatch problem.

The remainder of the paper will be organized as follows: Section 2 discusses some related work relevant to the DAP method. Section 3 describes the design of DAP method. Section 4 provides two alternative implementations for the DAP method. Section 5 evaluates the two alternative implementations. Finally, a conclusion is made with listing contributions and future work.

2 Related Work

The design of DAP e-payment method relates to three kinds of technologies: agent systems, business rules and event messaging.

2.1 Agent-Based Systems

In most multi-agent systems ([21], [13]), a middle agent usually acts as both a service provider when it perceives a request and a service requester when it needs a service. It is capable of "thinking" and produces intelligent feedback. Multi-agent systems are often used as the infrastructure for data aggregation and communication [19] for flexible and distributed control and automation (e.g. [20], [22]). In e-payment system design, multi-agent systems are widely adopted to construct e-payment systems, for example,

SAFER [9]. SAFER enables multiple SAFER communities to be distributedly worked in a modularized pattern, where payment agents could be created, evolved and roamed. SAFER is distributed, flexible to accommodate multiple e-payment schemes to supports B2C payments.

2.2 Rule-Based Modeling

Rule-based systems [1] are the systems that use stored knowledge patterns to dynamically interpret information. They usually could model human behavior based on "rules that stipulate that a certain action be initiated should a specific set of conditions occur" [3]. The modeled systems are not rigorously designed but could be flexible by following rules that may be edited. Rules are often used to trigger events for message input and dispatch. For example, we can use a set of rules to modify the system behavior through a dispatching service. In e-payment systems, rules could be used to design payment policies based on a rule specification, for example, DMDCON multi-policies [14] or SAFER ruler on priority, factor, condition and payment method name [9].

2.3 Event-Driven Architecture

Event-driven architecture (EDA) ([10], [11], [8]) is a software architecture pattern that is event-based for message production, detection, consumption, and reaction. It can simplify the systemic and uncertain input format. In this architecture, an event is a significant change in state. For example, when an e-payment button is clicked (i.e. an event), the required e-payment means is required for completing the deal. The idea of EDA can be applied to build services distributed on Internet, which is light-weighted in a plug-and-play fashion, for example, JEDI [6]. An event-driven design could represent more dynamic system architectures when using an event-based architecture definition language (e.g. [16]). In heterogeneous e-payment systems integration, the event-driven concept can be applied to build distributed e-payment services, which are autonomously developed in various locations.

3 Dynamic Account Payment Method

In this section, we describe our proposal of developing a new e-payment method, named Dynamic Account Payment (DAP) to solve the e-payment mismatch problem.

3.1 An Architectural View of DAP Method

The design of DAP method applies the concepts of agents, rules and events. Particularly, rules are used to build payment logic, for example, the payment protocol between consumer and DAP module. Events are triggered messages for payment actions. Agents are software modules that execute payment actions on the events. Based on the three concepts, the DAP method is designed in an architecture as described in Fig. 1, which consists of four roles - existing e-payment systems (EPS), DAP e-payment portal (DAP Portal), online shops, and consumers.

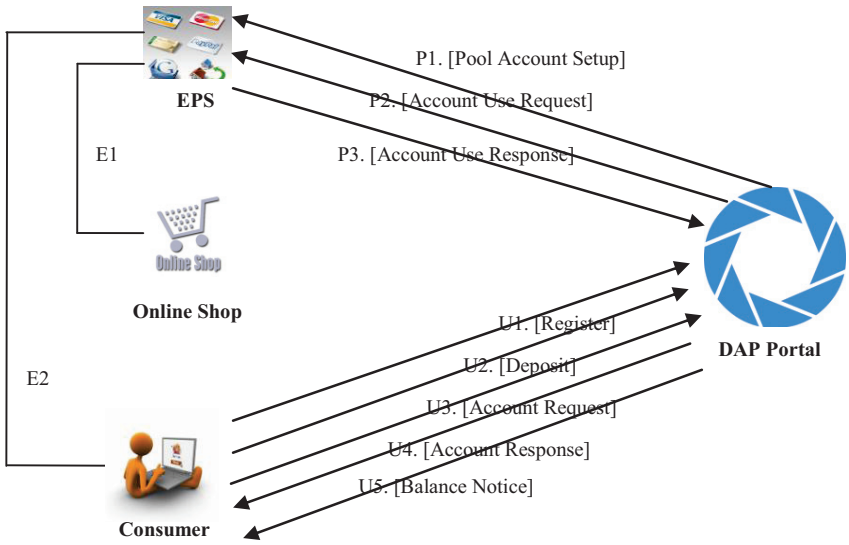


Figure 1: Architecture of DAP System Design

In Figure 1, E1 and E2 are the existing e-payment relations, where E1 denotes that an online shop accepts an e-payment mean E1 of EPS and E2 refers to that consumer has an e-payment mean E2 of EPS. Obviously, E1 and E2 are heterogeneous and are the cause of the e-payment mismatch problem. In the architecture, the DAP method introduces a third-party system, called DAP Portal, to mediate heterogeneous e-payments between Consumer and EPS to solve the e-payment mismatch problem. The process of mediation can be described in two aspects as follows.

The first aspect states that a designed DAP Portal must establish a long-term cooperative relationship with every EPS. Such cooperation could be strategically made as a part of value chain formation (Chen and He, 2009). Particular processes involved in the cooperation are:

- P1 (Pool Account Setup). DAP Portal request every EPS to setup a pool of consumer e-payment accounts for payment by anyone. Each account in the pool has attributes of start use time, end use time, and use limit. This means that each account can be used as a time-constrained temporary account for any consumer, but it is only valid between start time and end time with a spending limit.
- P2 (Account Use Request). For every request of using an account from the account pool, DAP Portal sends an account use request to an EPS with the information of allowed time duration and spending limit by transferring money from its central account in that EPS (or direct transfer) to the requested account. The EPS then immediately sets the start use time and end use time, and allows a payment from this temporary account.
- P3 (Account Use Response). EPS responds to DAP Portal after the allowed time duration is finished, telling whether the limited money has be spent or not. This allows DAP Portal to trace the consumer's payment status and determine how much it should actually deduct from the consumer's deposit account in DAP Portal.

The Second aspect states that a consumer must register a deposit account in DAP portal for

temporarily using a DAP pool account for a target EPS payment. The processes, enabling a consumer to avoid the e-payment mismatch problem, are:

- U1 (Register). A consumer register a DAP deposit account to obtain a DAP-unique account number.
- U2 (Deposit). A consumer deposits some money in DAP Portal for future use.
- U3 (Account Request). When a consumer meets an e-payment mismatch, it requests a temporary account of the target EPS from DAP portal by telling the needed payment amount.
- U4 (Account Response). When DAP Portal has received the account request from the consumer, it checks whether the consumer has enough fund in DAP deposit account. If it has enough fund, it sends a temporary e-payment account of the target EPS with spending limit and allowed payment time.
- U5 (Balance Notice). When the allowed e-payment time is passed, DAP Portal reports the new balance to the consumer.

The above two aspects combined together show a high-level architectural design of DAP method, which integrates heterogeneous B2C e-payment systems to avoid B2C e-payment mismatch problem.

3.2 REA Framework

DAP method is technically designed in a framework of rules, events and agents (REA), shown in Figure 2, based on the above-mentioned DAP architecture. It is effective for separating graphic user interfaces from message event and event handling. It is beneficial to seamlessly integrate the distributed roles of EPS, online shoppers and consumers in a module-independent manner.

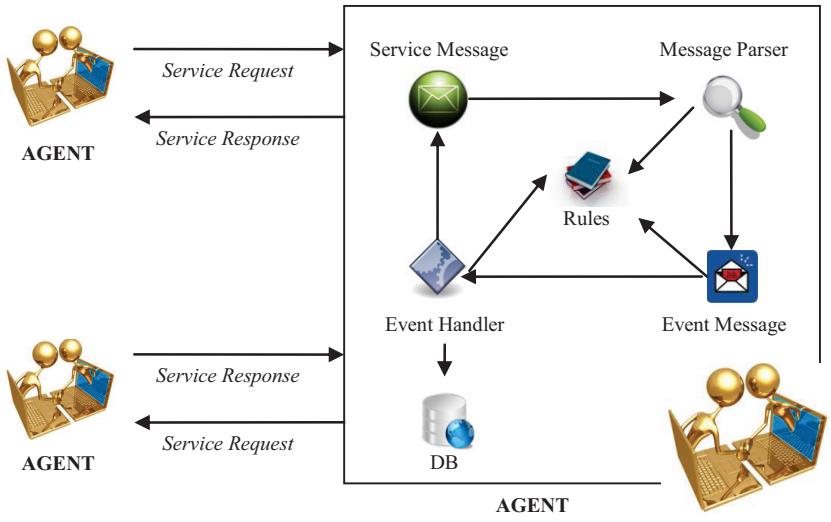


Figure 2: REA Framework

By REA framework, we design DAP method in a peer-to-peer (P2P) multi-agent network. Each role in terms of DAP Portal, EPS, online shop and consumer is regarded as an agent,

which interacts with each other to solve e-payment mismatch problem. In this framework, an agent is technically designed as a service provider, which both requests and responds an e-payment message generated based on rules that have set. The service request (i.e. the incoming service message of a receiving agent) is parsed based on the given message templates and rules to generate an event message. This event message describes how the request message should be handled and/or responded by predefined event handlers.

4 Two Alternative Implementations for DAP Method

In this section, we provide two alternative implementations of DAP method: one is a manual approach and the other is an automatic approach.

4.1 Manual Approach of Implementing DAP Method

A manual approach, shown in Figure 3, implements DAP method by including three roles of EPS, consumers and DAP Portal, where each one is regarded as an agent.

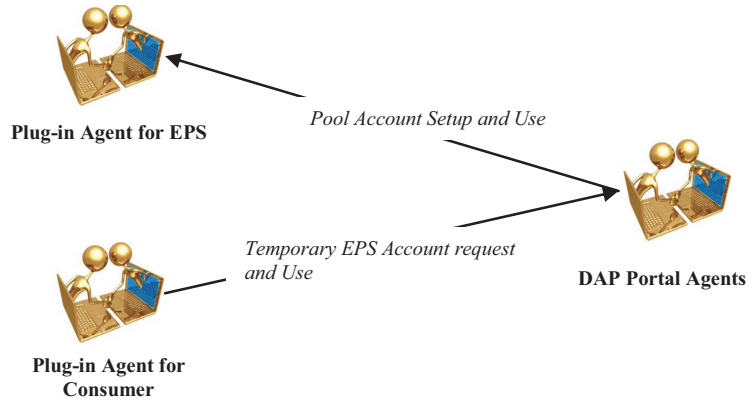


Figure 3: Manual Approach of Implementing DAP Method

In this approach, consumer is provided a plug-in agent, which is responsible for requesting a temporary EPS account for using in a target online shop from DAP Portal. The agent presents a graphic user interface to consumer for doing jobs of registering a DAP Portal account, depositing money in DAP Portal account, requesting a temporary EPS account for using in any target online shop, and checking DAP portal account status. Differently, EPS plug-in agent is designed as an e-payment integration tool by DAP method. It consists of account pool and account service manager to enable DAP Portal to setup accounts, set account use constraints and manage account balance.

The advantage of the manual method is that online shop does not need to be involved in the design and implementation of DAP method. The disadvantage is that consumer has to manually request the temporary account for using in the target online shop, which consumes more time and inconvenient.

4.2 Automatic Approach of Implementing DAP Method

An automatic approach, shown in Figure 4, implementing DAP method by removing the consumer plug-in agent and adding a plug-in agent in online shop.

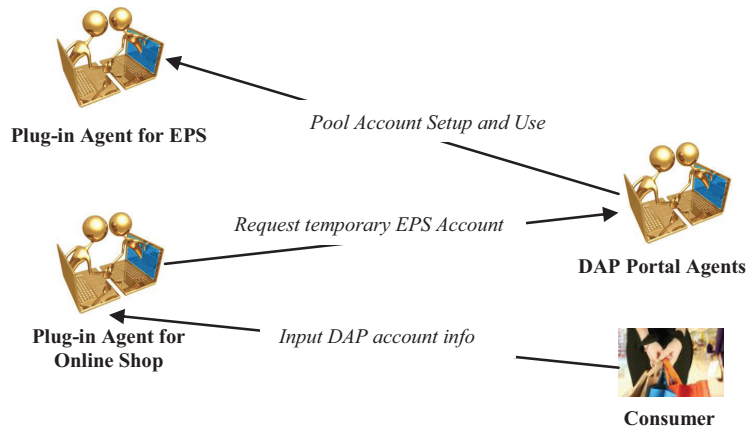


Figure 4: Automatic Approach of Implementing DAP Method

In this approach, consumers do not need to manually request any temporary target EPS account. The plug-in agent for online shop will launch an e-payment interface when the shopping cart goes to the payment stage and a consumer selects the payment method of DAP Portal. The consumer is asked to input the payment amount and his/her DAP Portal account and password to trigger a DAP payment request. DAP Portal will automatically respond to the online shop with a temporary EPS account with password. Before this response, DAP Portal will have a conversation with the EPS agent to set a payment duration and payment limit for the used account.

The advantage of this approach is that consumers are free of manual request and use of target EPS account and the shopping efficient is increased. The disadvantage is that online shops have to be integrated. Since there are millions of online shops we could find, such integration is hard and time-consuming in view of marketing effort.

5 Performance Evaluation

We adopt three steps to make experiments on the performance comparison evaluation on Manual Approach and Automatic Approach. Particularly, the steps are as follows:

(1) Implemented both Manual Approach and Automatic Approach as shown in the processes Figure 5(1) and Figure 5(2), and generated some sample data in DAP portal and several EPS.

(2) Made sample experiments on both Manual Approach and Automatic Approach to obtain sample data of experiments. The sample experiment environment is the VMWare Workstation installed in Lenova Thinkpad T60 with hard drive 100G, RAM 2G and CPU Intel T2500 2.0GHz Core Duo.

(3) Simulated the two processes shown in Fig. 5(1) and Fig. 5(2) in SIMUL8 software (simul8.com).

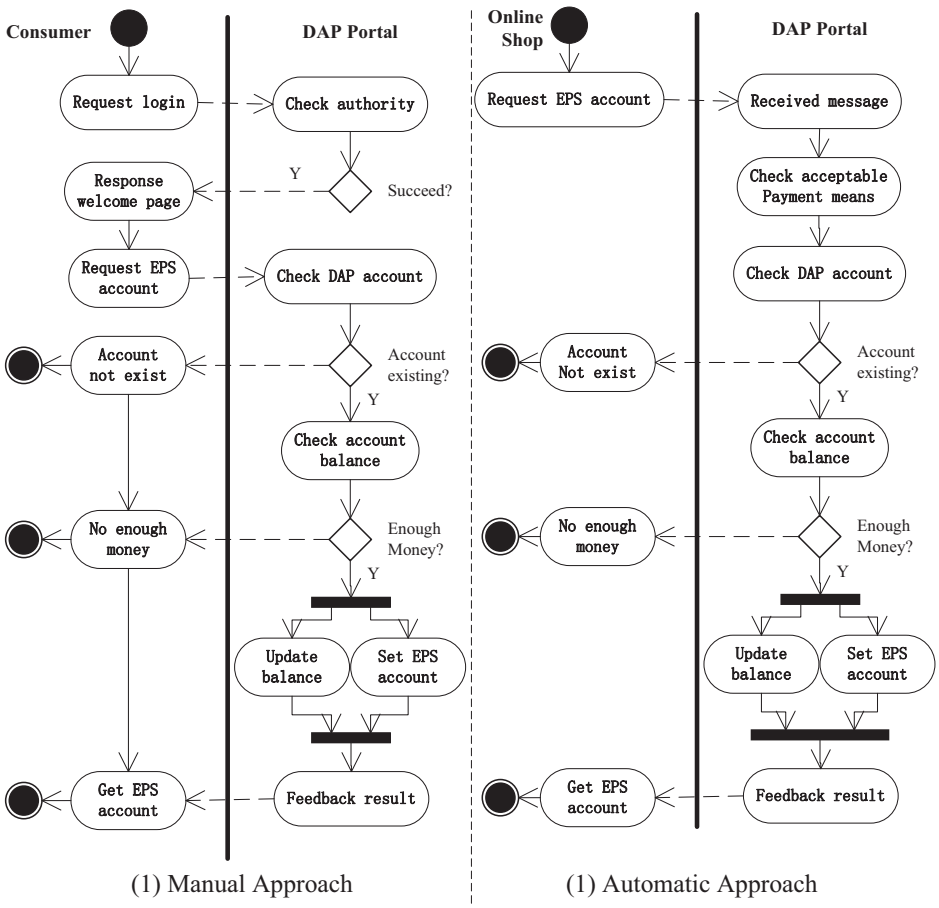
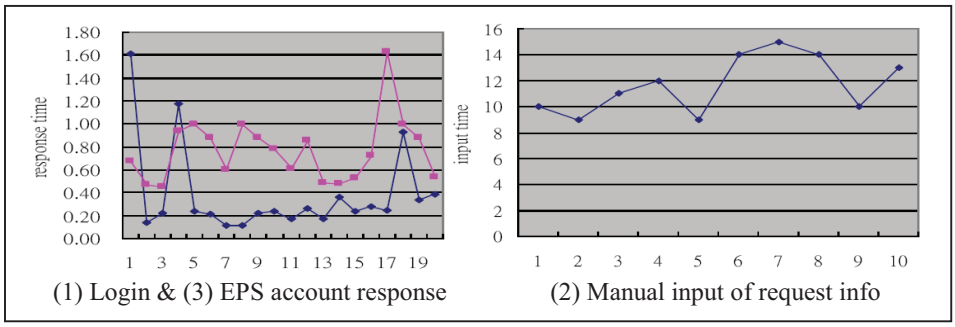


Figure 5: EPS account request process in Manual Approach and Automatic Approach

5.2 Sample Experiment Results

For Manual Approach, our sample experiments are divided into three separate sub-experiments for three sub-processes, which are: (1) the time of sending the DAP Portal login information to DAP Portal until responding the welcoming page of requesting temporary EPS account, (2) the time of manual input of temporary EPS account request information in web page form, and (3) the time of sending temporary EPC account request information to DAP Portal until it responds the temporary EPS account information or failure information. The results of the three sub-experiments are illustrated in Figure 6.



(1) Login & (3) EPS account response (2) Manual input of request info

Figure 6: Sample experiments on Manual Approach

The experiment results are summarized in Table 2, showing the time cost of minimum (Min), average and maximum (Max) for all three sub-processes.

Table 2: Summary of sample experiment results on Manual Approach (Unit: Second)

Experiment Process	**Min**	**Average**	**Max**
Response to DAP Portal login request	0.1	0.4	1.6
Manual input of EPS account request info	9	12	15
Response to EPS account request	0.5	0.8	1.6
Total	*9.6*	*13.2*	*18.2*

The sample experiments show that manual input of EPS account request information in a web form occupies a large portion of the entire time this is spent.

For Automatic Approach, our sample experiments are made on an integrated process of requesting temporary EPS account by online shop on behalf of a consumer. The result of experiments are illustrated in Figure 7.

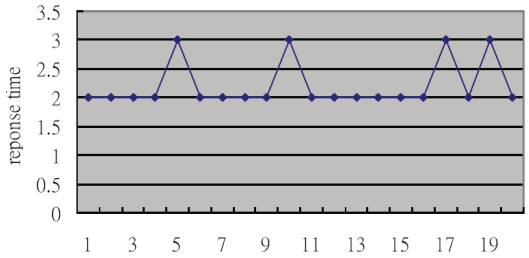


Figure 7: Sample experiments on Automatic Approach

The results can be summarized in Table 3, showing the response time at minimum (Min), average and maximum (Max).

Table 3: Summary of sample experiment results on Automatic Approach (Unit: second)

Experiment process	Min	Average	Max
Response to requesting a temporary EPS account	2	2.2	3

The sample experiments show that Automatic Approach spends much less time for responding to a temporary EPS account request than Manual Approach.

5.3 Simulation Results

To fairly compare the performance of Manual Approach and Automatic Approach to evaluate their acceptability in practice, we simulate the two approaches using the processes

described in Figure 5 (1) and Figure 5(2), using SIMUL8 software. For Manual Approach, we set default condition of input period being operated to 1 minute and use Table 2 as the running template. For Automatic Approach, we set default condition of input period being operated to 10 seconds and apply Table 3 as the running template. With this setting, we have simulation result, shown in Table 4.

Table 4: Simulation result for Automatic Approach and Manual Approach

Approach	Run Time	Number Entered	Net Number Entered	Success Percentage
Automatic Approach	1	14109	13307	94%
	2	14066	13259	94%
	3	13950	13179	94%
	4	14248	13475	96%
	5	13847	13069	94%
Manual Approach	1	2458	2187	89%
	2	2474	2191	89%
	3	2364	2154	91%
	4	2407	2166	90%
	5	2414	2179	90%

Table 4 shows that Manual Approach and Automatic Approach have success rates up to 89% and 94%, at the input time of 1 minute and 10 seconds, respectively.

Table 5: Simulation result for Automatic and Manual Approaches in 95% confident interval range

Approach		Low 95% Range	Average	High 95% Range	Input period
Manual Approach	Net enter number	2156.39707	2175.4	2194.40293	1 (min)
	Enter number	2369.13222	2423.4	2477.66778	
	Success percentage	91%	90%	89%	
Automatic Approach	Net enter number	13070.08328	13257.8	13445.51672	10 (sec)
	Enter number	13853.7174	14044	14234.2826	
	Success percentage	94%	94%	94%	

Table 5 shows the comparison of success rates between Manual Approach and Automatic Approach at 95% confident interval. It is noticeable that Manual Approach and Automatic Approach have a performance gap of 3% to 5%, but both are around or above 90% of success rate. This experiment result indicates that input periods set to 1 minute and 10 seconds for Manual Approach and Automatic Approach, respectively, are appropriate.

5.4 Discussion of Experiment Results

The experiment result shows that the Automatic Approach is more efficient than Manual Approach. However, whether this higher performance is an indicator of adopting Automatic Approach needs further discussion. Since Automatic Approach that implements DAP method relies on the integration of online shop, the adoption of this approach depends on whether an online shop is strategically allied with DAP Portal. If an online shop is unknown to DAP Portal without any integration, B2C e-payment mismatch problem still exists. In this sense, Manual Approach is the only way of resolving the problem. Thus, we believe that both implementation approaches are valuable to solving B2C e-payment mismatch problem.

6 Conclusion

This paper has proposed a new dynamic account payment (DAP) method to resolve a B2C e-payment mismatch problem. The idea of DAP method is that when B2C e-payment

mismatch problem happens, a consumer can request (or lease) a temporary payment account of the required e-payment system from a third-party e-payment integrator. This method is technically designed in a REA framework, applying the technical concepts of rules, events and agents. The designed DAP method is implemented in two approaches of Manual Approach and Automatic Approach. The former requires a consumer manually requests the target e-payment account while the latter moves the request of target e-payment account to online shops.

Performance evaluation shows that Automatic Approach is superior to Manual Approach in payment efficiency. However, Manual Approach is indispensable because not all online shops could be integrated into DAP systems.

DAP method is an important contribution to heterogeneous B2C e-payment integration. It is a new method comparing with the other two existing methods of e-payment standardization and vendor-based vertical integration. In future, we will implement a freeware based on the Manual Approach.

References

[1] BADICA, C., BRAUBACH, L. AND PASCHKE, A. 2011. Rule-based Distributed and Agent Systems. In *Proceedings of 5th International Symposium on Rules (RuleML 2011)*. Lecture Notes in Computer Science, vol. 6826. Springer, Berlin, 3-28.

[2] BALFE, S. AND PATERSON, K. 2008. e-EMV: emulating EMV for internet payments with trusted computing technologies. In *Proceedings of the 3rd ACM workshop on Scalable trusted computing (STC'08)*. ACM Press, 81-92.

[3] BERNARD, J. 1988. Use of a Rule-Based System for Process Control. *IEEE Control Systems Magazine*, October, 3-13.

[4] CHEN, Y. AND HE, Y. 2009. Study on Value Alliance Model: A New E-business Model for Enterprise. In *Proceedings of 2009 International Symposium on Information Engineering and Electronic Commerce*, IEEE Computer Society, 388-392.

[5] CHOUDHARY, V. AND TYAGI, R.K. 2009. Economic incentives to adopt electronic payment schemes under competition. *Decision Support Systems*, 46, 552–561.

[6] CUGOLA, G., NITTO, E. AND FUGGETTA, A. 2001. The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27, 9, 827-850.

[7] DAHLBERG, T., MALLAT, N., ONDRUS, J. AND ZMIJEWSKA, A. 2008. Past, present and future of mobile payments research: A literature review. Electronic *Commerce Research and Applications*, 7, 165–181.

[8] DUNKEL, J., FERNÁNDEZ, A., ORTIZ, R. AND OSSOWSKI, S. 2011. Event-driven architecture for decision support in traffic management systems. Expert Systems with Applications, 38, 6, 6530-6539.

[9] GUAN, S-U., TAN, S. L. AND HUA, F. 2004. A Modularized Electronic Payment System for Agent-based E-commerce. *Journal of Research and Practice in Information Technology*. 36, 2, 67-86.

[10] HENDRICKSON, S., DASHOFY, E. AND TAYLOR, R. 2005. An (architecture-centric) approach for tracing, organizing, and understanding events in event-based software architectures. In *Proceedings of 13th Int'l Workshop on Program Comprehension (IWPC'05)*, IEEE, Los Alamitos, CA, 227 – 236.

[11] JURIC, M. 2010. WSDL and BPEL extensions for Event Driven Architecture. *Information and Software Technology*, 52, 10, 1023-1043.

[12] KIM, C., TAO, W., SHIN, N. AND KIM, K-S. 2010. An empirical study of customers' perceptions of security and trust in e-payment systems. *Electronic Commerce Research and Applications*, 9, 84–95.

[13] LI, C. AND LI, L. 2011. A multi-agent-based model for service-oriented interaction in a mobile grid computing environment. Pervasive and Mobile Computing, 7, 2, 270-284.

[14] LI, Z. AND YE, X. 2006. Towards a Dynamic Multi-Policy Dissemination Control Model (DMDCON). *SIGMOD Record*, 35, 1, 33-38.

[15] LIM, A. 2008. Inter-consortia battles in mobile payments standardisation. *Electronic Commerce Research and Applications*, 7, 202–213.

[16] LUCKHAM, D. AND VERA, J. 1995. An Event-Based Architecture Definition Language", IEEE Transactions on Software Engineering 21(9) pp. 717-734.

[17] MARTINS, S. AND YANG, Y. 2011. Introduction to bitcoins: a pseudo-anonymous electronic currency system. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '11)*. IBM Corp. Riverton, NJ, USA.

[18] SCHUH, S. AND STAVINS, J. 2010. Why are (some) consumers (finally) writing fewer checks? The role of payment characteristics. *Journal of Banking & Finance*, 34, 1745–1758.

[19] SHAKSHUKI, E., HUSSAIN, S., MATIN, A. R. AND MATIN, A. W. 2006. P2P Multi-agent Data Transfer and Aggregation in Wireless Sensor Networks. In *Proceedings of 2006 IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS 2006)*. IEEE, Los Alamitos, CA, 645 - 649.

[20] SUGAWARA, K. 2002. An Agent-Based Framework for Developing Flexible Distributed Systems. In *Proceedings of First IEEE International Conference on Cognitive Informatics (ICCI'02)*, IEEE Computer Society, 101-106.

[21] TRZEBIATOWSKI, G., DENZINGER, J., TIMM, I.J. AND UNLAND, R. (Eds). 2004. *Multiagent System Technologies*. Lecture Notes in Artificial Intelligence, Vol. 3187, Springer-verlag, Berlin.

[22] YANG., X., SHENG, W. AND WANG, S. 2005. Agent-Based Distribution Control and Automation. In *Proceedings of IEEE International Symposium on Communications and Information Technology (ISCIT 2005)*. IEEE, Los Alamitos, CA, 1257-1262.

A MAS-based Infrastructure for Negotiation and its Application to a Water-Right Market

Bexy Alfonso, Vicente Botti, Antonio Garrido, and Adriana Giret

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
{balfonso,vbotti,agarridot,agiret}@dsic.upv.es

Abstract

This paper presents a MAS-based infrastructure for the specification of a negotiation framework that handles multiple negotiation protocols in a coherent and flexible way. Although it may be used to implement one single type of agreement mechanism, it has been designed in such a way that multiple mechanisms may be available at any given time, to be activated and tailored on demand (on-line) by participating agents. The framework is also generic enough so that new protocols may be easily added. This infrastructure has been successfully used in a case study to implement a simulation tool as a component of a larger framework based on an electronic market of water rights.

Keywords: negotiation, multi-agent systems, water-right market

1 Introduction

Last decades have witnessed an increasing interest in the design and application of computational infrastructures and tools, based on intelligent agents, to virtual architectures and organizations that give support to multiple ways of negotiation. Negotiation usually involves a dynamic collection of semi-independent autonomous entities (representing heterogenous software agents or humans, departments, enterprises, information resources and other organizations) each of which has a range of problem solving capabilities and resources at their disposal. These entities exhibit complex behaviors; they usually co-exist, collaborate and agree on some computational activity, but sometimes they compete with one another in a ubiquitous virtual scenario that is a sort of 'looking-glass reflection' of the real world.

Automated negotiation is essential to undertake complex behavior and architectures, including conflict identification, its management and resolution, search for consensus, assessment of agreement stability and equilibrium analysis in situations where two or more parties have opposing preferences [15]. This line of research has addressed developments for group decision support systems and meeting support systems, which can be extrapolated to automated negotiation [8, 10].

Therefore, negotiation is interesting from an application point of view but also to provide artifacts that facilitate the design, experimentation and simulation of involving agreements. In this paper we intend to profit from that experience and look at one of such artifacts: a generic negotiation MAS-based framework in which different negotiation protocols may become available. The contributions of this general framework are multiple. i) As it is defined for the Magentix2 [2] platform for open MASs, it embodies easy communication and interaction protocols among agents, roles and organizations. It also uses Jason [5] as a high-level language for programming agents, providing them with high reasoning skills. ii) Interactions among agents aim at achieving individual and global goals, and are structured via collaboration, argumentation, negotiation and, eventually, via agreements and contracts [19]. iii) It is composed of flexible negotiation mechanisms and their supporting preparatory and ending activities. iv) As a by-product, it creates standardized negotiation modules to be grafted into larger scenarios or as plug-ins in peer to peer interactions. v) It has been used as a proof of concept in *mWater* [6, 13], a water-right market where negotiation is essential, also embedded in a decision support system where water usage is subject to conflicts whose solution may involve different types of negotiation. vi) It provides new areas of opportunities for an agreement computing solution [19], including agility, heterogeneity, reconfigurability, cooperation, argumentation, reputation and trust issues under a MAS perspective.

2 Technological Background

There are various technologies involved in the implementation of our MAS infrastructure. First, the MAS platform in itself, which manages agents and their interactions, allowing the information exchange among them and also with the environment. Second, a language to define the agents behavior —in this case Jason, which follows the agents' BDI model. Third, to support the human-software agents' interactions it is necessary to design a Graphical User Interface (GUI) and an artifact to orchestrate the communication between the GUI and the MAS.

2.1 MAS Platform

We use Magentix2 [2] as our MAS platform because: i) it provides powerful techniques to facilitate agents' communication; ii) it supports interactions protocols between agents organizations/societies through conversations management; iii) it allows the use of high-level reasoning structures when programming the agents; and iv) it includes security issues for distributed systems, so it offers a dynamic and flexible model for complex systems. In short, Magentix2 gives us support at the three levels stated in [16]: organization level, interactions level and agent level.

Conversations Factory: an Artifact for Communication

A Conversations Factory [11] is mainly a Magentix2 mechanism to support FIPA interaction protocols [12]. Each conversations factory allows us to keep a complete interaction among two or more agents having an *initiator* (the one who starts the

conversation) and one or more *participants* (the other agents in the conversation). The two main structures supporting conversations are *CProcessor* and *CFactory*. The former manages the sent/received messages in each step of the conversation, performing the corresponding actions, and determines the next step in the conversation. The latter creates the conversations and the *CProcessors* that correspond to a specific protocol. If the agent is playing the role of *initiator*, the conversation can start without needing an external event. On the other hand, if the agent is a *participant* an event is required for it to be part of the conversation.

2.2 Programming language

Magentix2 allows us to use a high-level language for programming agents, in this case is Jason [5], which is an extension of the AgentSpeak language. AgentSpeak allows us to define agents in terms of beliefs, goals and plans. Beliefs represent the vision of each agent of the current state of the world in which such an agent is situated. Beliefs change frequently due to a 'perception' of the agent over its environment, because some information has been sent to it through a message, or because it explicitly modifies those beliefs as a consequence of some previous reasoning. Agents' goals represent the agents' intention to reach a state where they believe the goals are true, what is called 'achievement goal'. Another kind of goal is satisfied when the agents retrieve updated information from their belief base 'test goals'. Finally, plans are just a sequence of steps that allow agents to reach some goals. The fact of adding a goal acts as a triggering event for executing the corresponding planned sequence of actions. There are other actions that act as triggering events for plan executions as it is the case of the deletion of achievement goals, adding and deletion of beliefs, and adding or deletion of test goals. If this sequence of actions does not fail, the goal is successfully reached.

Jason provides a kind of action called 'internal action'. It is a structure that allows us the execution of legacy code (Java in this case). Thanks to this, the agent has access to the structures provided by the platform [3] in order to make use of the conversations factory in a more simplified way. By using some of the Magentix2 predefined internal actions, each agent can customize what it does in those steps of the conversations on which it needs to perform some 'reasoning', delegating details such as synchronization, timeouts, errors management, etc. in the platform. Magentix2 is also responsible for updating the state of each agent (by updating its beliefs) for it to make decisions, which behaves as an indirect communication.

3 Our Generic Negotiation Model

The infrastructure for a generic negotiation model can be seen as a set of entities and roles regulated by mechanisms of social order, and created in order to negotiate with some good, service or resource.

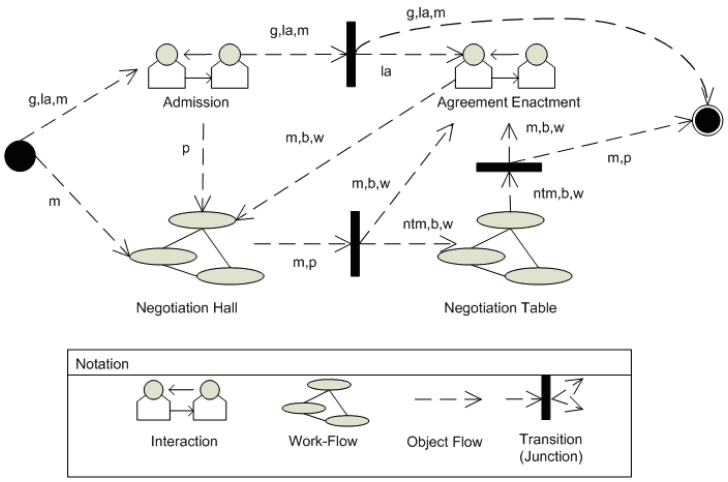


Figure 1: Generic Negotiation workflow structure. Roles: g - guest; p - participant; b - black; w - white; m - mediator; ntm - negotiation table manager; la - legal authority.

3.1 Main Structure

Our negotiation model follows a MAS specification based on conversations, and regulation on (structural) norms. It is defined as a generic organization for negotiation (see Figure 1)[1], where any participating agent may become involved in a negotiation process.

After admission is granted, each negotiation involves first a preliminary process of invitation and filtering of parties, then the negotiation process itself and, finally, some form of settlement process through which the agreements among participants are made explicit and, if appropriate, communicated to the organization.

3.2 Users and Roles

There are seven roles that interact in the model, as depicted in Figure 1. A guest role (g) is a user that wants to enter the negotiation. The guest may be specialized into a real participant (p), and furthermore as black (b) and white (w) to differentiate the parties that are acting in a given negotiation. Finally, there are four types of staff roles. The mediator role (m) represents a negotiation facilitator agent who runs standard activities, such as managing the specific parameters of the negotiation protocols. The negotiation table manager role (ntm) represents an agent who executes activities that are specific of a given negotiation protocol, for example accept valid negotiators, tune negotiation parameters of the table, mediate in the negotiation or conflict resolution process, expel negotiators, etc. The legal

[1] At a glance, each interaction/conversation represents an atomic process and/or dialog among agents; a workflow represents complex interaction models and procedural prescriptions. The dynamic execution is modeled through arcs and transitions, by which the different participating roles of the organization may navigate.

authority role (la) represents an agent who is in charge of activities for agreement enactments that are executed as a result of a successful negotiation process.

Note that, unlike other approaches, our definition introduces an explicit intelligent management into the negotiation model in the form of the mediator and negotiation table manager. These two roles have demonstrated to be very helpful to improve and facilitate the internal behavior of the organization. On the one hand, the mediator must be aware of the organizational conventions, the rules of the market and the negotiation structure. On the other hand, the negotiation table manager must obey the particular rules of the protocol to be used within the negotiation, and this is usually domain-dependant —different protocols require the application of different sequences of steps.

3.3 Workflow

The workflow activities in the generic negotiation model of Figure 1 are specified through a main structure which includes two other workflows: the *NegotiationHall* and *NegotiationTables*, plus two supporting interactions, *Admission* and *AgreementEnactment*.

Admission. It allows Guest agents to register to become a Party, and to 'jump start' a negotiation process. Once negotiation is open, this interaction allows Party agents to enter and negotiate by registering individual data for management and enforcement purposes (these data are domain-dependent and can be used, for example, for enforcing particular conventions and managing activities).

NegotiationHall. Actual negotiation starts here (see Figure 2), where Party agents become aware of any activity and/or initiate concurrent activities for negotiation. There are three interactions that provide virtual scenarios for the: i) creation of, and invitation to, negotiation tables (NTC); ii) exchange of information about active agreements and ongoing negotiation tables (IE); and finally, iii) execution of specific activities in case of an anomalous/critical situation (CS).

Negotiation Tables are created in two ways: i) by the organization itself, for example periodic negotiation tables about a set of issues, or ii) initiated on-demand by a participating agent. The negotiation tables are created in the NTC interaction, which responds to the *FIPA request* standard protocol [12]. Figure 3 and 4 show the steps of the protocol from the Party's perspective (*initiator*) and from the Mediator's perspective (*participant*), respectively. It issues the following illocution:

$request(p_x, m, open, protocol(params), \delta, pt, at, C)$, where the semantic is as follows. Party agent p_x requests (see Figure 3) to the Mediator, m, to *open* a negotiation table with a given negotiation *protocol*. This protocol is instantiated with the set of values for the parameters *params*. The table is created to negotiate about a deal δ. The requesting party, p_x, will participate as pt that can take one of these values: p, that is an observer Party; a Black party b; or a White party w. at is the access type that can be *Public*, any body can be invited; or *Private*, only Party agents that fulfill the set of constraints C can participate in the negotiation table.

When the Mediator, m, receives a request to open a negotiation table (see Figure 4), it instantiates a new Negotiation Table scenario with the requested negoti-

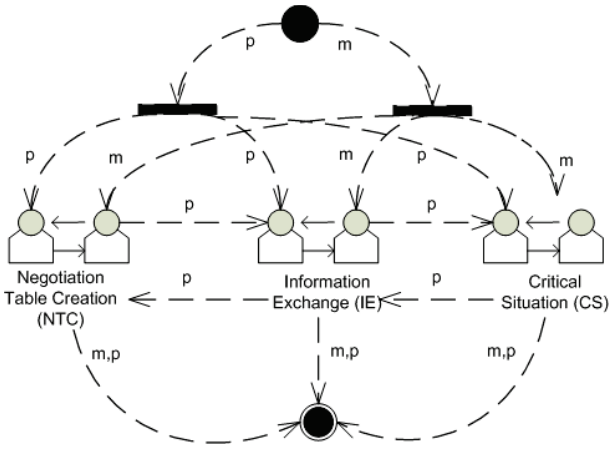


Figure 2: *Negotiation Hall* workflow structure.

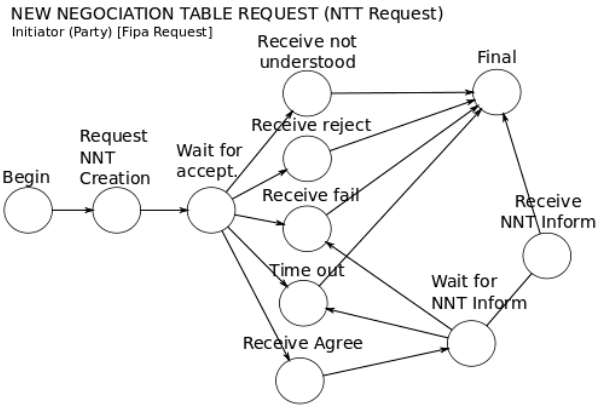


Figure 3: Party's behavior for requesting a New Negotiation Table.

ation protocol, for example a standard double auction, a face-to-face negotiation, a blind double auction, etc., and the given parameters. Moreover, a Negotiation Table Manager, *ntm*, is created to manage the execution of the negotiation table. Next, m issues an information illocution to the p_x agent who requested the table.

$inform(m, p_x, table_{ID}, error)$, where $table_{ID}$ is the ID of the new table if it was successfully created, or a *null* value when the table can not be created due to *error* conditions.

In order to complete the negotiation table creation the Mediator needs to invite other Party agents to the new negotiation table. When the created negotiation table has a *Public* type of access, the m broadcasts an invitation message to all the participants:

$inviteAll(m, table_{ID}, protocol, \delta, C)$; in other words, the invitation message states the $table_{ID}$ of the negotiation table that is receiving players; the negotiation pro-

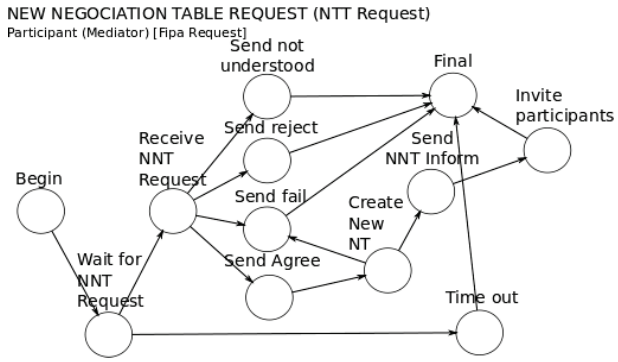


Figure 4: Mediator's behavior during the conversation for Opening a New Table.

tocol *protocol* used in that table; the set of issues, δ, that is being negotiated; and the set of constraints, C, to participate in are also made public.

On the other hand, if the created negotiation table has a $Private$ type of access, the m has to select first the set of possible candidates to invite, say $P_{table_{ID}}$, and then send an invitation message to every such candidate:

$invite(m, p_y, table_{ID}, protocol, \delta, C)$, where each candidate $p_y \in P_{table_{ID}}$.

NegotiationTable. It is organized in a flexible and scalable fashion in order to easily include new negotiation protocols. Each instance of a *Negotiation Table* interaction is managed by a *Negotiation Table Manager*, *ntm*, who knows the structure, specific data and management protocol of the given negotiation protocol. The framework provides pre-defined protocols such as face-to-face, Dutch auction, English auction, standard double auction, closed bid envelope, blind double auction with mediator, among others. Nevertheless, new negotiation protocols may be easily added provided that the new definition complies with the generic structure.

Every generic negotiation table is defined as a three interaction structure (see Figure 5). The first interaction is *Registration*, in which the *ntm* applies a filtering process to assure that only valid agents can enter a given negotiation table (recall situations when a private negotiation table is executing or only a sub-group of Party agents that fulfill a set of constraints may participate in the table). The specific filtering process will depend on the given negotiation protocol and possibly on domain specific features. The second interaction is the negotiation protocol, in which the set of steps of the given protocol are specified (see bellow for a sample negotiation protocol specification). Finally, in the last interaction, *Validation*, a set of closing activities are executed, for example registering the final deals, stating the following steps for the agreement settlement, verifying that the leaving party satisfies the leaving norms of the negotiation table, etc. The set of activities to be executed in this interaction is domain specific and will also depend on the given negotiation protocol.

AgreementEnactment. Once an agreement has been successfully reached, it is settled here according to the given conventions. This may be a rather elaborate

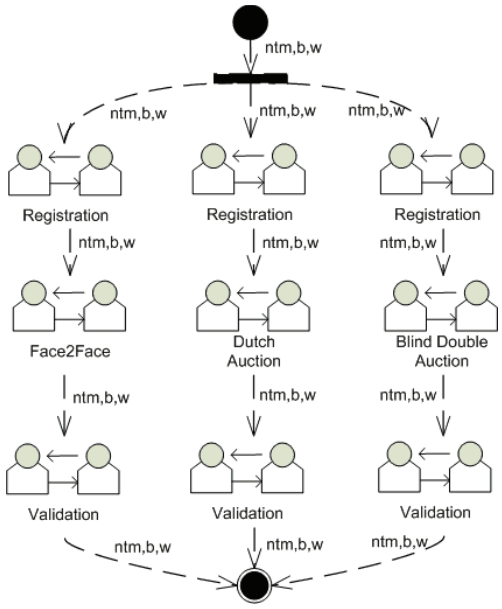


Figure 5: *Negotiation Table* workflow structure.

process. First of all, the Mediator checks whether or not the agreement satisfies some formal conditions. If the agreement complies with these, a transfer contract is agreed upon and signed by the Party agents involved, and then the agreement becomes active. Once an agreement is active it may be executed and, consequently, other Party agents may initiate a grievance procedure that may overturn or modify the agreement. Even if there are no grievances that modify a contract, parties may not fulfill the contract properly and there might be some contract reparation actions. If things proceed smoothly, the agreement subsists until maturity.

4 Case Study: *mWater*, a Water-Right Market

4.1 *mWater* Overall Description

Water scarcity is a major concern in most countries. It has been sufficiently argued that more efficient uses of water may be achieved within an institutional framework where water rights may be negotiated under different market conditions [20]. In hydrological terms, a water market can be defined as an institutional, decentralized framework where users with water rights are allowed to voluntarily trade them, always fulfilling some pre-established norms, to other users in exchange of some compensation [14, 20]. Because of water's unique characteristics, such markets do not work everywhere, they are not homogenous, nor do they solve all water-related issues [20]. Also, even subtle changes in the market design (allowed participants, legislation, protocols, etc.) are very costly and difficult to evaluate.

mWater is a particular instance of the MAS infrastructure for negotiation pre-

sented above, and it is used as a simulation tool for What-If Analysis of water-right markets policies [6, 13]. More specifically, *mWater* assists in designing appropriate water laws and regulate, either privately or publicly, the users' actions, interactions and their eventual trade.

4.2 *mWater* as a Simulation Tool

mWater builds on a MAS infrastructure, simulates a flexible water-right market, and includes its own ontology for dealing with water issues and both the trading and grievance processes. We have focused our model on humans' actions: agents are the crucial component in these models and our interest relies on the social aspect of the market, which is usually missing in other markets in the literature. This simulator includes heterogeneous and autonomous intelligent agents representing the different independent entities in the market. We focus on demands and, in particular, on the type of regulatory (in terms of norms selection and agents behavior), and market mechanisms that foster an efficient use of water while also trying to prevent conflicts among parties. In this scenario, this system plays a vital role as it allows us to define different norms, agents behavior and roles, and assess their impact without jeopardizing the real-world market, thus enhancing the quality and applicability of its results as a decision support tool.

The user can configure simulation parameters such as: the group of water-users that will participate in the market[2], the norms and regulations that define the policies in the market, the seasons in which the water-right transfer will take place, etc. The simulation tool executes with a given configuration and the user can assess the market's behavior by means of indicators such as: number of water-right transfer agreements, volume of water transferred, amount of money, overall social satisfaction of the water-users that participated in the market, number of conflicts generated, etc.

4.3 *mWater* in Action

Figure 6 shows a snapshot of the *mWater* simulator in action. This interface allows the user, i.e. the water policy maker, to choose different input values that involve simulation dates, participants, norms (in the form of protocols used during the trading negotiation) and some decision points that can affect the behavior of the participants[3].

To implement human-agents interactions, in order to have a tool for studying different behaviors and situations, it was necessary to create some GUIs with the required options for the human to make changes in the system and pass information to the rest of agents at execution time. For this we implemented a Web page with PHP as scripting language and an interface application to 'pass' all the requests

[2]It is important to point out that the simulation we have developed is a mixed-initiative simulation in which there are software agents that are completely autonomous/automated and other software agents that are simple interfaces for human users. In this way, it is very easy to include complex social behaviors that are hard to implement or highly time consuming.

[3]In our current implementation, these additional decision points rely on a random basis, but we want to extend them to include other issues such as short-term planning, trust, argumentation and ethical values.

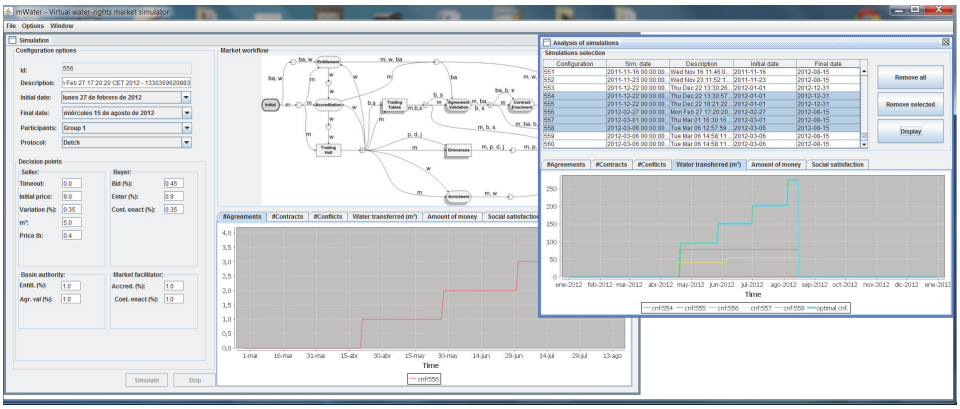


Figure 6: Snapshot of the *mWater* simulator.

from the Web page to the MAS, and all the results from the MAS to the Web page. This makes possible to count on a MAS composed by a mixture of automated agents and humans, and even a system completely based on automated agents. Figure 7 shows how a user can participate in a Japanese Auction of a water right, by interacting with other human or automated agents.

This simulation tool allows users to analyze: i) how the conventions, norms and negotiation protocols of the market change over time; ii) how participants in these markets (re)act to these changes; and ii) how to extrapolate the empirical outcomes of the market, in terms of economic and environmental impact, to deal with the social (welfare) aspect of the market. Our preliminary experiments shed light on the benefits that a collaborative AI perspective may bring to the policy makers, general public and public administrators. Also, from the experts' evaluation we can conclude that a tool like this provides an advantageous tool to help build a more efficient water market with more dynamic norms.

5 Further Uses for the Generic Negotiation Model

The infrastructure for generic negotiation that we have presented here has several application uses, from both the academia and industry point of view. From the academia standpoint, it can be used as a testbed for other developments within the agreement technologies paradigm (`http://www.agreement-technologies.org`). In particular, there are several challenging questions on:

- Organization and roles. How beneficial is the inclusion of collective, heterogeneous roles, their collaboration (and trust theories) and how the policies for either flat or hierarchical group formation affect the system behavior? To answer this we need to capture all those roles currently recognized by legislation that have any impact on negotiation and agreement management, specially in grievances and conflict resolution.

- Collective decision-making, reconfigurability, cooperation, social issues and

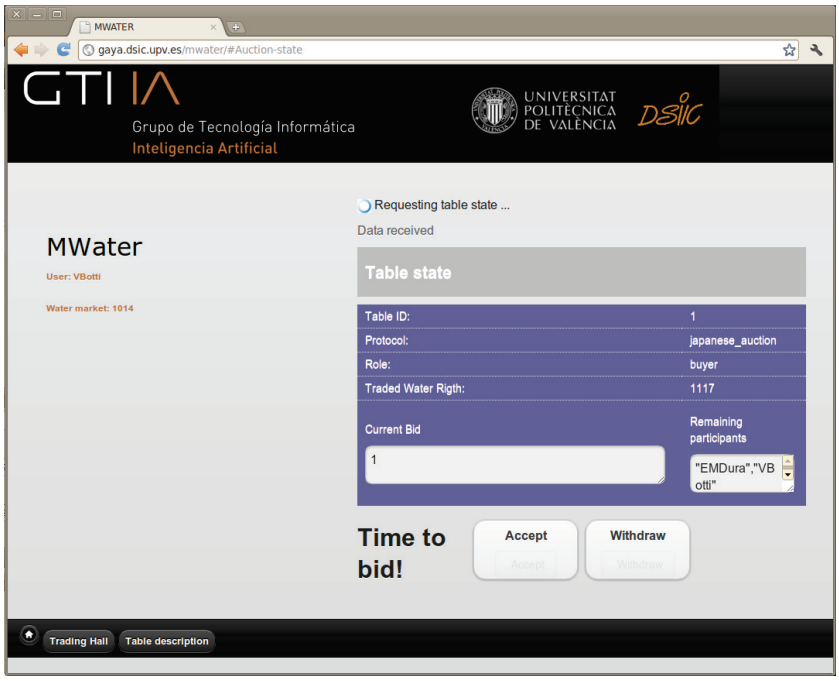


Figure 7: Snapshot of the human-agents GUI. The user can participate in a Japanese auction with other humans and/or automated agents.

coordination. What is the impact of argumentation, judgement aggregation, reputation, prestige and multi-party negotiation in the system performance? The answer to this question is not straightforward and requires simulation tools for performance assessment, as seen in section 4.

- Institutional limitations. What type of enforcement mechanisms are necessary and how they change *w.r.t* the evolution of regulation? This is highly related to the definition, adoption and compliance of (emerging) norms and, more particularly, how to model and reason on them? To solve this, we need to face the problem of expressiveness: the type of norms we have dealt with so far has a formal representation, but other types of representation may be more complex to handle. Finally, ensuring norm compliance is not always possible (or desired), so norm violation and later detection via grievances usually makes the environment more open, dynamic and realistic for taking decisions.

From the industry standpoint, there exist further applications in the form of simple tools that can be embedded within our MAS framework:

- A decision-support tool for policy simulation. Policy-making is a hard task. Designing and taking legal decisions involves a complex balance among different factors, such as economic, social, administrative or environmental aspects. Consequently, a decision-support tool that allows policy-makers to

easily predict, analyze and measure the suitability and accuracy of modified regulations for the overall system, before using other operational tools for the real floor, shows very important. Our experiments with *mWater* shed light on the benefits that a collaborative AI perspective for a water-right market may bring to the policy-makers, general public and public administrators.

- A GUI tool for human negotiation that facilitates the human interaction with software agents. Particularly, our GUI provides a simple, though effective way to set up parameters and dynamic changes, which affect the performance of the system, during the negotiation process (and also while simulating this process). Moreover, it intuitively provides the results generated after such an interaction process, which can be used as an analysis tool to evaluate protocols.

- A general tool open to other negotiation processes, such as other electronic markets; the workflow structure, roles and negotiation interaction remains the same. Our experiences show that our negotiation framework is general enough and can be valid for other markets. Particularly, we are applying these ideas to a by-product exchange market to boost the re-use of waste, thus being part of our current work.

6 Conclusions through Related Work

Computing has become an inherently social activity rather than a solitary one, leading to new forms of conceiving computational systems which require both interaction and negotiation. Some proposals have been effectively developed in literature to implement a negotiation framework. That is the case of the Jade platform [1, 4], which is a FIPA compliant platform that provides Java classes to handle all the FIPA interaction protocols. In this sense, the agents' interactions must be also programmed in Java by using the constructions provided by the platform. Another multi-agent platform with support for interaction protocols is Jadex [7, 17]. Jadex follows a typical BDI model and can be executed alone or under other communication platforms using *adapters*. A Jadex agent is defined through an XML file and the Java classes that implement it. Jadex also owns the 'interaction protocols' capability, offering built-in support for most of the FIPA interaction protocols. However, both Jade and Jadex use Java classes for implementing FIPA interaction protocols, so the programmer can not use other specialized programming languages, such as AgentSpeak, more expressive to model and describe agents. This does not prevent us from addressing the problem using the Java approach; in fact, so far it has been broadly used. However, in MASs, it is desirable to use tools and languages that better fit with the autonomous and proactive agents' nature. In this sense, Magentix2 [2] supports a high-level language for programming conversational agents (i.e. agents whose interactions respond to interaction protocols) and the rest of the capabilities offered by similar platforms. It also owns a conversations manager that stores and automatically adds the information required in the creation of messages during the conversation. Moreover, with Magentix2 it is possible to dynamically modify the sequence of steps in the interaction protocol in order to create more

open and flexible conversations (new states and transitions between the conversation steps can be created at execution time). These features have been partially included in other platforms, whereas all of them are included in Magentix2, which makes it become an ideal infrastructure for a negotiation architecture.

From our point of view the common denominator in most of the current real, social systems is, interestingly, a negotiation process. Although some works have proposed the construction of formal conceptual models with some negotiation [9, 18], they do not always report significant advances from a collaborative AI perspective. In this paper we have established the infrastructure foundations for the specification of a multi-agent-based negotiation framework as the basis for modeling virtual scenarios, and put it into practice within a water-right market, where negotiation plays a vital role. The work presented in this paper is based on the lessons learned in [6, 13]. But now, the generic negotiation framework has been implemented in Magentix2 to offer a flexible and easy way to adapt to applications in which autonomous features in regulated environments are required. Thus, the technical contributions of this work are:

- Design a generic MAS infrastructure that captures the main steps that happen in an agent-based scenario, including mechanisms for exchanging information, negotiating and dealing with the critical situations that may appear thereafter.
- Introduce the users and intelligent roles that are necessary within an agent-based setting. Differently to existing approaches, we introduce the roles of intelligent mediators, which are very valuable for the process.
- Provide multiple negotiation strategies that are managed in a three-step unified way: registering, negotiating and validating the reached agreement. This also allows us to include different protocols in a flexible fashion.
- In order to test the applicability of this generic framework, we have put these ideas into practice with *mWater*. This water market is very illustrative and has allowed us to explore the influence that the repetitive interaction of participants exerts on the evolution of the market. Also, it has given us enough evidence that the generic framework for negotiation provides a solid foundation for complex markets.

Acknowledgments

This paper was partially funded by the Consolider AT project CSD2007-0022 INGENIO 2010 of the Spanish Ministry of Science and Innovation; the MICINN projects TIN2011-27652-C03-01 and TIN2009-13839-C03-01; and the Valencian Prometeo project 2008/051.

References

[1] Jade. `http://jade.tilab.com`.

[2] J. M. Alberola, J. M. Such, A. Espinosa, V. Botti, and A. García-Fornes. Magentix: a Multiagent Platform Integrated in Linux. In *EUMAS*, pages 1–10, 2008.

[3] B. Alfonso, E. Vivancos, V. Botti, and A. García-Fornes. Integrating jason in a multi-agent platform with support for interaction protocols. In *Proceedings of the compilation of the co-located workshops on AGERE!'11*, SPLASH '11 Workshop, pages 221–226, New York, NY, USA, 2011. ACM.

[4] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley and Sons, 2007.

[5] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-agent Systems in Agent Speak Usign Jason*. John Wiley & Sons, 2007.

[6] V. Botti, A. Garrido, J. A. Gimeno, A. Giret, and P. Noriega. The role of MAS as a decision support tool in a water-rights market. In *AAMAS 2011 Workshops, LNAI 7068*, pages 35–49. Springer, 2011.

[7] L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: a BDI agent system combining middleware and reasoning. In M. C. M. K. R. Unland, editor, *Software Agent-Based Applications, Platforms and Development Kits*, pages 143–168. Birkhäuser-Verlag, 9 2005.

[8] G. B. DeSanctis and B. Gallupe. A foundation for the study of group decision support systems. *Knowledge based systems*, 33(5):589–609, 1987.

[9] P. Eckersley. Virtual markets for virtual goods, 2003. Available at http://www.ipria.com/publications/wp/2003/IPRIAWP02.2003.pdf (accessed May 2012).

[10] J. Fjermestad and S. Hiltz. Group support systems:a descriptive evaluation of case and field studies. *Journal of Management Information Systems*, 17(3):115–161, 2001.

[11] R. L. Fogués, J. M. Alberola, J. M. Such, A. Espinosa, and A. García-Fornes. Towards Dynamic Agent Interaction Support in Open Multiagent Systems. In *Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence*, volume 220, pages 89–98. IOS Press, 2010.

[12] Foundation for Intelligent Physical Agents. *FIPA XC00025E: FIPA Interaction Protocol Library Specification*.

[13] A. Giret, A. Garrido, J. A. Gimeno, V. Botti, and P. Noriega. A MAS decision support tool for water-right markets. In *Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems (Demonstrations@AAMAS)*, pages 1305–1306, 2011.

[14] J. Gomez-Limon and Y. Martinez. Multi-criteria modelling of irrigation water market at basin level: a Spanish case study. *European Journal of Operational Research*, 173:313–336, 2006.

[15] G. Kersten and H. Lai. Satisfiability and completeness of protocols for electronic negotiations. *European Journal of Operational Research*, 180(2):922–937, 2007.

[16] M. Luck and AgentLink. *Agent technology: computing as interaction: a roadmap for agent-based computing. Compiled, written and edited by Michael Luck et al.* AgentLink, Southampton, UK, 2005.

[17] A. Pokahr, L. Braubach, A. Walczak, and W. Lamersdorf. *Developing Multi-Agent Systems with JADE*, chapter Jadex-Engineering Goal-Oriented Agents, pages 254–258. Wiley and Sons, 2007.

[18] C. Ramos, M. Cordeiro, I. Praça, and Z. Vale. Intelligent agents for negotiation and game-based decision support in electricity markets. *Engineering intelligent systems for electrical engineering and communications*, 13(2):147–154, 2005.

[19] C. Sierra, V. Botti, and S. Ossowski. Agreement computing. *KI - Künstliche Intelligenz*, 25(1):57–61, 2011.

[20] M. Thobani. Formal water markets: Why, when and how to introduce tradable water rights. *The World Bank Research Observer*, 12(2):161–179, 1997.

A New Platform for Developing, Management and Monitoring Open MultiAgent Systems

Carolina Zato[1], Javier Bajo[2], and Juan M. Corchado[1]

[1] *Departamento Informática y Automática, Universidad de Salamanca, Salamanca, Spain*
carol_zato@usal.es,corchado@usal.es

[2] *Universidad Pontificia de Salamanca, Salamanca, Spain* jbajope@usal.es

Abstract

This article presents PANGEA, an agent platform to develop open multiagent systems, specifically those including organizational aspects such as virtual agent organizations. The platform allows the integral management of organizations and offers tools to the end user. Additionally, it includes a communication protocol based on the IRC standard, which facilitates implementation and remains robust even with a large number of connections. The introduction of a CommunicationAgent and a Sniffer make it possible to offer web services for the distributed control of interaction

Keywords: multiagent platform, Web services, virtual organizations, IRC protocol.

1 Introduction

One of the current lines of investigation for multiagent systems aims to create an increasingly open and dynamic system. This involves adding new capabilities such as adaption, reorganization, learning, coordination, etc. Virtual agent Organizations (VOs) [9][12] emerged in response to this idea; they include a set of agents with roles and norms that determine their behavior, and represent a place where these new capabilities will assume a critical role. Possible organizational topologies and aspects such as communication and coordination mechanisms determine in large part the flexibility, openness and dynamic nature that a multiagent system can offer. There are many different platforms available for creating multiagent systems that facilitate the work of the agent; however those that allow for the creation of VOs number much fewer, and it is difficult to find one single platform containing all of the requirements for a VO. The remainder of the paper is structured as follows: the next section introduces some existing platforms. Section 3 presents an overview of the main characteristics of the platform. Finally, section 4 explains a case study and presents some results.

2 Related Works

All platforms for creating multiagent systems existing to date should be studied according to two principal categories: those that simply support the creation and interaction of agents, and those that permit the creation of virtual organizations with such key concepts as norms and roles. We will first present those platforms that do not incorporate organizational aspects. The FIPA-OS [8] agent platform was created as a direct derivative of the FIPA [25] standard. Another agent platform is the April Agent Platform (AAP) [7] which, unlike the majority of platforms using Java, implements the April language [24]; its development and technological support has been discontinued. One of the strong points of this platform is that it provides services to facilitate the development and deployment of agents on the Internet and is also compliant with Web Services and Semantic Web standards. One of the most recent platforms still in development is JASON [3] [4]. Its greatest contribution is the easy implementation of BDI agents [28]. The Java-developed platform contains AgentSpeak in its nucleus, an interpreter agent that acts as a language extension [27]. The platform offers two operation modes: one that runs all agents in the same machine, and another which allows distribution using SACI (Simple Agent Communication Infrastructure) [29], which in turn uses KQML [10] language instead of RIPA-ACL [11]. In practice, the most used platform for developing multiagent systems in real case studies is JADF (Java Agent Development Framework) [2]. The JADE platform focuses on implementing the FIPA reference model, providing the required communication infrastructure and platform services such as agent management, and a set of development and debugging tools. Jadex [5] is a software framework for the creation of goal-oriented agents following the belief-desire-intention (BDI) model. The Jadex project facilitates a smooth transition from developing conventional JADE agents to employing the mentalistic concepts of Jadex agents. With the exception of JASON, these platforms follow the FIPA standard, can create agents (some with different models), and manage communication among agents and services. With VOs, however, it is necessary to consider the normative and organizational aspects that the platform itself must provide. MadKit [1] was one of the first platforms to consider basic organizational aspects. The platform architecture is rooted in the AGR (agent-group-role) model [16]; however, while it can handle the concept of role, it does not consider a role a class entity, and the behavior associated with the role is directly implemented in the agent who assumes it. Roles are strongly linked to agent architectures. This approach harms the reusability and modularity of organizations [14]. Another pioneering platform with regards to structural aspects was Jack Teams [23]. JACK Teams is an extension of JACK Intelligent Agents [6], which provides a team-oriented modelling framework. Both are extensions of the Java programming language; the implemented source code is first compiled into regular Java code before being executed. S-MOISE+ is an organizational middleware that follows the MOISE+ model [18]. It is an extension of SACI [29] where the agents have an organizational aware architecture. Our research found systems developed in conjunction with JASON and using S-Moise+ as middleware to achieve a more complete model [17]. The result was J-Moise+ [1], which is very similar to S-Moise+ regarding overall system concepts. The main difference is how the agents are programmed: in S-Moise+ agents are programmed

in Java (using a very simple agent architecture), while in J -Moise+ they are programmed in AgentSpeak. One of the main disadvantages of VO oriented platforms is the slight loss in the concept of service and, consequently, the management of these services and the Directory Facilitator (DF) described in the FIPA standard. THOMAS was developed in response to this twofold need. THOMAS is based on the idea that no internal agents exist and architectural services are offered as web services. As a result, the final product is wholly independent of any internal agent platform and fully addressed for open multiagent systems [15]. Finally, one of the most complete and recent platforms that we found is Janus [13]. Janus is the next step towards platform organizations known as TinyMAS (no longer under development.). This platform was specifically designed to deal with the implementation and deployment of holonic and multiagent systems. Its primary focus is to support the implementation of the concepts of role and organization as first-class entities (a class in the object-oriented sense). This consideration has a significant impact on agent implementation and allows an agent to easily and dynamically change its behaviour [14]. In conclusion, it could be said that when dealing with all aspects of complex multiagent systems such as VOs, it is also necessary to deal with multiple levels of abstractions and openness, which is not the case for most solutions.

3 PANGEA Overview

As we have mentioned, we are looking for a platform that can integrally create, manage and control VOs. In general terms, the proposed platform includes the following characteristics:

- Different models of agents, including a BDI and CBR-BDL architecture.
- Control the life cycle of agents with graphic tools.
- A communication protocol that allows broadcast communication, multicast according to the roles or suborganizations, or agent to agent.
- A debugging tool.
- Module for interacting with FIPA-ACL agents.
- Service management and tools for discovering services.
- Web services.
- Allow organizations with any topology.
- Organization management.
- Services for dynamically reorganizing the organization.
- Services for distributing tasks and balancing the workload.
- A business rules engine to ensure compliance with the standards established for the proper operation of the organization.

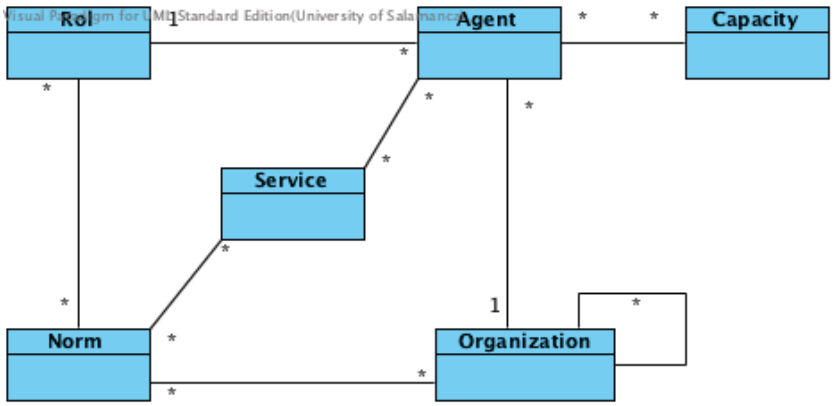


Figure 1: Principal classes of the system

- Programmed in Java and easily extensible.
- Possibility of having agents in various platforms (Windows, Linux, MaccOS, Android and IOS)
- Interface to oversee the organizations.

Figure 1 displays the principal entities of the system, and illustrates how the roles, norms and the organizations themselves are classes that facilitate the inclusion of organizational aspects. The services are also included as entities completely separate from the agent, facilitating their flexibility and adaption.

When launching the main container of execution, the communication system is initiated; the agent platform then automatically provides the following agents to facilitate the control of the organization:

- OrganizationManager: the agent responsible for the actual management of organizations and suborganizations. It is responsible for verifying the entry and exit of agents, and for assigning roles. To carry out these tasks, it works with the OrganizationAgent, which is a specialized version of this agent.
- InformationAgent: the agent responsible for accessing the database containing all pertinent system information.
- ServiceAgent: the agent responsible for recording and controlling the operation of services offered by the agents.
- NormAgent: the agent that ensures compliance with all the refined norms in the organization.
- CommunicationAgent: the agent responsible for controlling communication among agents, and for recording the interaction between agents and organizations.
- Sniffer: manages the message history and filters information by controlling communication initiated by queries.

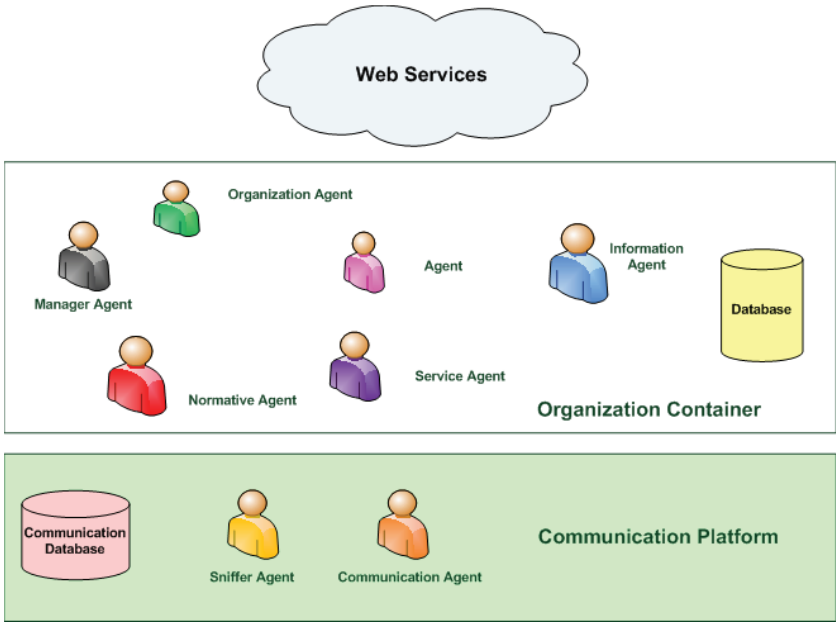


Figure 2: Architecture

The platform examines two modes of operation. In the first mode, the agents reside in the machine itself, while in the second mode the platform allows for the possibility of initiating all agents in different machines. The latter case has the disadvantage of allowing only minimal human intervention since it is necessary to previously specify the address of the machine where each of the agents are to reside; however it has the advantage of greater system distribution. We hope to create a service oriented platform that can take maximum advantage of the distribution of resources. To this end, all services are implemented web services. This makes it possible for the platform to include both a service provider agent and a consumer agent, thus emulating a client-server architecture. The provider agent knows how to contact the web service; once the client agent's request has been received, the provider agent extracts the required parameters and establishes the contact. Once received, the results are sent to the client agent. Each suborganization or work unit is automatically provided with an OrganizationAgent by the platform during the creation of the suborganization. This OrganizationAgent is similar to the OrganizationManager, but is only responsible for controlling the suborganizationn, and can communicate with the OrganizationManager if needed. If another suborganization is created hierarchically within the previous suborganization, it will include a separate OrganizationAgent that communicates with the OrganizationAgent from the parent organization. These agents are distributed hierarchically in order to free the OrganizationManager of tasks. This allows each OrganizationAgent to be responsible for a suborganization although, to a certain extent, the OrganizationManager can always access information from all of the organizations. Each agent belongs to one suborganization and can only communicate with the Or-

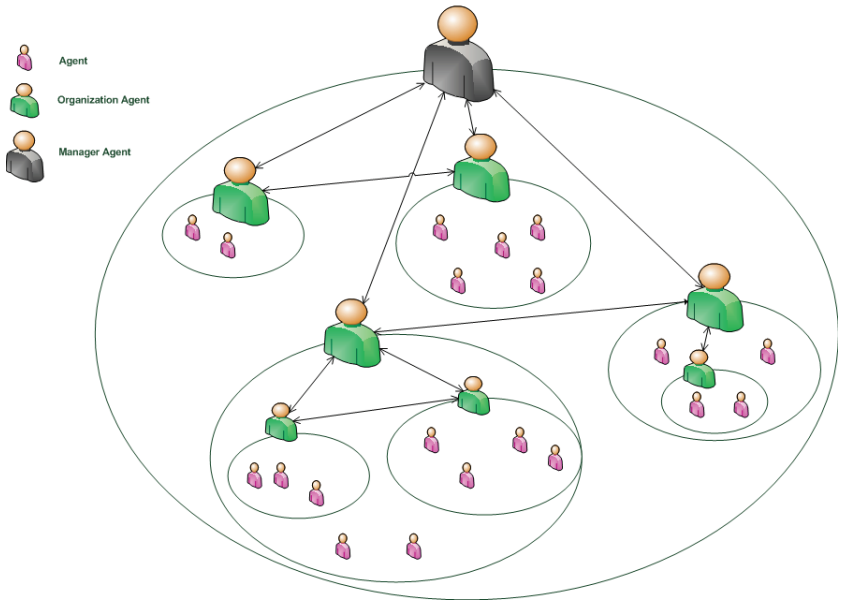


Figure 3: OrganizationManager and OrganizationAgents

ganizationAgent from its own organization; this makes it possible to include large suborganizational structures without overloading the AgentManager. All of the OrganizationAgents from the same level can communicate with each other, unless a specific standard is created to prevent this.

3.1 Communication Platform

This section will focus on describing the communication platform and protocol. As observed in Figure 2, the communication platform includes two main agents: the CommunciationAgent and the Sniffer. The first is in charge of checking the connections to confirm that the agents are online and see which ones have disconnected. It is also in continual communication with the NormAgent to ensure that the agents respect the lines of communication and comply with the standards. The Sniffer is in charge of recording all communication, offers services so that other agents can obtain history information, and facilitates the control of information flow for programmers and users. The IRC protocol was used to implement communication. Internet Relay Chat (IRC) is a real time internet protocol for simultaneous text messaging or conferencing. This protocol is regulated by 5 standards: RFC1459 [26], RFC2810 [19], RFC2811 [20], RFC2812 [21] y RFC2813 [22]. It is designed primarily for group conversations in discussion forums and channel calls, but also allows private messaging for one on one communications, and data transfers, including file exchanges [26]. The protocol in the OSI model is located on the application layer and uses TCP or alternatively TLS [19]. An IRC server can connect with other IRC servers to expand the user network. Users access the IRC networks by connecting a client to a server. There have been many implementations of clients,

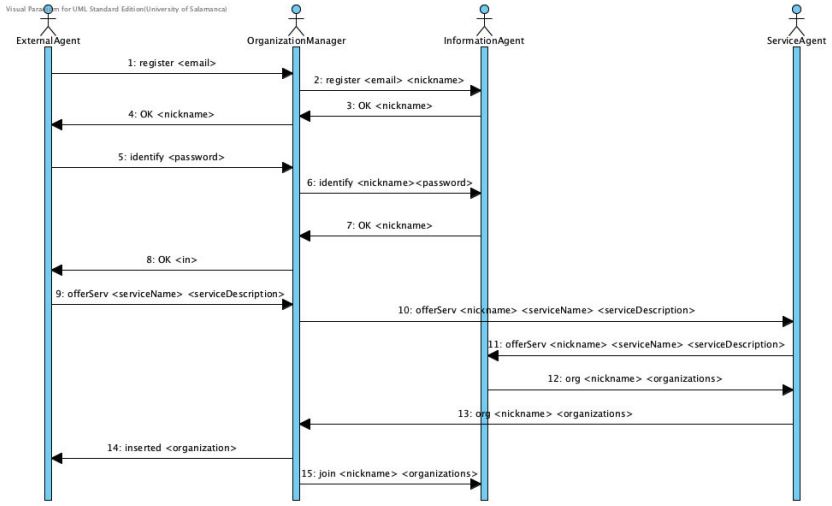


Figure 4: Sequence of steps for an agent to enter an organization

including mIRC or XChat. The original protocol is based on flat text (although it was subsequently expanded), and used TCP port 6667 as its primary port, or other nearby ports (for example TCP ports 6660-6669, 7000) [21]. The standard structure for an IRC server network is a tree configuration. The messages are routed only through those nodes that are strictly necessary; however, the network status is sent to all servers. When a message must be sent to multiple recipients, it is sent similar to a multidiffusion; that is, each message is sent to a network link only once [19] This is a strong point in its favor compared to the no-multicast protocols such as SimpleMail Transfer Protocol (SMTP) or the Extensible Messaging and Presence Protocol (XMPP). One of the most important features that characterize the platform is the use of the IRC protocol for communication among agents. This allows for the use of a protocol that is easy to implement, flexible and robust. The open standard protocol enables its continuous evolution. There are also IRC clients for all operating systems, including mobile devices. All messages include the following format: prefix command command-parameters \r \n. The prefix may be optional in some messages, and required only for entering messages; the command is one of the originals from the IRC standard. The following diagram illustrates the message flow required for an agent to enter an organization. These messages use the command PRIVMSG followed by the parameters indicated by the arrows in the diagram.

4 Case Study and Results

The platform we have developed can create a general type of organization, and includes the possibility of creating open and highly dynamic systems. In order to test the architecture, a case study was prepared to simulate a working environment. Four organizations were created to simulate four different departments

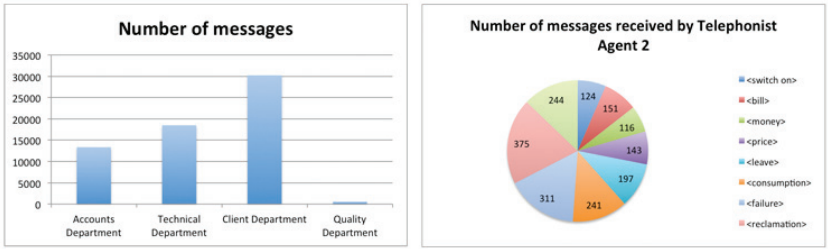


Figure 5: Messages Delivered

within a company: accounting (composed of 4 accounting agents, one manager and 2 secretaries); quality control (composed of 2 evaluating agents and two training specialist agents); technical services (composed of 6 technical agents); and customer service (composed of 8 telephonist agents). According to the role of each agent, there are specific services offered that allow them to resolve the queries they receive. In one possible case, the client agent contacts the telephonist agent, which simply receives the requests and redirects it to the agent qualified to resolve the request. The telephonist agent extracts the key words from the message sent by the client and contacts the Services Agent to determine which agent can address the required service. If the message contains the keyword "invoice", the query will be handled by the Accounting agent; if the keyword is "switch on" it will be handled by the Technical agent. Once the client is in contact with the appropriate agent, the agent can communicate with other agents in its organization to carry out the task. Four 30-minute simulations were performed with 20 different types of requests randomly provided. Studying the Evaluation and Sniffer agents it was possible to study how both the simulation and message flow unfolded. Focusing specifically on the Sniffer, it is possible obtain summary charts and diagrams, and specific numbers. Once the query is made, the Sniffer consults the database, filters the data and returns a URL that displays the desired data. It is possible to obtain the number of each type of message that a specific agent has received. Each message includes a tag that identifies the type of message, which makes it possible to filter information.

It is also possible to obtain a diagram of messages according to organization instead of agents. Using the message identifier, it is also possible to see which agents processed a given request; using the Evaluation agents we can determine the number of requests processed by each agent. We can conclude that the architecture we are developing has great potential to create open systems, and more specifically, virtual agent organizations. This architecture includes various tools that make it easy for the end user to create, manage and control these systems. One of the greatest advantages of this system is the communication platform that, by using the IRC standard, offers a robust and widely tested system that can handle a large number of connections, and that additionally facilitates the implementation for other potential extensions. Furthermore, the use of the Communication and Sniffer agents, offers services that can be easily invoked to study and extract message information.

5 Acknowledgements

This project has been supported by the Spanish CDTI. Proyecto de Cooperacin Interempresas. IDI-20110343, IDI-20110344, IDI-20110345, and the MICINN TIN 2009-13839-C03-03 project. Project supported by FEDER funds

References

[1] J. 20. Hbner. J -moise+ programming organisational agents with moise+ & jason. *Technical Fora Group at EUMAS'07*, 2007.

[2] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa-compliant agent framework. *Proceedings of the Practical Applications of Intelligent Agents*, 1999.

[3] H. J. F. Bordini, R. H. and R. Vieira. Jason and the golden fleece of agent-oriented programming. *Multi-Agent Programming: Languages, Platforms and Applications. Springer-Verlag. chapter 1, 3-37.*, 2005.

[4] H. J. F. Bordini, R. H. and M. Wooldridge. Programming multi-agent systems in agentspeak using jason. *John Wiley & Sons, Ltd.*, 2007.

[5] L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A short overview. *Proccedings Main Conference Net.ObjectDays 2004, pp. 195-207*, 2004.

[6] R. R. H. A. Busetta, P. and A. Lucas. Jack intelligent agents - components for intelligent agents in java. *Technical report, Agent Oriented Software Pty. Ltd, Melbourne, Australia*, 1998.

[7] J. K. Dale, Jonathan and F. Labo. April agent platform. *http://designstudio.lookin.at/research/relate%20survey /Survey%20Agent%20Platform/April%20Agent%20Platform.htm*, (accessed 11/29/2011).

[8] Emorphia. Fipa-os. *http://fipa-os.sourceforge.net*.

[9] F. M. Ferber, O. Gutknecht. From agents to organizations: an organizational view of multi-agent systems. *Agent-Oriented Software Engineering VI, Vol. LNCS 2935 of Lecture Notes in Computer Science, Springer-Verlag: 214-230*, 2004.

[10] T. Finin and Y. Labrou. Kqml as an agent communication language. *Bradshaw (ed.), Software Agents, pp. 291-316. Cambridge, MA*, 1997.

[11] T. F. for Intelligent Physical Agents. Fipa standar status specification. *http://www.fipa.org/repository/standardspecs.html*, (accessed 11/29/2011).

[12] K. C. Foster, I. and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl, vol. 15, no. 3: 200–222*, 2001.

[13] S. Galland. Janus: Another yet general-purpose multiagent platform. *Seventh AOSE Technical Forum, Paris*, 2010.

[14] G. S. H. V. Gaud, N. and A. Koukam. An organizational platform for holonic and multiagent systems. *Proccedings of Sixth International Workshop on Programming Multi-Agent Systems (ProMAS'08), of the Seventh International Conference on Autonomous agents and Multiagent Systems (AAMAS). E. Hindriks, A. Pokahr and S. Sardina (Eds.), pp. 111-126.*, 2008.

[15] A. Giret. An open architecture for service-oriented virtual organizations. *Programming Multi-Agent Systems: 7th International Workshop, ProMAS'09*, 2009.

[16] O. Gutknecht and J. Ferber. Madkit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems. *Technical Report R.R.LIRMM 9718, LIRM*, 1997.

[17] B. R. P. G. Hbner, J.F. Using jason and moise to develop a team of cowboys. *Hindriks, K., Pokahr, A., Sardina, S. (eds.) Proceedings of the Seventh International Workshop on Programming Multi-Agent Systems (ProMAS 08), Agent Contest, held with The Seventh International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2008), LNAI, vol. 5442, pp. 238-242. Springer, Heidelberg*, 2009.

[18] J. S. S. Jomi Fred Hubner and O. Boissier. A model for the structural, functional, and deontic specification of organizations in multiagent systems. *Guilherme Bittencourt and Geber L. Ramalho, editors, Proceedings of the 16th Brazilian Symposium on Artificial Intelligence (SBIA'02), volume 2507 of LNAI, pages 118-128, Berlin. Springer*, 2002.

[19] C. Kalt. Internet relay chat: Architecture. *RFC 2811*, 2000.

[20] C. Kalt. Internet relay chat: Channel management. *RFC 2811*, 2000.

[21] C. Kalt. Internet relay chat: Client protocol. *RFC 2812*, 2000.

[22] C. Kalt. Internet relay chat: Server protocol. *RFC 2813*, 2000.

[23] A. O. S. P. Ltd. Jack intelligent agents teams manual. s.l. : Agent oriented software pty ltd. 2005.

[24] F. G. McCabe and K. L. Clark. April-agent process interaction language. *Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents (ECAI-94), Michael J. Wooldridge and Nicholas R. Jennings (Eds.).*, 1994.

[25] P. D. O'Brien and R. C. Nicol. Fipa towards a standard for software agents. *BT Technology Journal, vol. 13, issue 3, pp. 51-59. Springer Netherlands*, 1998.

[26] J. Oikarinen and D. Reed. Internet relay chat protocol. *RFC 1459*, 1993.

[27] A. S. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. *W. Van de Velde and J. Perram, editors, Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), no. 1038 in LNAI, pp. 42-55, London*, 1996.

[28] G. M. P. Rao, A. S. Modeling rational agents within a bdi-architecture. *J. Allen, R. Fikes, E. Sandewall (Ed.), Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91) (pp. 473-484). San Mateo, CA, USA: Morgan Kaufmann publishers, Inc*, 1991.

[29] SACI. Saci simple agent communication instrastructure. *http://www.lti.pcs.usp.br/saci/*.

A Tool for Retrieving Meaningful Privacy Information from Social Networks

Ricard L. FOGUÉS[1], Jose M. SUCH[1], Agustin ESPINOSA[1], and Ana GARCIA-FORNES[1]

[1] *Departament de Sistemes Informàtics i Computació, Universitat Politècnica de València,* {rilopez,jsuch,aespinos,agarcia}@dsic.upv.es

Abstract

The use of social networking services (SNSs) such as Facebook, Flickr, or MySpace has grown explosively in the last few years. People see these SNSs as useful tools to find friends and interact with them. SNSs allow their users to share photos, videos, and express their thoughts and feelings. Even though users enjoy the capabilities that these SNSs offer, they have became aware of privacy issues. The public image of a subject can be affected by photos or comments posted on a social network. Therefore it is important for SNS users to control what others can see in their profile. Recent studies demonstrate that users are demanding better mechanisms to protect their privacy. An appropriate approximation to solve this problem is a tool that automatically suggests a privacy policy for any item shared on a SNS. The first step for any mechanism to recommend and predict privacy policies is to retrieve meaningful privacy information from the SNS, such user communities and the relationships of them. Most SNSs rely on groups to help users specify their privacy policies. Therefore, a basic functionality of such a mechanism is to group the user's friends automatically. Although SNSs treat all of the friends of a user the same, without taking into account different degrees of the friendship, this is not a realistic approach. Hence, another factor to consider when defining a privacy policy is the type of relationship between the owner of the item being shared and its potential viewers. In this work, we present a tool called Best Friend Forever (BFF) that automatically classifies the friends of a user in communities and assigns a value to the strength of the relationship ties to each one. We also explain the characteristics of BFF and show the results of an experimental evaluation.

Keywords: Information retrieval, social network, social media, privacy, tie strength.

1 Introduction

Social networking services (SNSs) are currently the services that are most more demanded by users worldwide. Facebook (with more than 800 million active users[1]) and Flickr (with 51 million registered members[2]) are two of the most successful

SNSs. People register to these SNSs and share images, videos, and thoughts because they perceive a great payoff in terms of friendship, jobs, and other opportunities [6]. However, the huge number of items uploaded to these SNSs and the persistence of these items in the social networks have the potential to threaten the privacy of their users[13]. For example, employers are becoming accustomed to checking the profile of the candidates in popular SNSs. If the privacy of the profile of a candidate is not properly set, what an employer sees in that candidate's profile may affect the employer's decision. It might even be possible for a stalker to infer the address of a person by looking at that person's photos posted in a social network.

Recent studies show that SNS users' awareness of privacy issues has increased lately [3]. To cope with these privacy threats users tend to adjust and modify the default privacy settings set up by the SNS since they feel that these default settings are not enough to protect them. Nonetheless, the current privacy setting mechanisms offered by the SNS seem difficult or confusing for users [16, 22]. For example, Facebook offers five privacy levels for each element shared: "public", "friends", "only me", "personalized", or "groups". While these five levels may seem sufficient, they require a certain amount of work by the user before they can be applied. In other words, groups have to be built in advance by the user. If we consider that the average number of friends in Facebook is 130 [1], classifying all of them into groups can represent a serious challenge. Furthermore, once groups are defined, if the user decides to exclude specific users of a group from seeing the shared content, he/she has to specify the excluded persons one by one.

Another problem users find when defining privacy policies is that most of the SNSs base their privacy models entirely on groups. Every friend of a user is the same; close friends, family or mere acquaintances are not distinguished. As Wiese et al demonstrated in [24] the willingness of users to share in social networks is dependent on the closeness (tie strength) of relationships. The works of Gilbert et al. [10, 9] and Xiang et al.[25] showed that it is possible to predict the strength of the relationship ties with the information available at the SNSs. As these related works prove, in order to suggest good privacy policies for SNS users the strength of the relationship ties must be taken into account.

These complications and obstacles may lead users to have privacy policies that do not fit their preferences. Another effect of not using properly adapted privacy policies is that the users feel as though they have lost control of their information and how it is shared among the SNS. Users both desire and need more tools to allow them to regain control over their privacy. Thus, in the long term, our aim is to develop autonomous agents that would help users define their privacy policies by automatically recommending them privacy policies that are appropriate.

A first step in creating appropriate privacy policies is to gather information about how the user interacts with others in the social network. In other words, we need to know more details about the social network connections of the user. In this work, we introduce Best Friend Forever (BFF), which is a tool that automatically obtains friend groups and a value for the strength the relationship ties. Moreover, it provides support so that the user can refine the results. The tool has been implemented as a Facebook application and is publicly available at `gti-ia.dsic.upv.es/bff`. The main objective of BFF is to offer users enough information to modify and adapt their privacy policies on Facebook. Our experimental results

suggest that our software accurately predicts user groups and tie strength values, and also requires little user intervention.

The rest of the paper is organized as follows. Section 2 introduces preliminary notions of tie strength and community finding algorithms in social networks. Section 3 presents an overview of BFF and its different elements. Section 4 reports the results of the experimental evaluation. Section 5 discusses some related works. Finally, Section 6 concludes the paper and outlines future research directions.

2 Background

One of the main features of our tool is that it is able to predict a value for the tie strength of each of the social connections of the user. Current SNSs have made little effort to differentiate users. Users are either friends or strangers, with nothing in between. This approach does not properly represent human relationships. As introduced in the paper of Granovetter[12], the concept of *tie strength* defines the relationship between two individuals. In his work, Granovetter describes two different types of ties: *strong* and *weak*. On the one hand, strong ties usually include relationships such as family and close friends. On the other hand, weak ties may refer to coworkers or less trusted friends. Granovetter defined four tie strength dimensions: duration, intimacy, intensity and reciprocal services. Later works proposed three additional dimensions[4, 23, 15]. These three additional dimensions are: (i) structural, which refers to factors like social circles, (ii) social distance, which refer to factors like political affiliation or education level, and (iii) emotional support, which embodies elements such as offering advice.

In Wiese's work [24], the authors find a high correlation between the willingness to share information and tie strength. Their research proved that the strength of a tie is even more significant than grouping for predicting sharing. They suggest that a mixture of grouping and tie strength might provide richer sharing policies. Their conclusions support our thesis that tie strength is an important variable when deciding who is able to access a given information. Therefore, it is necessary to know the tie strength in order to suggest adequate privacy policies.

Automatic friend grouping is the other main feature of our software. Many SNSs offer grouping features. However, they are not automatic and the users need to take an effort and group all their contacts. If we consider that the average number of contacts in Facebook is 130, this can represent a time-consuming task. Friend groups become useful when defining privacy policies. It is easier for a human user to assign the same access privileges to a group of friends than specifying a privacy policy separately for each persons in the group.

In order to group persons we use *communities* [11]. Communities are usually defined as natural divisions of network nodes into densely connected subgroups. In our context, the nodes are the contacts or friends of a given participant, and the connections between the nodes are friend relationships. There are many community finding algorithms [8]. In this work we use the the hierarchical diffusion algorithm proposed by Shen et al. in [20]. Section 3.2 introduces some details of this algorithm.

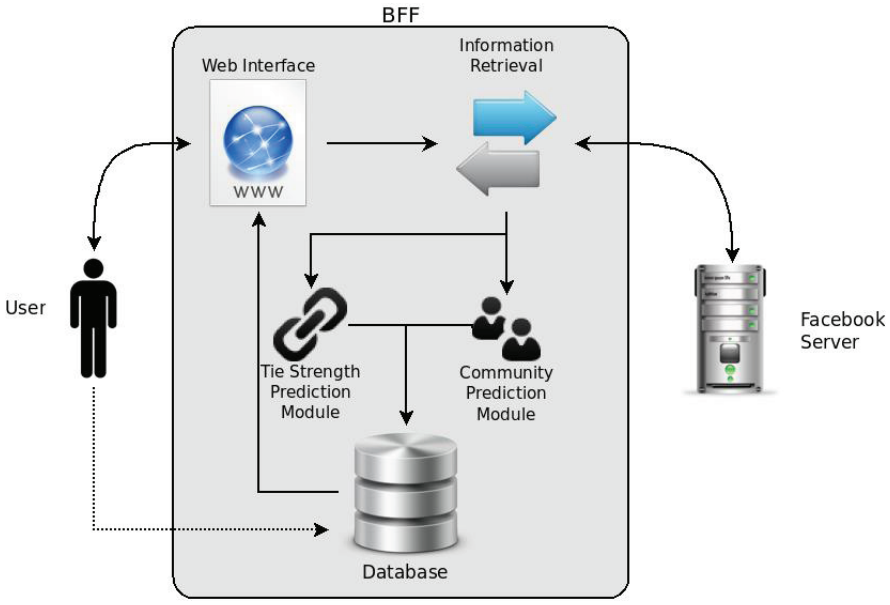


Figure 1: BFF Overview

3 Best Friend Forever

This section introduces our tool and gives a complete overview of it. BFF aims to retrieve information from the social network of a participant in order to help to automatically recommend privacy policies. Specifically, the data needed is tie strength and friend groups. BFF is written in PHP and Javascript and is publicly accessible. Due to our experimentation needs, BFF is currently working as a web page; however, in the future, we plan to distribute BFF as a software program that users can execute in their own computers or on a trusted web server in order to preserve their privacy.

BFF is composed of two modules: (i) community prediction, and (ii) tie strength prediction. The community prediction module is in charge of create chunks of users from the participant's contacts. The tie strength prediction module establishes a value of tie strength to each one of the participant's friends. In general, the input of BFF is the Facebook account of the participant, and the output is a set of user groups and a value of tie strength for each one of those users.

Figure 1 shows an overview of BFF and how it works. The interface between BFF and the user is a web page. BFF collects information from the user's Facebook account. Therefore, before users can use BFF, they have to login to Facebook and give permission to BFF to access their Facebook information. Once the permission is given, BFF requests information from the Facebook server. When all the necessary information has been collected, the information is passed to the community prediction module and to the tie strength prediction module. These modules predict a set of groups and tie strength values for the friends of the user. The predictions are shown to the user as a suggestion. The dotted line represents the

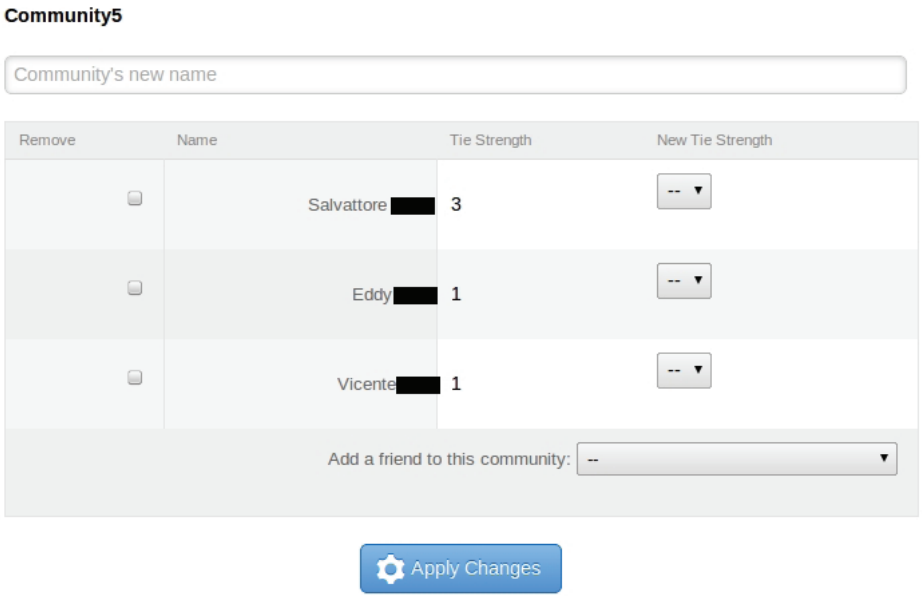


Figure 2: Result Sample

possibility of the user to modify and adapt the suggestions created by the two modules. These modifications are stored in the database for future reference.

BFF aims to be a tool that can be used by real users of Facebook. Therefore, it has to work as fast as possible. BFF allows the users to configure the amount of time they want the whole process to take. The user can choose among three options *fast*, *normal*, and *thorough*. The amount of information collected depends on the configuration chosen by the user. The fast configuration only collects the user's most recent information on Facebook, normal configuration collects some old information, and thorough configuration collects every available information. Clearly, a faster process will be less accurate and more inclined to prediction errors than a thorough process where more information is collected.

Figure 2 shows the screen where the results of BFF are presented to the user. In this example, the figure only depicts one of the communities automatically created by BFF. Part of the name of the members of the community has been hidden to preserve their privacy. As shown in Figure 2, the members of the community are sorted by their tie strength value. The participant can change the name of the community, remove members from the community, add new ones, and change the tie strength value for any member in that community.

3.1 Tie Strength Prediction Module

As stated in the introduction, BFF predicts the tie strength of the relationships of the participant with each person that is socially connected to her. We model tie strength as a linear combination of predictive variables. During the creation of BFF the usability of our tool was a key factor. With this in mind, BFF has to be

capable of predicting the tie strength accurately in a reasonable amount of time, and every user should be able to get an accurate prediction. These two requisites (time cost and generalization) conditioned the selection of predictive variables. Next paragraph explains the selected predictive variables.

The variable *Days since last communication* measures the recency of the communication. *Days since first communication* is an approximation of the duration of the friendship. *Wall messages* counts the number of messages exchanged using the wall. *Photos together* counts the photos where both persons (participant and friend) are tagged. *Links shared* counts the number web page links traded between the friend and the participant. *Initiated wall posts* counts the number of publications posted by the friend on the participant's wall. *"Likes"* counts the number of likes given by the friend to the participant's publications. *Inbox messages exchanged* counts the number of private messages traded between both persons. *Number of friends* is the total number of friends of the friend. *Educational difference* measures the difference in a numeric scale: none = 0, high school = 1, university = 2, PhD = 4. Finally, *the mean strength of mutual friends* is also taken into account, and it captures the idea of how a relationship is modified by the tie strength of mutual friends.

The selected predictive variables are based on the variables proposed in [10]. In their work, the authors propose a set of 72 predictive variables. The authors did not consider the cost of collecting the variables and their generalization, they only considered the predictive capabilities of the variables. As stated before, two requisites for the predictive variables are their collecting cost and their generalization. With regard to time cost, BFF collects the information from Facebook; each time BFF needs to ask Facebook for an item, it has to send an HTTP request to Facebook. This operation may take a few seconds; therefore, collecting many variables from a very active participant account can take a long time. BFF had to restrict the number of predictive variables. In the matter of generalization, BFF tie strength prediction cannot depend on variables that require the participant to have specific characteristics. For example, language dependent variables are inappropriate as they would limit the different users that would be able to use BFF. The ten selected predictive variables for BFF satisfy both requisites, they can be collected fast and are valid for any user. Moreover, the selected predictive variables cover every tie strength dimension. Table 1 shows the tie strength dimensions and the predictive variables that belong to each dimension.

The equation below represents the tie strength s_i of the i^{th} friend. R_i stands for the vector of ten predictive variables of the i^{th} friend. μ_M is the mean strength of mutual friends between the user and the i^{th} friend. Finally, β is the vector of weights applied to the predictive variables and γ is the weight applied to the mean strength of mutual friends. In order to set the weight of each variable we used the findings of [10] as we wanted to avoid the use of a model that completely lacked information on the relative importance of each variable to predict tie strength. As a future work, we plan to perform a fine tunning of the weights of the variables.

$$s_i = \beta R_i + \gamma \mu_M$$

$$M = \{s_j : j \text{ and } i \text{ are mutual friends}\}$$

Dimension	Variables
Intimacy	Number of days since first communication. Number of friends.
Intensity	Wall messages. Initiated wall post. Inbox messages exchanged.
Duration	Days since first communication.
Social distance	Educational difference.
Reciprocal services	Links shared
Emotional support	Likes
Structural dimension	Mean strength of mutual friends.

Table 1: Predictive variables and tie strength dimensions

After collecting the predictive variables for the friends of the user, the variables are normalized. Then, the tie strength is calculated for each user. The results are normalized to a numeric scale 1-5, where 1 represents that both persons are very distant (mere acquaintance) and 5 that they are very close. The results are presented graphically, as shown in Figure 2, so that users are sorted by group and by tie strength. It is easier to figure out the value of the tie strength of a person by comparing that relationship to others. As in the grouping step (explained below), the participant can refine the results of the tie strength calculation.

3.2 Community Prediction Module

The community prediction module is based in the hierarchical diffusion algorithm proposed by Shen et al. in [20]. The algorithm is founded on the triadic closure principle, which suggests that, in a social network, there is an increased likelihood that two people will become friends if they have friends in common. The algorithm is divided into two steps: (i) the thresholding step, and (ii) the diffusion step. In the first step, the core members of the communities are chosen. These members are those users that are highly connected to others. Once the core members have been selected, the diffusion step performs a cascade joining, and new members are added to the communities formed by the core members. The diffusion step follows a direct-benefit model in networks, which states that people benefit from directly copying others' decisions. In their paper, the authors did not test their algorithm on a network of a SNS like Facebook. However, according to the results of our experimental evaluation, it performs very well in this environment. Moreover, this algorithm has great performance in terms of computational cost for the average size of Facebook communities.

When the participant uses our software, the community prediction module queries Facebook about the friends of the participant and the friends of those friends (mutual friends) in order to build the graph that will be the input of the algorithm. The community prediction module suggests the community division calculated by the algorithm. The participant can accept the groups proposed or modify them at will.

4 Experimental Evaluation

The goal of our experimental study is to evaluate the accuracy of our BFF tool in terms of community and tie strength prediction. Specifically, we want to answer the following questions:

- How effective is the community module in grouping the contacts of a user?
- How accurate are the predictions of the tie strength module?
- Do users perceive that BFF is a good tool in general? In other words, do they think that BFF is capable of inferring accurate information from their available data on Facebook?

To answer these questions, we performed an experimental evaluation with Facebook users. Our results indicate that BFF is an effective tool. Furthermore, users considered BFF to be a good tool and valued it positively. In what follows, we first introduce the experimental settings and then report our findings.

4.1 Participants

Our 17 participants were mostly students and members of the Polytechnic University of Valencia. The sample consisted of 4 women (23.5%) and 13 men (76.5%) . The minimum number of Facebook friends was 58; the maximum was 529 (mean of 186.94). In total, we analyzed 3178 friend relationships. All of the participants used Facebook regularly.

4.2 Method

The participants in our experiment had to try BFF and evaluate its performance. BFF was created to ensure that its use would be easy for anyone. The participants only had to access to the web page of BFF, log in with their Facebook account, and start the application. During the experimental evaluation, the time configuration was deactivated since we wanted all of the participants to evaluate BFF with the same configuration settings. The forced configuration was "normal", which on average takes 10 minutes to complete for a user with a number of friends of around 100.

After BFF completed its process, the participants were requested to correct any possible errors in tie strength prediction and in user grouping. Users could change the tie strength value of any contact, move users freely from one community to another, and create new communities. These possible corrections were stored in order to evaluate the performance of BFF.

Finally, the participants were requested to answer a short survey to find out their opinion about BFF. The survey was composed of the four following questions:

1. How well did BFF group your friends into communities?
2. How well did BFF predict the tie strength between you and your friends?
3. In general, how accurate do you think BFF is?

4. How accurate do you think BFF is considering it only accesses your information on Facebook? For example, if one of your friends on Facebook is your brother, but you have never interacted with him on Facebook, it is impossible for BFF to accurately predict the tie strength between you and your brother.

Each question was rated on a scale 1-5: 1 = very bad, and 5 = very good. The first and second question addressed specific parts of BFF (the grouping feature and the tie strength prediction respectively). The third and fourth questions were general questions. The intention of the fourth question was to clarify the limitations of BFF to the users. Currently, BFF is limited to the bounds of Facebook; therefore, it only considers the interactions and social connections that occur on Facebook. In future work, we expect to collect information from different sources than Facebook, so BFF will be able to avoid this limitation.

4.3 Results

With regard to tie strength prediction, the module performed very accurately. It achieved a Mean Absolute Error of 0.1155 on a discrete scale 1-5, where 1 is the weakest and 5 is the strongest. We chose to discretize[1] the tie strength in order to facilitate the understanding of the results to the users. Moreover, according to our findings, when the tie strength module predicted incorrectly, 51% of the time it overrated tie strength and 49% of the time it underrated the strength. This suggests that tie strength prediction is not biased.

The performance of the community prediction module was also very accurate; it achieved an accuracy of over 95%. The participants performed mainly two types of modifications on friend community predictions:

- The participant divided the largest community into several sub-communities. An interesting fact about this situation is that the moved contacts usually had a low tie strength (2.5 average). Thus, tie strength may affect the way a user groups his/her friends. An idea for future research is to determine how tie strength may play a role in the community prediction.

- The second more common modification was the user combining communities formed by only one or two members with low tie strength into a larger community. These new communities can be identified as communities of acquaintances. It seems that participants preferred to manage these contacts as a single group, even when they did not share anything but the fact that they had few friends in common and a low tie strength value.

Another important factor to analyze was the number of corrections that the participants needed to make to the suggestions. As stated previously, SNS users struggle to set up privacy settings. If the aim of BFF is to lighten the burden of this task, its suggestions cannot contain a huge number of errors that need correction.

[1]The discretization process might have caused a higher prediction error. For example, a user with a tie strength of 3.6 and another with a strength of 4.4 will be both assigned a strength of 4 during the discretization process. As future work, we plan to study the effect of discretization in the prediction error, so that we could achieve a trade-off between the understandability of the results and the error introduced because of the discretization.

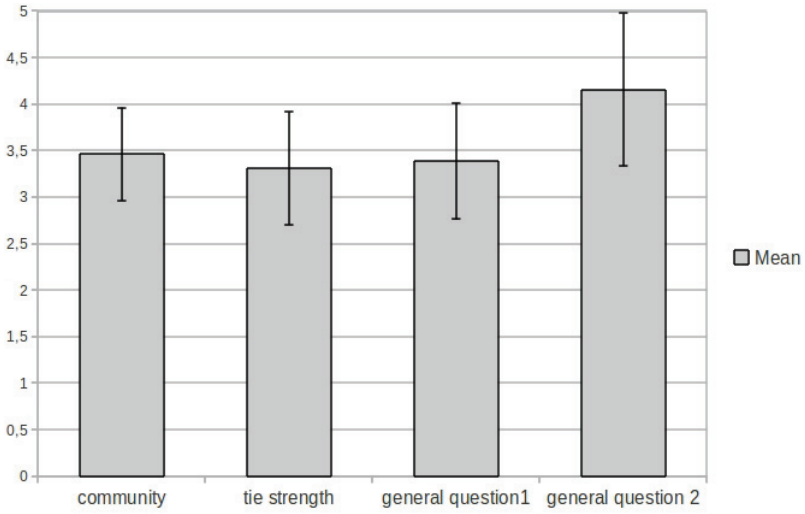


Figure 3: Mean and standard deviation for the survey questions

The mean number of corrections made was 19.3 per participant. Specifically, the participants made an average of 13.12 tie strength corrections and an average of 6.2 community corrections. Considering that the average number of friends of our participants was 186.94, having to perform only 19.3 changes could speed up the process of organizing friends before setting privacy policies.

The participants also rated the performance of BFF by answering a short survey. The results show that the participants rated BFF performance positively. The participants perceived a slightly better accuracy in community prediction than in tie strength. This shows that tie strength prediction is a more complex task due to the high number of variables that the model considers. Another result to note is that the participants rated the second general question (question 4) higher than the first general one (question 3). When answering the first general question, the participants did not consider the limitations of BFF. Therefore, even when almost every friend was rated correctly, they detected mistakes. Due to the brief explanation in the second question about how BFF works, the participants realized that BFF is limited by the bounds of Facebook, and, for example, that it cannot predict the tie strength of a relationship that mainly occurs outside Facebook. When the participants became aware of the limitations of BFF, they took into account how they interacted with others on Facebook in order make their judgments. This explains the better rating for the second general question.

5 Related Work

Recent works have proposed models to predict tie strength. Gilbert et al.[10] proposed a model, based on Granovetter's work, that predicted tie strength among the users of Facebook. The authors identified a set of 74 predictive variables that can be found on Facebook. They achieved an accuracy of 84%. Another work that

predicts tie strength of social links is [14]. Like in the work of Gilbert, the authors define a set of 50 predictive variables. In this work the authors aim to discriminate strong links from weak links. However, they do not consider a scale in the strength of the link, they are either strong or weak. These two works use a supervised learning model that needs human intervention to work properly. Aiming at the same objective, Xiang et al.[25] proposed a model to infer relationship strength based on profile similarity and interaction activity, with the goal of automatically distinguishing strong relationships from weak ones. It is worth noting that this model relies on an unsupervised learning method, but it lacks a empirical evaluation with real users. All three works show that it is possible to infer tie strength from the available personal data in a SNS. These three works differ from ours in that they aim to create models to predict tie strength from the information available on a SNS. However, they do not offer tools that social network users can use to help them to form friend groups and set privacy policies. Moreover, they only consider the predictive capabilities of the variables chosen for their models, but they do not take into account factors like the computational cost of collecting these variables, which is an important factor when creating a usable tool.

The other main feature of BFF is that it suggests friend groups to the participant user. The main idea is that with the grouping and tie strength information the user has enough elements to create appropriate privacy policies. The work of Fang and coworkers [7] proposes a tool that suggests privacy policies for certain elements of a Facebook user profile. This work bases the privacy suggestions in grouping user's contacts in contexts. Every contact in the same context is granted the same access permissions. The authors present a tool called Privacy Wizard that helps user to set the privacy policies to protect user's traits, like birth date, address, and telephone number. However, this work does not consider tie strength, and as the authors proved in [24], it is a key variable to consider when determining the disclosure degree of the elements being shared in a social network.

Other works present mechanisms that can partially infer users' social network and its characteristics from sources of information different than SNSs. In [5] the authors propose a method that extracts a social network for a user given her mailbox and the information available on Internet. A similar approach is presented in [18]. In this work the authors present POLYPHONET. From a given set of persons, the authors find the social connections among them by querying to Google. The authors estimate the strength of the relationship between two persons by co-occurrences of their two names. These two works differ from ours in that they do not rely in a SNS to extract social information from users. However, this approach also has limitations. Relying on information sources that do not necessarily contain social relevant information may lead to errors. For example, two persons may appear in several web pages together but do not have any social link. In order to avoid this problem, both works ([18, 5]) require a predefined set of persons that will form the social community. In contrast, relying only on Facebook data guarantees that the social links will actually exist, but may also lead to errors. Even when the connection truly exists, the interactions between two persons may occur outside Facebook. Therefore, the strength of such link will be incorrectly predicted by our software. In the future, we plan to expand the search of variables for defining the groups and the tie strength with information

that can be found outside the social network, like the information available in the participant's mailbox or in the personal web page of a user of the social network.

The work of Murukannaiah and Singh [19] presents Platys Social. The authors developed a software that runs on a mobile device. This software learns a user's social circles and the priority of the user's social connections from daily interactions. The software infers the interactions from information that is available on mobile devices, such as wi-fi networks, bluetooth connections, phone calls, and text messages. The work of Murukannaiah and Singh presents a new approach for extracting social information from the real world, and not only from Internet. Their work and ours could be merged so that tie strength could be computed taking into account day by day encounter frequency and the information stored on a SNS like Facebook.

6 Conclusions and Future Work

In this paper, we have presented a new tool for social network mining. This tool is our first attempt to build a software that can help users to better understand their social relationships on a SNS like Facebook. Currently, BFF is focused on community and tie strength prediction. However, in the long term, we plan to expand it with new functionalities and features. The modular architecture of BFF allows us to develop new modules that can be easily added to BFF. These new modules will rely on the capability of BFF to properly predict tie strength and user communities. In order to be confident in the current capabilities of BFF, we evaluated it using real-world data from real users of Facebook. BFF achieved a Mean Absolute Error of 0.1155 for predicting tie strength and an accuracy of 95% in friend grouping. Furthermore, on average, participants only needed to perform 19.3 corrections to BFF suggestions, taking into account that the average number of friends of the participants was 186.94, BFF can positively accelerate the process of organizing friends. Finally, users considered that BFF was good at predicting tie strength and groups, and they considered it to be a good tool overall.

Many research paths open from here. The first one, and the motivation of this work, is to use the extracted information to predict privacy policies. Users limit what they share and with whom depending on the type of the relationship. Therefore, a tool that correctly infers the types of relationships may be able to predict suitable privacy policies. Furthermore, the ability of BFF to create groups of users also matches the functionality of many SNSs that offer the possibility for users to group their friends. Using these two features, we can create a new functionality for privacy policy recommendation. Users perceive the utility of SNSs by sharing photographs, videos, and other items with their contacts. However, privacy issues can stop users from fully enjoying the functionalities of a SNS. By automating the process of privacy policy definition and how the information is disclosed on a SNS, we can reduce the burden that these systems impose on users, thus increasing their utility.

Another path for further research is to add the possibility for BFF to predict tie strength and friend communities using not only the information available at Facebook, but also using other environments for searching. As the works [5, 18, 19]

prove, social information can be extracted from several different environments. The information available at users' mailbox, personal web pages, Internet search engines could be collected by BFF. The development of a module that could be deployed on a mobile device would allow BFF to also consider daily user interactions. The addition of new sources of information will change how tie strength and grouping are predicted. For the tie strength model, new variables will have to be considered, so the weight of the variables may have to change. With regard to the community finding algorithm, connections outside Facebook may increase the weight of some edges, thus changing the selection of core members during the threshold step. Besides, it will be necessary to take into account how the addition of new variables can affect the efficiency of the tie strength and community predictions.

Apart from being of crucial importance for developing autonomous agents that recommend privacy policies to users, the information that our tool provides can also be the basis (or at least it can play a very important role) to solve many other problems. For instance, the tie strength among agents is used to obtain the optimal social trust path in complex social networks [17]. Moreover, agents could judge the outcome of a negotiation as being distributively fair based on the tie strength between them [21].

7 Acknowledgements

This work has been partially supported by CONSOLIDER-INGENIO 2010 under grant CSD2007-00022, and TIN 2008-04446 and PROMETEO/2008/051 projects. Ricard L. Fogués is working with a FPI grant from Programa de Ayudas de Investigación y Desarrollo (PAID) de la Universitat Politècnica de València.

References

[1] Facebook website. Facebook Statistics http://www.facebook.com.

[2] Yahoo advertising solutions. http://advertising.yahoo.com/article/flickr.html.

[3] D. Boyd and E. Hargittai. Facebook privacy settings: Who cares? *First Monday*, 15(8), 2010.

[4] R. Burt. *Structural holes: The social structure of competition*. Harvard Univ Pr, 1995.

[5] A. Culotta, R. Bekkerman, and A. McCallum. Extracting social networks and contact information from email and the web. 2004.

[6] N. Ellison, C. Steinfield, and C. Lampe. The benefits of facebook friends: social capital and college students use of online social network sites. *Journal of Computer-Mediated Communication*, 12(4):1143–1168, 2007.

[7] L. Fang and K. LeFevre. Privacy wizards for social networking sites. In *Proceedings of the 19th international conference on World wide web*, pages 351–360. ACM, 2010.

[8] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, 2010.

[9] E. Gilbert. Predicting tie strength in a new medium. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, CSCW '12, pages 1047–1056, New York, NY, USA, 2012. ACM.

[10] E. Gilbert and K. Karahalios. Predicting tie strength with social media. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 211–220. ACM, 2009.

[11] M. Girvan and M. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821, 2002.

[12] M. Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973.

[13] R. Gross and A. Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 71–80. ACM, 2005.

[14] I. Kahanda and J. Neville. Using transactional information to predict link strength in online social networks. In *Proceedings of the Third International Conference on Weblogs and Social Media (ICWSM)*, 2009.

[15] N. Lin, W. Ensel, and J. Vaughn. Social resources and strength of ties: Structural factors in occupational status attainment. *American sociological review*, pages 393–405, 1981.

[16] H. Lipford, A. Besmer, and J. Watson. Understanding privacy settings in facebook with an audience view. In *Proceedings of the 1st Conference on Usability, Psychology, and Security*, pages 1–8. USENIX Association Berkeley, CA, USA, 2008.

[17] G. Liu, Y. Wang, and M. Orgun. Optimal social trust path selection in complex social networks. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI*, pages 1391–1398, 2010.

[18] Y. Matsuo, J. Mori, M. Hamasaki, T. Nishimura, H. Takeda, K. Hasida, and M. Ishizuka. Polyphonet: An advanced social network extraction system from the web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(4):262 – 278, 2007. World Wide Web Conference 2006Semantic Web Track.

[19] P. Murukannaiah and M. Singh. Platys social: Relating shared places and private social circles. *Internet Computing, IEEE*, (99):1–1, 2011.

[20] K. Shen, L. Song, X. Yang, and W. Zhang. A hierarchical diffusion algorithm for community detection in social networks. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2010 International Conference on*, pages 276–283. IEEE, 2010.

[21] C. Sierra and J. Debenham. The LOGIC negotiation model. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8. ACM, 2007.

[22] K. Strater and H. Lipford. Strategies and struggles with privacy in an online social networking community. In *Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction-Volume 1*, pages 111–119. British Computer Society, 2008.

[23] B. Wellman and S. Wortley. Different strokes from different folks: Community ties and social support. *American journal of Sociology*, pages 558–588, 1990.

[24] J. Wiese, P. Kelley, L. Cranor, L. Dabbish, J. Hong, and J. Zimmerman. Are you close with me? are you nearby? investigating social groups, closeness, and willingness to share. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 197–206. ACM, 2011.

[25] R. Xiang, J. Neville, and M. Rogati. Modeling relationship strength in online social networks. In *Proceedings of the 19th international conference on World wide web*, pages 981–990. ACM, 2010.

Alpha Test-bed: A New Approach for Evaluating Trust Models

David Jelenc[1], Ramón Hermoso[2], Sascha Ossowski[2], and Denis Trček[1]

[1] *University of Ljubljana, Faculty for Computer and Information Science, Tržaška 25, Ljubljana, Slovenia,* {david.jelenc,denis.trcek}@fri.uni-lj.si

[2] *University Rey Juan Carlos, CETINIA, Calle Tulipán s/n, 28933 Móstoles, Madrid, Spain,* {ramon.hermoso,sascha.ossowski}@urjc.es

Abstract

During last years many trust models have been proposed in the literature. However, test-beds that evaluate how well trust models perform are scarce. Most of the approaches evaluate trust models by measuring utility gains of agents that use trust models. We argue that such measurements are ambiguous, because they actually measure the quality of the decisions that agents make and not the quality of the calculations that their respective trust models produce. In this paper we put forward a novel test-bed that directly measures the outputs of trust models instead of decision-making processes. We achieve this by evaluating the rankings of agents that we derive from a trust model's estimations. We provide two metrics, namely accuracy and coverage, to qualitatively assess the performance of trust models. We also provide a prototype implementation of the test-bed.

Keywords: trust model, test-bed, agent, multi-agent system.

1 Introduction

The concept of trust has become a cornerstone in many areas. Examples include multi-agent systems (MAS) [1], service-oriented architectures (SOA) [2], wireless-sensor networks (WSN) [3], P2P networks [4] and many others [5]. In all those areas participants, commonly referred to as agents, have some degree of autonomy, which allows them to freely select interaction partners. But such selections must be prudent if agents are to achieve desired goals. Trust models are proven to be helpful in such cases. Research so far has been focused in three areas, namely i) the design of trust models that estimate an agent's trust by aggregating its local experiences and obtained opinions, ii) the development of reputation mechanisms that present significant advances in the analysis and processing of opinions, and iii) design of test-beds that evaluate (the well-functioning of) trust models. While the first two have attracted most of the attention, we deal with the third in this paper.

We argue that most of the proposed test-beds evaluate trust models with two important drawbacks. First, *ad-hoc* implemented test-beds are often constrained to a certain domain, for which the originally-tested trust models were intended, and we cannot reuse them in other scenarios. The second problem

of the existing approaches lies in the evaluation procedure and in the metrics they use for evaluation. Their evaluation procedure measures an *utility* that an agent with a trust model gathers as a result of interactions. Their rationale is that the trust model guides the agent to select the best collaborators for interactions. However, we argue that calculated trust is not the only factor that influences the selection of partners – an agent usually considers various aspects of potential counterparts, and while trust is important, it is only one of many. Thus when we evaluate trust models by measuring the utility obtained in interactions, we are in effect evaluating not the validity of the calculated trust but the validity of the made decisions (that is, partner selections). Moreover, such evaluation is troublesome even if we disregard all other aspects and assume that partner selections are only influenced by trust. This is because such approach evaluates only a subset of a trust model's output – it validates only if the most trusted agent is actually the correct one. Trust estimations towards other agents are ignored and thus not evaluated. Moreover, the metrics in the existing test-beds depend on the utility that is measured. Examples range from computer network metrics, such as the accuracy of routing and number of hops [6], to more abstract ones, like the amount of earnings in an art appraisal contest [7]. While such metrics are sufficient to rank trust models, they offer poor insight into their actual performance.

In this paper we put forward a test-bed to tackle the presented shortcomings. We describe the design and present an implementation of a test-bed called Alpha. Alpha test-bed evaluates trust models by validating rankings of agents that emerge from trust models' estimations. The main idea of the test-bed is to give a trust model to a particular agent, called agent α, and then fix the other agents in the system with a pre-defined behaviour. Additionally, the test-bed also assigns interaction partners to agent α. This way we ensure that agent α receives the same information regardless of the trust model it uses. This enables us to easily compare the performance of different trust models – all we have to do is change α's trust model and run the evaluation with the same parameters. The remainder of the paper is organised as follows: Section 2 introduces suitable trust models and formally describes their components that are relevant for evaluation. We propose the Alpha test-bed in Section 3 following above described principles. Then we present a prototype implementation in Section 4. We discuss the proposal with related work in Section 5, and conclude the paper with Section 6.

2 Trust models

Trust models form trust from various types of information sources. These include experiences from interactions, opinions from other agents in the system, the analysis of underlying social structures of agents and so on. However, most of the models only pay attentions to the use of experiences and opinions, which is why we shall restrict ourselves to investigate only such models. This section describes an information environment, in which agents with such trust models operate, and presents a formal notation of their components. Despite the mentioned restriction, we believe that the architecture of the test-bed as we present it here, is applicable even to models that use additional information sources. In such cases, one would have to provide additional facilities that generate suitable information that would later be given to the trust model.

2.1 Information sources and information flows

The sources of information and its flow through an agent are shown in Fig. 1. The main parts of the figure are **agent** α, which represents the agent whose trust model we intend to test, **opinions from agents**, which represents agents that provide agent α with opinions about third parties, and the **interactions with agents**, which represent agents that interact with agent α. We also show an **environment** component, which represents other types of information from which α can estimate trust (social networks, virtual organizations), but we shall not utilize it in this proposal. Agent α's internal architecture is further decomposed into i) *interpretation*, ii) *trust model* and iii) *decision making*.

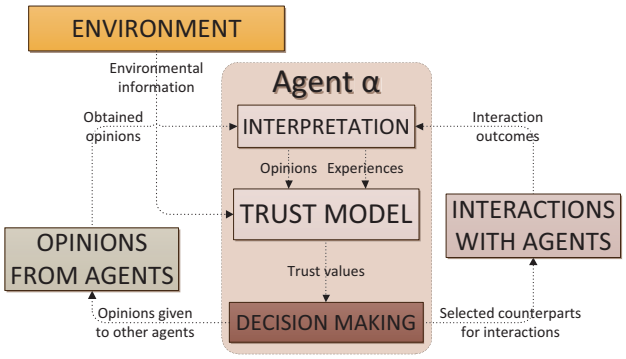


Figure 1: Sources and flow of trust information

The two information flows in this schema are the interaction flow and the opinion flow. The interaction flow goes from *interactions* to agent α and then back to *interactions*. In this flow α's *interpretation* component processes interaction outcomes and outputs experience values that are then conveyed to the *trust model*. The *trust model* uses experiences (as well as opinions) to calculate trust values. The latter are then used by the *decision making* component to select collaborators for interactions. Decision making is usually a very complex process and while trust values are an important part of its input, decision making in selecting interaction counterparts may also consider other factors. However, they are domain specific and generally independent from the trust model. The opinion flow goes from *opinions* to agent α and then back to *opinions*. When agent α obtains an opinion the *interpretation* component transforms the opinion to an understandable format for the *trust model*. The trust model then uses those transformed opinions (as well as experiences) to compute trust values that are used by the *decision making* component to report trust values to the other agents.

2.2 Formalization

Here we formally represent concepts that are relevant for evaluation. We denote the set of all agents in the system with $\mathcal{A}$. Even though α is an agent and part of the system, we exclude it from $\mathcal{A}$ due to his special status (individual to be tested), thus $\alpha \notin \mathcal{A}$. Additionally, we denote the set of all types of services that agents offer with $\mathcal{S}$.

Experiences An experience $\varepsilon \in \mathcal{E}$ is a record of an interaction between some agent and agent α. We denote experience as a tuple $\varepsilon = \langle a, s, t, \lambda \rangle$, where $a \in \mathcal{A}$ represents the agent that provided α with service $s \in \mathcal{S}$ at time $t \in \mathcal{T}$, and $\lambda \in \Lambda$ represents the assessment that α gave to the interaction. An assessment $\lambda \in \Lambda$ is a value with which agent α evaluates the performance of the service provider. While the test-bed is agnostic to the types of assessments (numeric, qualitative, binary, descriptive), it requires that the user provides a mapping from $[0, 1]$ to the set of assessments Λ. The mapping is needed for interpreting interaction outcomes and converting them into assessments. The set of all experiences is denoted with $\mathcal{E}$.

Trust values A trust value $\tau \in \Theta$ represents α's trust towards a particular agent. We denote trust value as a tuple $\tau = \langle a, s, \omega \rangle$, where $\omega \in \Omega$ represents α's trust degree towards agent $a \in \mathcal{A}$ for service $s \in \mathcal{S}$. A trust degree $\omega \in \Omega$ is a value with which an agent expresses the level of trust towards another agent. Similar to assessments, the test-bed permits all kinds of trust degrees, but it requires that the user provides a mapping from $[0, 1]$ to the set of trust degrees Ω. The mapping is needed for interpreting opinions and converting them into trust degrees that the tested model can understand. Additionally, the test-bed requires that set Ω is totally ordered, which means that any two trust degrees have to be comparable with a greater-than-or-equal relation. Analogously, we denote the set of all possible trust values with Θ.

Often we will need to reference a single component of a trust tuple. In such cases we will use *projections*; the agent projection $\pi_{\mathcal{A}}(\langle a, s, \omega \rangle) = a$, the service projection $\pi_{\mathcal{S}}(\langle a, s, \omega \rangle) = s$ and the trust degree projection $\pi_{\Omega}(\langle a, s, \omega \rangle) = \omega$.

Opinions An opinion $o \in \mathcal{O}$ is a statement about trust that was given by one agent about a third party to agent α. We denote opinion as a tuple $o = \langle a_o, a_p, s, t, \omega \rangle$, where $a_o \in \mathcal{A}$ denotes the agent that told α that its trust degree towards agent $a_p \in \mathcal{A}$ for service $s \in \mathcal{S}$ is $\omega \in \Omega$. Symbol $t \in \mathcal{T}$ denotes the time at which opinion was given and $\mathcal{O}$ denotes the set of all possible opinions.

Trust model A trust model is an architectural component that computes α's trust towards other agents. We denote trust model as a set function $\text{TrustModel} : \mathcal{T} \times \mathcal{P}(\mathcal{E}) \times \mathcal{P}(\mathcal{O}) \to \mathcal{P}(\Theta)$ that at any given time $t \in \mathcal{T}$ takes a set of experiences $\varepsilon_{set} \in \mathcal{P}(\mathcal{E})$ and a set of opinions $o_{set} \in \mathcal{P}(\mathcal{O})$ and outputs a set of trust values $\tau_{set} \in \mathcal{P}(\Theta)$, thus $\tau_{set} = \text{TrustModel}(t, \varepsilon_{set}, o_{set})$.

3 The Alpha test-bed

Trust is hard to measure directly. An often used approach is to measure the utility that an agent endowed with a trust model obtains from interactions. Such evaluation coincides with measuring the interaction outcomes in Fig. 1. This is troublesome, because it directly measures not the effect of the trust model but the effect of the decision making (since this is the component directly responsible for selecting interaction partners). And while the trust model influences the decision making, there are also other factors that come into account. Measuring interaction outcomes thus has two important consequences. First, the results of such evaluation are ambiguous, because one cannot assure whether the obtained utility is due to the use of a good trust model or due to

the use of a good decision making component. Secondly, it forces evaluation to use metrics that are specific to the gained utility. Such metrics measure the performance of an agent but not the actual quality of its trust model.

Our goal is to test the performance of a trust model directly and independently of the decision making component. The problem is that while trust model and decision making have different goals, they also influence each other. The goal of the decision making component is to select such interaction partners that an agent maximizes its utility in interactions. Decision making can use calculated trust values to reach better decisions. On the other hand, the goal of a trust model is to provide a comparable mental attribute of an agent's potential partners – to provide estimations of their trustworthiness. But because trust model estimates trust from experiences (which are derived from past interactions), the decision making also influences the trust model. This means that two agents, who have the same trust models but different decision making components, can eventually reach different trust estimations. The difference will be due to different partner selections in the past and thus different inputs to their trust models. To overcome this inter-dependency, we propose the test-bed to be in charge of the decision-making process and thus in charge of the partner selection process.

We put forward a test-bed that directly measures the output of a trust model – calculated trust values. During the evaluation the test-bed generates the complete information that represents α's experiences and obtained opinions. In other words, the test-bed controls not only the behaviour of other agents in the system but also the selection of α's interaction counterparts. The generated information is given to α's trust model, whose output is then evaluated.

3.1 Capability of agents

As stated above, the test-bed controls the behaviour of agents in $\mathcal{A}$. It controls how they behave in interactions with α and also how they report opinions to α. We model behaviour of agents in such situations with capabilities.

Definition 1 (Capability) *A capability represents the ability and willingness of an agent $a \in \mathcal{A}$ to provide a reliable service $s \in \mathcal{S}$ to agent α at time $t \in \mathcal{T}$. We express it as a real number from $[0, 1]$, where 0 and 1 represent the lowest and highest capability, respectively, thus $\mathrm{C} : \mathcal{A} \times \mathcal{S} \times \mathcal{T} \to [0, 1]$.*

Capabilities represent the ground truth – the actual performance of agents in interactions. We use capability as a basis (i) to construct α's experience (to simulate interactions outcomes) and (ii) to create opinions that agents give to α. (Detailed descriptions of generating interaction outcomes and generating opinions are given in sections 3.3 and 3.4.) The actual capabilities are never revealed to α, so it can only infer from interactions or when agents provide opinions about other agents. Therefore we use capabilities as an integral part of evaluation.

3.2 Metrics

Estimating the quality of future interactions is the essential task of any trust model [8]. Since capabilities define the quality of interactions, we can evaluate the trust model's performance by measuring the similarity between calculated trust and the actual capabilities of agents. We could measure similarity by

computing differences between estimated trust degrees and the capabilities of agents, but this would require all trust models to estimate trust with values from $[0, 1]$. While some models represent trust in this way, there are also many others, which use different representations. Mapping estimations to the interval $[0, 1]$ may be unsuitable for some models, because they were not intended for such cases. However, trust degree is a comparable attribute, which means that any trust model should be able to rank agents by their trustworthiness (regardless the actual domain of trust degrees).

Definition 2 (Evaluation) *Evaluation function* $\mathrm{Eval} : \Theta \times \Theta \times \mathcal{T} \to \{0, 1\}$ *determines if a pairwise comparison of given trust values* $\langle a_i, s, \omega_i \rangle, \langle a_j, s, \omega_j \rangle \in \Theta$ *is aligned with the pairwise comparison of the respective agents' capabilities at time* $t \in \mathcal{T}$.

$$\mathrm{Eval}(\langle a_i, s, \omega_i \rangle, \langle a_j, s, \omega_j \rangle, t) = \begin{cases} 1 & \textit{if } [\ \omega_i \geq \omega_j \\ & \quad \wedge\ \mathrm{C}(a_i, s, t) \geq \mathrm{C}(a_j, s, t)\ ] \\ & \quad \vee\ [\ \omega_i < \omega_j \\ & \quad \wedge\ \mathrm{C}(a_i, s, t) < \mathrm{C}(a_j, s, t)\ ] \\ 0 & \textit{else} \end{cases}$$

Since the set of trust degrees Ω is totally ordered, we can evaluate any pair of trust values with the pairwise comparison from Def. 2. The pairwise comparison of trust values is aligned with the pairwise comparison of capabilities when, for a given service s and time t, the capability of agent a_i is lower than (or equal to) the capability of agent a_j, while the degree of trust towards agent a_i is also lower than (or equal to) the degree of trust towards agent a_j. The comparison is also aligned, if, analogously, the capability of agent a_i is greater than the capability of agent a_j, while the degree of trust towards agent a_i is also greater than the degree of trust towards agent a_j. In all other cases the comparison of trust values is not aligned with the comparison of capabilities.

To evaluate an entire set of trust values, we invoke evaluation function for every possible pair of trust values in that set. We call this accuracy.

Definition 3 (Accuracy) *Accuracy* $\mathrm{Acc} : \mathcal{P}(\Theta) \times \mathcal{S} \times \mathcal{T} \to [0, 1]$ *evaluates all trust values from a given set* $\tau_{set} \in \mathcal{P}(\Theta)$ *that concern the given service* $s \in \mathcal{S}$. *The evaluation is performed at given time* $t \in \mathcal{T}$.

$$\mathrm{Acc}(\tau_{set}, s, t) = \frac{\sum\limits_{\substack{\tau_1, \tau_2 \in \tau_{set} \\ \pi_{\mathcal{S}}(\tau_1) = s \\ \pi_{\mathcal{S}}(\tau_2) = s}}^{\tau_1 \neq \tau_2} \mathrm{Eval}(\tau_1, \tau_2, t)}{\mathrm{Count}(\tau_{set}, s) \cdot (\mathrm{Count}(\tau_{set}, s) - 1)}$$

where function $\mathrm{Count} : \mathcal{P}(\Theta) \times \mathcal{S} \to \mathbb{N}_0$ returns the number of trust values that in set $\tau_{set} \in \mathcal{P}(\Theta)$ concern the given service $s \in \mathcal{S}$.

$$\mathrm{Count}(\tau_{set}, s) = \Big| \bigcup_{\substack{\tau \in \tau_{set} \\ \pi_{\mathcal{S}}(\tau) = s}} \pi_{\mathcal{A}}(\tau) \Big|$$

The accuracy tells us how well does the ranking of agents imposed by the calculated trust values coincide with the ranking of agents imposed by their actual capabilities. In the numerator we sum the invocations of evaluation function, in which we compare all possible pairs of trust values that concern the given service. In the denominator we calculate the total number of comparisons.

This serves as a normalization factor, which ensures that when all evaluated pairs of trust values are correct, the accuracy is 100%.

The differences in capabilities between agents are often very small. In such cases, we require that trust model differentiates only between agents that have *substantial differences* in capabilities. For such purposes we redefine accuracy.

Definition 4 (Accuracy with sensitivity) *Accuracy with sensitivity parameter* $\mathrm{Acc} : \mathcal{P}(\Theta) \times \mathcal{S} \times \mathcal{T} \times [0,1] \to [0,1]$ *evaluates only those pairs of trust values, in which the difference in capabilities between agents is larger than the sensitivity parameter* $\delta \in [0,1]$*. This type of accuracy is defined as follows:*

$$\mathrm{Acc}(\tau_{set}, s, t, \delta) = \frac{\sum_{\substack{\tau_1,\tau_2 \in \tau_{set} \\ \pi_{\mathcal{S}}(\tau_1)=s \\ \pi_{\mathcal{S}}(\tau_2)=s \\ \tau_1 \neq \tau_2}} \mathrm{Eval}(\tau_1, \tau_2, t) \cdot \mathrm{Cmp}(\pi_{\mathcal{A}}(\tau_1), \pi_{\mathcal{A}}(\tau_2), s, t, \delta)}{\sum_{\substack{\tau_1,\tau_2 \in \tau_{set} \\ \pi_{\mathcal{S}}(\tau_1)=s \\ \pi_{\mathcal{S}}(\tau_2)=s \\ \tau_1 \neq \tau_2}} \mathrm{Cmp}(\pi_{\mathcal{A}}(\tau_1), \pi_{\mathcal{A}}(\tau_2), s, t, \delta)}$$

where the comparable function $\mathrm{Cmp} : \mathcal{A} \times \mathcal{A} \times \mathcal{S} \times \mathcal{T} \times [0,1] \to \{0,1\}$ determines whether the difference in capability between given agents $a_i, a_j \in \mathcal{A}$ for the selected service $s \in \mathcal{S}$ and at time $t \in \mathcal{T}$ is big enough.

$$\mathrm{Cmp}(a_i, a_j, s, t, \delta) = \begin{cases} 1 & \text{if } |\,\mathrm{C}(a_i, s, t) - \mathrm{C}(a_j, s, t)| \geq \delta \\ 0 & \text{else} \end{cases}$$

In the numerator of the redefined accuracy, we use comparable function to determine when to take the evaluation function into account, while the sum of invocations in the denominator serves as the normalization factor. As expected, in case we set sensitivity parameter to $\delta = 0$ the accuracy from Def. 4 collapses to the accuracy from Def. 3.

The accuracy by itself says nothing about the number of trust values in the given set with respect to the number of all agents. For instance, if the set of trust values contains three values, but the entire society consists of 100 agents, the accuracy can still be 100% if the pairwise comparison of those three values is correct. But this is hardly a good result. Such cases show the need for an additional metric.

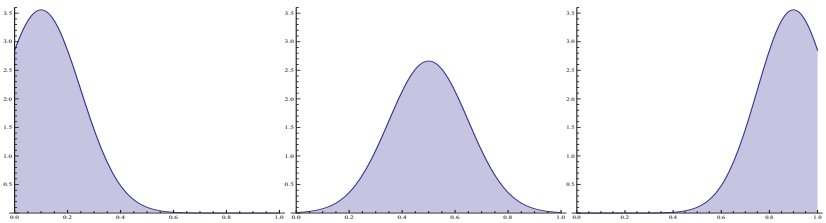
Definition 5 (Coverage) *Coverage* $\mathrm{Cov} : \mathcal{P}(\Theta) \times \mathcal{S} \to [0,1]$ *indicates the percentage of agents towards which, for a given service* $s \in \mathcal{S}$*, trust value is defined in the given set* $\tau_{set} \in \mathcal{P}(\Theta)$*.*

$$\mathrm{Cov}(\tau_{set}, s) = \frac{\mathrm{Count}(\tau_{set}, s)}{|\mathcal{A}|}$$

Coverage returns the percentage of agents towards which α is able to compute trust for a given service. In the case of 3 out of 100 agents the coverage is 3%. To get a sense of how good calculated trust values are, we have to consider both, accuracy and coverage. While accuracy tells us how accurate is the ranking of agents imposed by the trust model, the coverage tells us what is the percentage of agents towards which trust model can estimate trust.

3.3 Generating experiences

Interactions between α and other agents are not actually performed, they are simulated. When α interacts with an agent, we create an experience



(a) Capability = 0.1 (b) Capability = 0.5 (c) Capability = 0.9

Figure 2: Example of PDFs for interaction outcomes

$\varepsilon = \langle a_i, s, t, \lambda \rangle$, where a_i denotes the providing agent, and symbols s, t and λ denote the type of service, time of interaction and the assessment of the interaction, respectively. We calculate the assessment in a two-step procedure. First, we determine the outcome of the interaction, and then we convert it into an assessment.

3.3.1 Modelling interaction outcomes

The outcome of an interaction depends on the capability of the agent that provides the service. Therefore we represent interaction outcomes as values from [0, 1], where 0 and 1 represent the worst and best possible interaction outcomes, respectively. But to simulate a realistic setting, a particular interaction outcome can be different than the actual capability of the providing agent. The difference, however, should not be substantial and in the long run, interaction outcomes should resemble the actual capabilities of providing agents. For this purpose we generate interaction outcomes with a pseudo-random generator. The generator operates with a probability distribution function (PDF) that is parametrized with the capability of the providing agent. The general shape of the PDF is defined as follows:

$$p(x; \mu, \sigma) = \begin{cases} \frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\int_0^1 e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt} & 0 \leq x \leq 1 \\ 0 & \text{else} \end{cases}$$

This is PDF of a truncated normal distribution with mean μ and standard deviation σ. Truncation ensures that a) the probability of obtaining a value outside of the interval [0, 1] is 0, and that b) the area under the PDF curve equals to one (i.e. that the $p(x; \mu, \sigma)$ is a proper PDF). To generate an interaction outcome between α and agent a_i for service s and at time t, we invoke a pseudo-random generator and set the parameter μ to the capability of the providing agent, thus $\mu = \mathrm{C}(a_i, s, t)$, while σ can be arbitrary. For example, Fig. 2 shows probability density functions for three agents with capabilities 0.1, 0.5 and 0.9, respectively. The σ is set to 0.15 in all three cases.

3.3.2 Converting interaction outcomes to assessments

Once the outcome of the interaction is determined, we convert it to an assessment. For that purpose the user must define an assessment conversion function $\mathrm{Conversion}_\Lambda : [0, 1] \to \Lambda$. Conversion is specific to the trust model, we are testing. For instance, if the set of assessments Λ is the same as [0, 1], the

conversion function can be an identity function, otherwise the user must provide some other appropriate mapping. In summary, the complete procedure to create an experience consists of (i) generating a random value and (ii) converting that value into an assessment. The random value represents the interaction outcome and is selected from [0, 1] using a PDF that is parametrized with the capability of the agent that provides the service. The assessment is computed from the interaction outcome with the user-provided conversion function.

3.4 Generating opinions

Agents in set $\mathcal{A}$ are supposed to provide opinions to α. In the real world, those opinions would come from their trust models. Since we are not interested in the performance of those trust models, only in the performance of α's trust model, we simulate trust models of other agents with a carefully configured pseudo-random generator. Thus we ensure that α receives the same opinions in every test, regardless its own and other's trust model.

Opinions are defined as tuples $o = \langle a_i, a_j, s, t, \omega \rangle$, where ω denotes trust degree that agent a_i told α to have towards agent a_j for service s at time t. In comparison to generating experience, generating opinions is a three-step procedure. First, we determine the trust degree of agent a_i towards agent a_j. This trust degree is internally represented with a value from [0, 1], where values 0 and 1 represent the lowest and the highest degrees, respectively. In the second step we determine how agent a_i is going to report this value to α. This means that we determine if agent a_i is going to lie to α, and if, how. In the third step, we convert internally represented trust degree to a trust degree that is compatible with α's trust model. The latter is accomplished using a user-provided mapping. Following sections explain the details of these steps.

3.4.1 Calculating internal trust degrees

We calculate internal trust degrees between agents in a similar way that we calculate interaction outcomes – we use a pseudo-random generator with a PDF that is parametrized with the capability of the agent that provides the service[1]. Thus, the internally represented trust degree of agent a_i towards agent a_j for service s and at time t is calculated with a pseudo-random generator with PDF that has the mean parameter set to $\mu = \mathrm{C}(a_j, s, t)$. The parameter σ can be, similar to before, arbitrary. The calculated value (in $[0, 1]$) represents the internal trust degree of agent a_i towards agent a_j for service s and at time t.

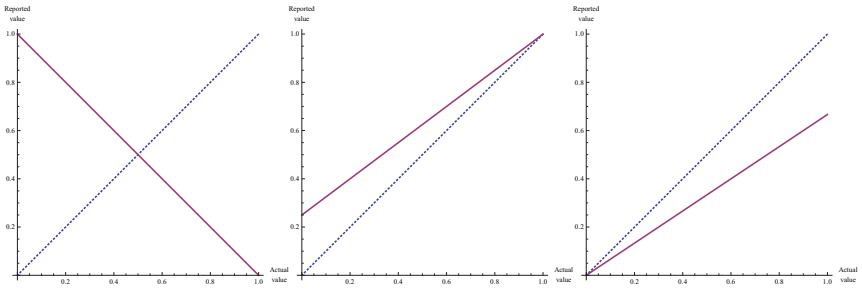
3.4.2 Modelling deception

In the second step of generating an opinion, we determine whether the agent that gives the opinion will be truthful and if not, how will the cheating process look like. We compute the value that will be reported to α with a deception model. The deception model is thus a mapping $d : [0, 1] \to [0, 1]$. The testbed assigns a deception model to every agent. The assignment also depends on which agent the opinion is about, the service, and the current time, thus

[1] The underlying assumption is that agents (as service providers) provide services with the same quality to all consumers. We handle quality discrimination (the case of providing different quality of service depending on the identity of the consumer) with deception models, which are described in the following subsection.

Deception : $\mathcal{A} \times \mathcal{A} \times \mathcal{S} \times \mathcal{T} \rightarrow \mathcal{D}$, where $\mathcal{D}$ stands for the all possible deception models. An example of such a set is given below.

We will borrow deception models from Yu and Singh [9]. They used three models of deception; complementary, exaggerated positive and exaggerated negative. In the complementary model agents report opinions with the complementary trust degrees – if the actual degree is good, they report bad and vice-versa. In the exaggeration models agents provide opinions with trust degrees that are in comparison to the actual trust degrees either overestimated (in case of positive exaggeration) or underestimated (in the case of negative exaggeration). The three models are shown in Fig. 3. To contrast the difference between actual and reported values, we used a dotted blue line to plot a truthful model – a model in which agents provide honest opinions.



(a) Complementary model (b) Positive exaggeration (c) Negative exaggeration

Figure 3: A graphical representation of deception models

We are adding two more models to this set, a *random* model and a *silent* model. In random model agents provide opinions with trust degrees that are chosen randomly, while in silent model agents do not provide opinions at all. The silent model simulates behavior of agents that are either unwilling or unable to share their opinions. Set $\mathcal{D}$ represents the set of all deception models that can be used in an evaluation. In our case we have $\mathcal{D} = \{d_t, d_c, d_{pe}, d_{ne}, d_r, d_s\}$, where every model is defined as follows (parameter $0 \leq \kappa \leq 1$ denotes the exaggeration coefficient; Fig. 3 uses $\kappa = 0.25$):

- truthful model $d_t(x) = x$;
- complementary model $d_c(x) = 1 - x$;
- positive exaggeration model $d_{pe}(x) = x \cdot (1 - \kappa) + \kappa$;
- negative exaggeration model $d_{ne}(x) = x \cdot (1 - \kappa)$;
- random model $d_r = \mathcal{U}(0, 1)$;
- silent model $d_s = \varnothing$.

Deception models in $\mathcal{D}$ constitute the most common ways of providing opinions in an open and dynamic multi-agent environment; there are some agents that provide honest opinions, some that lie extensively [9] and some that are biased in either positive or negative direction [10]. Random model simulates the most unpredictable agents whose opinions are the least reliable, while the silent model accounts for the scarcity of trust information in such environments.

We can also use deception models to simulate discriminating behaviour of agents as service providers. When an agent behaves differently towards Alpha than it behaves towards other agents, then the opinions of other agents about this particular provider become lies from Alpha's standpoint. For Alpha, there is no difference between an agent providing a false opinion about a third party, and an agent providing a truthful opinion about the same third party, if the third party discriminates. In either case, the opinion is of no use to Alpha.

3.4.3 Conversion of internal trust degrees

In the last step of generating an opinion we convert internally represented trust degrees to trust degrees that are compatible with the tested trust model. Trust degrees that we used so far were internal, which means they were all expressed on a closed interval [0, 1], with 0 and 1 being the lowest and the highest degrees, respectively. But because not all trust models use such representation of trust degrees, we have to convert it into a proper domain. We do this with an user-provided trust degree conversion function $\text{Conversion}_\Omega : [0, 1] \to \Omega$. The same principles used for the assessment conversion function also apply here.

Therefore, the complete procedure to create an opinion consists of (i) generating a random value, (ii) calculating the reported value and (iii) converting that reported value into a trust degree. The random value represents internal trust degree between two agents and is selected in the range [0, 1] using a PDF that is parametrized with the capability of the agent in question. The calculated number is then modified with the deception model of the agent that provides the opinion. Finally, the reported value is transformed into a proper trust degree with the user-provided conversion function.

4 Alpha implementation

Since the evaluation is performed in discreet time steps, we implemented the Alpha test-bed as an agent-based simulator. We used the Repast Simphony [11] as a platform, since it offers many of the needed facilities, such as event scheduling, graph plotting or data exporting. Alpha test-bed is thus a standard Repast model with a couple of additional features that enable dynamic addition of new trust models and other pluggable components.

4.1 Pluggable approach

We designed the Alpha test-bed to be extendible. This means that we hard-coded only the main concepts, while some parts can be changed by providing a suitable plug-in. Alpha test-bed currently supports four types of pluggable components, namely scenarios, trust models, metrics and deception models. While we provide default implementations for all components, adding new ones is straightforward. The user implements the interface of the desired component, packages the implementation in a JAR and annotates the JAR with a standard Service Provider configuration file. Once placed inside test-bed's classpath, the test-bed dynamically loads JAR at runtime.

Scenarios are responsible for generating experiences and opinions. They i) assign capabilities and deception models to agents; ii) define the order, in which agent α interacts with other agents; and iii) define the order, in

which agent α queries other agents for opinions. We use scenarios to construct an information environment, in which we evaluate trust models. For instance, to test a trust model in an environment, in which agent α interacts only with a small number of agents and has to rely on its reputation mechanism to calculate trust towards the remaining agents, we should use a scenario that reflects such setting – that generates appropriate experiences and opinions. On an implementation level, scenarios are classes that implement `IScenario` interface. Implementing classes have to provide methods to generate experiences `Set<Experience> generateExperiences(Time)`, generate opinions `Set<Opinion> generateOpinions(Time)` and generate capabilities `Map<Agent, Capability> getCapabilities(Service, Time)`. The current version has a few scenario implementations, but new ones can be easily added.

Trust models are parameters of the evaluation, but they are loaded by the test-bed as plug-ins. They have to be programmed as classes that implement the `ITrustModel` interface. The class must provide a method `void calculateTrust(Set<Experience>, Set<Opinion>)`, which updates the model with new experiences and opinions, and a method `Map<Agent, Rank> getRankings(Service)`, which returns the rankings of agents for the given type of service.

The third type of plug-in are metrics. Their implementations are defined by the `IMetric` interface. While we provide Accuracy and Coverage out of the box, users can add new ones by implementing `double evaluate(Map<Agent, Rank>, Map<Agent, Capability>)`.

4.2 Running an evaluation

Running an evaluation is a two phase process. First, the user has to bootstrap the test-bed and set up the parameters. Once this is done, the evaluation starts as a series of discrete time steps. During the evaluation the test-bed outputs various data that are shown to the user.

At the bootstrapping phase the user has to select a scenario, a trust model, and set of metrics. Loading scenario often requires passing in additional parameters, such as the number of agents and different types of services in the system, the assignment of capabilities and deception models to agents, and so on. Trust models may also require additional parameters – the number and the type of these parameters depend on the trust model. The third type of parameters concerns metrics. Some metrics require parameters (i.e. accuracy with sensitivity), while others do not (i.e. coverage). Once all plug-ins are loaded and initialised with parameters, we can start the evaluation.

The evaluation consists of a series of operations that are carried out at every tick. At the start of a tick, the test-bed requests new experiences and opinions by invoking scenario with `Set<Experience> generateExperience(Time)` and `Set<Opinion> generateOpinions(Time)`. Then it conveys the results to the trust model by calling `calculateTrust(Set<Experience>, Set<Opinion>)`. Next, the test-bed queries the trust model for rankings by invoking `Map<Agent, Ranking> getRankings(Service)`. The rankings are finally evaluated by calling `double evaluate(Map<Agent, Rank>, Map<Agent, Capability>)`. The test-bed repeats the last two operations for every type of service and for every metric. Once the results are calculated, the Repast facilities plot graphs of metrics and, optionally, write the results to a file. The described sequence is repeated at every tick.

5 Related work

When a new trust model is proposed, its authors often build an *ad-hoc* and domain oriented simulator [1, 12]. Whilst such simulators can demonstrate the effectiveness of proposed models, they cannot be used as general purpose test-beds, since they are domain specific. This means that when one wants to test another, slightly different trust model, one has to add additional reasoning to it, just to be compatible with the specific domain of that test-bed. This limits the test-beds generality and the generality of its results.

Only a few researchers address the problem of building a general purpose test-bed. The ART test-bed [7] was a courageous and well-known initiative to fill this gap. They proposed a test-bed, in which agents enact the role of art appraisers. Agents have to estimate the actual worth of paintings by either using their own judgement (capabilities are assigned to them by the test-bed) or by asking other agents for help. Agents can use trust models to i) evaluate, how other agents can help them in estimating prices of paintings and to ii) decide, how opinions from other agents can help them in finding additional appraisers. During evaluations the test-bed pays agents for their appraisals in proportion to how accurate their appraisals are. Moreover, agents can use earnings to purchase opinions or appraisals from other agents and thus improve quality of their own appraisals. At the end of competition the agent with the most earnings wins. The setting of ART test-bed is a good example of a competitive multi-agent environment, where agents need complex reasoning to perform well. Analysis of past ART competitions revealed that the successful competitors did not concentrate much on developing good trust models, but more on deciding how to invest their earnings (i.e. building a good decision making processes) [13].

TRMSim-WSN [6] is a dedicated test-bed for benchmarking trust and reputation models in the wireless-sensor networks. The test-bed creates a WSN, equips nodes with a tested trust model and defines, which sink-nodes will behave benevolently and which maliciously. Then it simulates WSN traffic. The nodes have to send packets to benevolent sink-nodes and avoid malicious ones. Test-bed's main metric is accuracy, which tells the percentage of nodes that send packets to benevolent sinks. TRMSim-WSN in effect measures decision making, but in comparison to ART test-bed, calculated trust is the only factor to consider. This approach evaluates only a subset of trust model's estimations – as long as the tested trust model selects a benevolent sink to be the most trustworthy (even when this is not the "best" sink), the model receives perfect score. However the Alpha test-bed evaluates the entire output of a trust model.

We distinct between *system-oriented* and *individual-oriented* approaches. Some test-beds, such as [12, 6], measure the trust model's effect on the global utility of the system, while others, such as [7, 1], measure the effect of trust model for an individual agent. In both cases, test-beds measure utility, which in our opinion yields ambiguous results.

6 Conclusion

We have put forward a novel test-beds for evaluating trust models. While others evaluate trust models by measuring an utility that an agent with a trust model gains in interactions, we evaluate trust models directly by measuring their outputs – trust values. We have demonstrated that existing test-beds

measure not the quality of an agent's trust model, but the quality of the agent's decisions. And we have demonstrated (with arguments and with examples in the literature) that this distinction is anything but minute. We have presented objective and comparable metrics, which are general and intuitive. Moreover, we designed and implemented a test-bed called Alpha to cover the mentioned issues. We thus provide a way of thoroughly testing and comparing trust models, which fills the existing gap in the literature. In future, we intend to thoroughly document and publicly release Alpha's source code together with additional user documentation. We also plan to test different trust models and compare their results.

References

[1] Trung Huynh, Nicholas Jennings, and Nigel Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 2006.

[2] Damjan Kovač and Denis Trček. Qualitative trust modeling in soa. *Journal of Systems Architecture*, 2009.

[3] Javier Lopez, Rodrigo Roman, Isaac Agudo, and Carmen Fernandez-Gago. Trust management systems for wireless sensor networks: Best practices. *Computer Communications*, 2010.

[4] Ernesto Damiani, De Capitani di Vimercati, Stefano Paraboschi, Pierangela Samarati, and Fabio Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, 2002.

[5] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 2007.

[6] Félix Gómez Mármol and Gregorio Martínez Pérez. Trmsim-wsn, trust and reputation models simulator for wireless sensor networks. In *Proceedings of the 2009 IEEE international conference on Communications*, 2009.

[7] Karen K. Fullam, Tomas B. Klos, Guillaume Muller, Jordi Sabater, Andreas Schlosser, K. Suzanne Barber, Jeffrey S. Rosenschein, Laurent Vercouter, and Marco Voss. A specification of the agent reputation and trust (art) testbed: experimentation and competition for trust in agent societies. In *In Proceedings of the 4th International Joint Conference on Autonomous Agents and MultiAgent Systems*, 2005.

[8] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys*, 2009.

[9] Bin Yu and Munindar P. Singh. Detecting deception in reputation management. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 2003.

[10] Paul Resnick and Richard Zeckhauser. Trust Among Strangers in Internet Transactions: Empirical Analysis of eBay's Reputation System. In *The Economics of the Internet and E-Commerce*. Emerald Group Publishing Limited, 2002.

[11] M. J. North, T. R. Howe, N. T. Collier, and J. R. Vos. A declarative model assembly infrastructure for verification and validation. In *Advancing Social Simulation: The First World Congress*, 2007.

[12] Ramón Hermoso, Holger Billhardt, Roberto Centeno, and Sascha Ossowski. Effective use of organisational abstractions for confidence models. In *Proceedings of the 4th European Workshop on Multi-Agent Systems EUMAS '06*, 2006.

[13] Mario Gómez, Jordi Sabater-Mir, Javier Carbó, and Guillaume Muller. Analysis of the agent reputation and trust testbed. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, 2008.

An agent platform for self-configuring agents in the Internet of Things

Inmaculada Ayala[1], Mercedes Amor[1], and Lidia Fuentes[1]
[1] *Universidad de Málaga, Departamento de Lenguajes y Ciencias de la Computación,*
{ayala,pinilla,lff}@lcc.uma.es

Abstract

The Internet of Things (IoT) envisions a world in which an heterogeneous set of devices are interconnected and collaborate using the Internet in order to provide valuable services for users. For the developer, the deployment of applications and services for the IoT requires managing an heterogeneous set of devices, communication protocols and underlying networks, in order to resolve interoperability issues due to the heterogeneity of the IoT nodes. Agent technology offers the necessary means to manage distribution and many other requirements of the IoT satisfactorily, however current agent platforms neither deal adequately with the heterogeneity of these environments, nor provide support for a self-configuring communication in Multi-Agent Systems. In this contribution we present *Sol*, an agent platform to develop IoT applications deployed as a family of self-configuring agents for heterogenous devices. We illustrate the benefits of our approach for several scenarios of an Intelligent Museum, and show its feasibility in terms of the response time of reconfiguration, and wireless data exchange, so important in the IoT.

Keywords: Agent Communication Technologies, Aspect Orientation, Internet of Things, Agent Platforms, Sensors, Handheld devices.

1 Introduction

The Internet of Things (IoT for short) envisions a world in which everyday objects (such as mobile phones, vehicles, electrical appliances, medical instruments, etc.) are interconnected and collaborate using the Internet in order to provide valuable services for users. This vision defines the IoT as a dynamic global network infrastructure with global self-configuring capabilities, based on standard and interoperable communication protocols where physical and virtual *things* have identities, physical resources and are seamlessly integrated into the information network [4]. Many technologies and new technological advances are making this vision possible: Ambient Intelligent (AmI) environments, automatic object identification (such as RFID and NFC), ubiquitous connectivity, just to mention a few. The possibilities offered by the IoT make the development of a huge number of novel applications possible with the goal of improving the quality of our lives: at home, while traveling, when being a tourist, when sick, or at work.

However, the IoT vision can be achieved by a convergence of technologies to cover the heterogeneity in different contexts inherent to these kind of systems [18]. The development of services and applications for the IoT demands dealing with a diverse set of heterogeneous devices (typical of AmI environments). A service must be able to be executed in a diversity of devices, with a variable set of physical features and software services availability. Also, new devices are continually appearing, so the IoT underlying technologies should provide the means to update the set of considered devices. Moreover, inter-device communication is a priority issue for the IoT, but the set of communication technologies and protocols is also diverse and sometimes the *objects* involved in a IoT application do not share the same communication protocol. In fact, sometimes the access to services is restricted by the set of communication protocols of each device, which is not desirable. Capabilities for different kinds of self-configuration, specially those focus on solving inter-communication differences, is considered a high priority for the IoT. So, the IoT needs an open architecture to maximize interoperability among heterogeneous systems and distributed resources including providers and consumers of information and services, whether they be human beings, software, smart objects or devices. This architecture should also consider that IoT can be formed by a myriad of different devices, which must addressed and located, using efficient mechanisms easy to use for application developers.

The agent technology offers the necessary means to manage many of IoT distribution requirements satisfactorily: software agents are reactive, proactive and their communication is supported by distributed agent platforms (APs). There are different APs which work in some of the IoT devices. So, agents running in IoT nodes provide a good way to encapsulate functionality, abstracting applications from underlying heterogeneous hardware and implementation details. Moreover, if every object or IoT node is an agent, the discovery of new agents is feasible (i.e new objects and IoT nodes) through the Directory Facilitator (DF) provided by the AP, dealing adequately with the addressability problem of the IoT.

However, current APs and agent development toolkits for lightweight devices are not completely capable of managing heterogeneity and have strong limitations for ensuring interoperability as IoT requires. The existing agent solutions that could be applicable to the IoT are available only for a specific set of devices. For example, Jade-Leap facilitates the execution and communication of agents on top of Android and J2ME-based lightweight devices using a TCP/IP based communication, while Agent Factory Micro Edition (AFME) allows the execution of a deliberative agent on top of mobile phones with CLDC/MIDP profiles and SunSPOT sensor motes by means of TCP/IP and Zigbee protocols. But also, the interoperation among different agent solutions is difficult to achieve. For example, Jade-Leap agents and AFME agents cannot communicate, even if using the same communication protocols, nor does AFME support the communication between agents in mobile phones, neither with desktop computers, nor with agents in SunSPOT sensor motes. In addition, current agent infrastructures do not provide explicit support for the dissemination of data to a group of related IoT nodes. The agent has to maintain itself, the set of IoT nodes interested in their data, and disseminate, through individual messages, new data when it is available. This solution can be a resource-consuming task when the number of interested target

agents is high, and also complicates agent implementation.

Our approach addresses these limitations of agent technology in the IoT at two different levels: (i) improving the design of a communication subsystem inside the agent architecture (i.e. agent level) to facilitate the reconfiguration of an agent communication mechanism to adjust it to different contexts; and (ii) endowing the agent infrastructure with the necessary means to manage interoperability limitations because of device and communication protocol heterogeneity, and extending the message transport service (MTS) provided by the agent infrastructure to support an efficient group communication (i.e. the AP level). At the agent level, we provide agents with the capacity of self-configuring their internal architecture in order to use different communication protocols, taking into account the context and the necessities of the application. This flexibility inside the agent's internal design also makes easier the simultaneous use of different message distribution mechanisms easier. In this paper we focus on the agent infrastructure level, presenting the design and implementation challenges of an AP, named *Sol*, which facilitates (1) the communication and interoperation of agents running in heterogeneous set of devices (such as SunSPOT sensor motes, Android-based lightweight devices and other mobile phones) and even using different communication protocols; (2) group message delivery (one-to-many communication) in an efficient manner. The *Sol* AP supports the native communication protocols of each device (e.g. ZigBee, WiFi) and acts as a gateway, performing specific functions in order to ensure interoperability. In order to address the efficient group message distribution, the *Sol* AP provides support for membership management (joining and leaving members) and adequate communication mechanisms when possible (e.g. using IP multicast). We illustrate the benefits of our approach with several scenarios of an Intelligent Museum, and show its feasibility in terms of the response time of reconfiguration, and wireless data exchange, so important in the IoT

The paper is organized as follows: Section 2 presents a case study and the motivation behind our work and gives details about the implementation of the *Sol* AP and section 3 shows some results that validate our proposal. The paper concludes with a section of discussion and work and some conclusions.

2 An agent platform for IoT applications

2.1 Case study and motivation/contribution

In order to illustrate how to use the AP, the IoT scenario we have chosen for a case study is an Intelligent Museum (IM). These kinds of buildings normally include a considerable number of sensors and personal lightweight devices and mobile phones, spread throughout the halls and rooms of the museum, which provide context-aware data and services. Sensors provide data and services that help museum guides and security staff in their work, and additionally, brings location based services to visitors. The IoT system has different services for their target users: in the case of the guide, it provides support for the organization of the route inside the museum considering the presence of other groups in the halls and rooms of the museum, and helps share information between the guide and him/her group of visitors; in the case of the security staff, the IoT nodes (sensors and personal

lightweight devices) provide information on the presence of people in the museum and environmental conditions (temperature, light, humidity,...) and sends global notifications to the different groups of people that are in the museum; and in the case of visitors, we take advantage of the fact that most people usually bring a mobile phone with them in order to provide location based information for example details about an exhibit. This IoT scenario is designed as a Multi-Agent System (MAS) whose agents are located in devices that people (visitors, guides and security staff members) bring with them (such as tablets, smart phones and mobile phones) and inside sensors located in the building. The MAS has four types of agents: one for the guides (*GuideAgent*), another for security staff members (*SecurityAgent*), another for visitors (*VisitorAgent*) and another for the sensors (*SensorAgent*), each one running in a specific device. These agents are deployed in the AP named *Sol*, presented in this paper, that runs in a desktop computer.

The set of communication scenarios and devices involved will help us to illustrate how our approach deals with the challenges raised in the introduction, which can be summarized as:

C1-Device Heterogeneity. We need to have intelligent agents able to be executed in any typical IoT device, which can range from sensors to hand-held devices. Additionally, it is necessary to have highly reconfigurable agents able to adapt to the hardware and software resources available in each device.

C2-Communication Heterogeneity. In this heterogeneous environment it is necessary to provide the technological means to enable the communication between agents that do not support the same transport protocols.

C3-Flexible Communication Infrastructure. Nowadays, the appearance and disappearance of new end systems and wireless communication technologies is becoming usual. In recent years we have seen how some of them have became obsolete (such as IrDa[1]) while new ones have appeared (e.g. NFC[2] and Wi-Fi Direct[3]). However, it is rather complex to extend current agent architectures to support new capabilities, which is not straightforward and depends on the programmer's expertise on certain agent architecture. More flexible agent architectures should allow software agents to cope with the evolution and emergence of new technologies e.g. new transport services, message encoding, etc. So, it is desirable to have both agent architectures and infrastructures (i.e. AP) easy to extend with new communication means, including the possibility to enable or disable these mechanisms even at runtime and to use them simultaneously when needed.

C4-Efficient Group-based Data/Message Distribution. Many of the applications and services deployed in the IoT require the dissemination of data to a set of group-related nodes. In practice, the realization of this type of communication one-to-many has to be performed by the agents themselves. Each agent has to manage and maintain the list of IoT nodes interested in its data, and disseminate, through individual messages, new data when it is available. This solution can be a resource-consuming task when the number of interested target agents is high, and also complicates agent implementation.

[1] http://en.wikipedia.org/wiki/Infrared_Data_Association
[2] http://en.wikipedia.org/wiki/Near_Field_Communication
[3] http://en.wikipedia.org/wiki/Wi-Fi_Direct

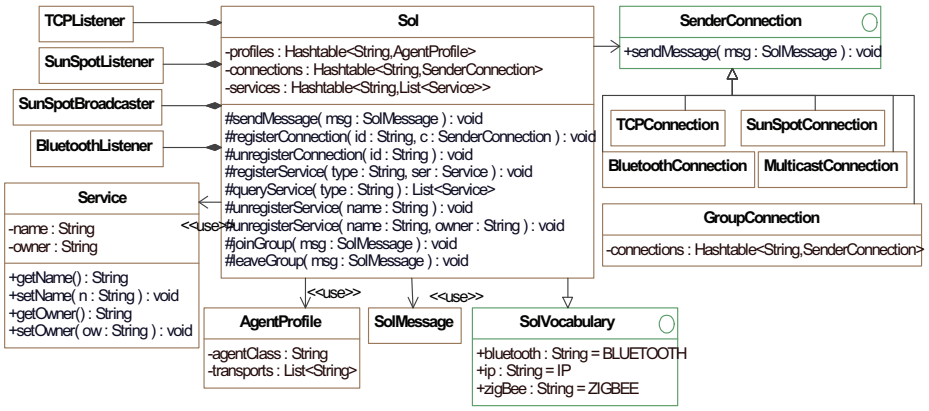


Figure 1: UML class diagram of the *Sol* agent platform.

2.2 The *Sol* agent platform

The *Sol* AP (see fig. 1), partially implements the FIPA[4] abstract architecture for lightweight devices. The main goal of the *Sol* AP is to support the interoperability of agents deployed in different IoT devices, through heterogeneous communication protocols. The current version of the AP works with a wireless personal area network (WPAN) mainly composed of MIDP/CLDC phones, Android devices and SunSPOT sensor motes. This AP acts as a middleware that provides a set of services to the agents that are deployed on it, and behaves as a gateway to support communication heterogeneity (fullfils C2). Specifically, our AP supports: (1) the registering and discovering of agents (Agent Management Service-AMS); (2) the registering and discovering of services (Directory Facilitator-DF); (3) The registration and membership of groups (Group Management Service - GMS); and (4) message communication service (MTS), which allows the communication between agents registered in the AP, extended to facilitate the distribution of group-based communication. The distribution of communication messages is supported internally by the Internal Platform Message Transport (IPMT), which resolves communication interoperability issues. Note that the AMS, DF and MTS are classical services provided by any AP. However, we have extended the MTS to support an efficient group communication, which is complemented with the GMS.

The internal design of *Sol* AP is shown in fig. 1. The main class (named *Sol*) supports the services enumerated below and stores information about the agents and groups deployed in the AP and the services provided by agents that are signed up in the MAS. However, agents do not interact with this class directly to access those services. Instead, all the interaction between the AP and the agents attached is ACL message-based. The ACL messages are represented in a special string-based format named *SolMessage* (see fig. 1).

Once an agent starts its execution, its first interaction is to join the AP (i.e. register in the *Sol* AP). Requests for registration are attended to *Listeners*. The AP

[4] http://www.fipa.org/

provides specific listeners for different protocols and technologies (TCP, SunSpot, Bluetooth listener in fig. 1). Internally, these classes instantiate threads which have sockets to listen to requests from handheld devices (Android enabled devices and mobile phones with MIDP profile) and SunSPOT sensor motes. Agents send a request message to register into the AP through this listener. In the registration message they specify its type, its identifier and the set and type of transport protocols (e.g. Bluetooth) supported. This information is stored in an instance of the *AgentProfile* class. Agent profiles are stored in a hash table (attribute *profiles*) indexed by the agent identifier. This data will be used for helping the agent access the MTS. Additionally, the *Sol* AP adds this agent to a group composed of all the agents with the same type. Both MTS access and groups are described further on in this section.

The DF provided by the *Sol* AP supports the main functions of a FIPA DF. This service is also supported by a set of specific listener classes. Agents may register their services with the DF or query the DF to find out what services are offered by other agents. Service descriptions provided by agents at the DF registration are stored in an internal hash table (*services* attribute in fig. 1). In the IM, each agent registers the services that it can provide to the system. For example, a *SensorAgent* that can measure and provide data on acceleration, luminosity and temperature, it registers in the AP as a provider of these data sensing services. So, if another agent needs to monitor luminosity, it queries the DF of the *Sol* AP to find out the identifiers of the agent providers of that service. Finally, it sends a request for data to the sensor agent, which periodically sends *inform* messages with data on luminosity.

Another of the contributions of *Sol* is the support for different communication paradigms and technologies. Specifically *Sol* supports peer to peer communication (the usual communication paradigm in FIPA compliant MAS) and multicast communication, which is often required by ubiquitous systems. Multicast communication facilitates the distribution of the same information to clustered components of the system (achieving C4). In order to introduce this kind of communication in the MAS the *Sol* AP incorporates the concept of group. A group is formed by a set of agents that share features. By default, there is a group for each type of agent that comprises the system, but additionally the user can define its own groups taking into account the role that the agent plays in the MAS and the application communication needs. For instance, since it is usual to communicate simultaneously with a set of service providers, we can define a group to include all the agent providers of the same service. A group is defined to facilitate the propagation of information between agents, and to make service provision more efficient. For example, in the IM we can define a group for all the agents that monitor a specific room. Although these agents monitor and provide different data (e.g. presence or humidity), they are situated in the same room, which is considered the common feature used to define the group (i.e. their position in the museum). The definition of this group is useful for the IM, for example for a *SensorAgent* which could send a message *room empty* to the group of agents present in the room. With this information the agents of the group could decrease their activity in order to save energy.

The AP provides a GMS for the creation, joining and leaving multicast groups. Agents request joining to a group, usually as part of the IoT application functional-

ity. If it is the first member of the group, then a new group is created represented by an instance of *GroupConnection*. The joining of an agent to a group depends on its profile (*AgentProfile* class in fig. 1). If the agent supports TCP/IP and multicast IP, a *MulticastConnection* is created (whether there is not a *MulticastConnection* in this group) and added to the *GroupConnection*. A *MulticastConnection* has assigned a multicast IP address. Once the joining process ends, the AP sends a message to the agent that includes the IP multicast address and port (how the agent uses this information to complete the joining process is described in Section 2.3) where group messages are sent. If the agent does not support TCP/IP (or multicast IP) then the individual (and preferred) agent transport address (represented by an implementation of *SenderConnection* interface) is included in the group connection object. In the case of Bluetooth, an active *BluetoothConnection* object is added to the corresponding *GroupConnection* (if the connection is not established, then it is created). Finally, agents that communicate using Zigbee do not require any special type of connection (i.e. *MulticastConnection*) for multicasting because they communicate using UDP datagrams.

Coming back to our case study, consider a group of *TouristAgent* which is visiting the museum with a guide. All of them form a group in the *Sol* AP, which will be useful for the guide to distribute contents to his/her tourist group. Each tourist may have a different kind of device, many of which probably do not include a Wi-Fi interface. Also, for some of them, Bluetooth would be the preferred communication option. For those tourist devices supporting Wi-Fi,the preferred option for receiving additional guide contents will be IP multicast, which provides better times (as we show in Section 3).

The MTS delivers messages between agents registered in the *Sol* AP. All the agents have access to at least one specific MTS provided by the AP. The MTS of *Sol* is supported by a set of connections. For each agent that is registered in the AP, the AP maintains a connection. After registering the agent through the AMS, a connection between the agent and the AP is established using the technology access that has been detailed at the registration stage. A connection is supported by concrete implementations of the *SenderConnection* interface. All the active connections are stored in a hash table. For each connection, the information stored is the identifier of the agent or the group in the AP and a class that implements the interface *SenderConnection*. The AP supports five types of connections (i.e. implementation of the interface): *TCPConnection*, *BluetoothConnection*, *MulticastConnection*, *SunSpotConnection* and *GroupConnection*. *TCPConnection* and *BluetoothConnection* are used by agents running in handheld devices to send and receive messages by means of TCP sockets and Bluetooth connections. These devices can receive multicast messages using UDP sockets by means of *MulticastConnection* objects. The *SunSpotConnection* is used to send UDP datagrams in order to communicate with SunSPOT sensor motes. Finally, *GroupConnection* represents a group of devices and stores an internal list of connections (*connections* attribute), which can refer to the other types of connections that have been described. With this design it is easy to add new devices and communication protocols to the MTS of the AP, since we only have to implement a listener (for the AMS and the DF service) and the *SenderConnection* interface for the new type of connection or device specific communication mechanism. The features that are

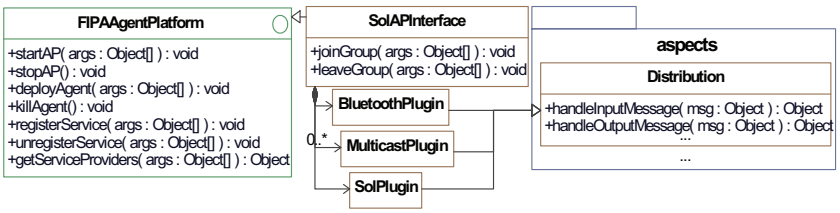


Figure 2: UML class diagram of the *SolPlugin*

described at this point address the C2 challenge presented in Section 2.1 because our AP has support for heterogeneous communication means.

In addition, we have already checked how this design meets challenge C3: The case of SunSPOT sensor motes is special and different to Bluetooth and classical TCP/IP because these devices connect to the *Sol* AP by means of the so called SunSPOT base station. The problem is that each time the base station is plugged into the system it is bound with a different IPv6 address, which must be known by the sensor motes to connect to it. Therefore an initial discovery process is necessary. This discovery process is implemented in the *SunSpotBroadcaster* class, which is a thread that periodically sends broadcast messages with the IPv6 address of the base station. Finally, *SunSpotListener* is an UDP socket to listen/sending datagrams from/to SunSPOT sensor motes.

As mentioned before, the distribution of messages to groups is also implemented as another type of connection (*GroupConnection*). *GroupConnection* has an internal hash table of *SenderConnection* implementations. The reason is that, although the best way to multicast a message to a group is to use a multicast IP address, we can not be sure that all the devices in the group support TCP/IP, UDP or IP multicast. So, although the *Sol* AP defines a IP multicast group, which uses it to multicast the message, the AP sends an unicast message to the groups members that do not support IP multicast. In this way, we still fulfill C2.

The *Sol* AP IPMT supports Bluetooth RFCOMM, UDP and TCP protocols at the transport level. It functions as a bridge since it resolves the differences in the communication protocols at the data link level. Sensors are normally connected forming a wireless sensor network (WSN) using the IEEE 802.15.4 standard (also known as ZigBee), while hand held devices use WiFi (802.11) and Bluetooth (802.15.1) as the access technology. So, *Sol* allows SunSPOT sensor motes to communicate with each other and with the so-called SunSPOT base station using ZigBee, but it also allows them to send and receive data from hand held devices behaving as a gateway. Apart from the differences of the data link level protocol, the *Sol* AP implementation has to deal with the limitations imposed on the TCP protocol implementation by the 802.15 network interface, especially those related with the number of active connections.

2.3 The communication concern inside the agent

The basis to achieve challenge C3 is an agent architecture that endows software agents with enough flexibility to communicate using different access technologies

and communication protocols (even simultaneously). By doing this, we solve the interoperability issue as part of the agent architecture, which gives more control to the agent developer to adapt the agent to operate and communicate through any network interface and communication protocol supported by the device. The AP presented has been designed to work with a family of agent architectures called Self-StarMAS [6]. Prior to describing the mechanism inside the agent to communicate using the *Sol* AP, some details of Self-StarMAS are given. The main feature of the internal architecture of a Self-StarMAS agent is that it is composed of a set of components and aspects[5] that helps to keep the application specific functionality from the communication-related concerns separate. These concerns, which are encapsulated as aspects, are the formatting of messages (*Representation* aspect) and the distribution of the messages using different communication means (*Distribution* aspect), among others. The advantages of separating these aspects can be found in [3]. The most important benefit of the aspect orientation is that it enhances the internal modularization of the system, defining loosely coupled architectures, that are easy to reconfigure even at runtime. Our agents can use different distribution aspect implementations to distribute messages, which can be changed or used simultaneously whenever it is necessary (achieving C3). The Self-StarMAS is therefore considered a family of self-configuring agent architectures for lightweight devices. The different versions of Self-StarMAS are able to be executed in Android devices, mobile phones with MIDP/CLDC profile, desktop computers and SunSPOT sensor motes (achieving C1). Additionally, the current implementation of Self-StarMAS provides implementations of the distribution aspect for Bluetooth, the Jade-Leap AP and for the *Sol* AP, which is the subject of this section.

The distribution aspect has the same design for the agents running in the different devices (sensors, hand held devices and desktop computers). This aspect allows the agent to communicate by means of different MTSs through the *Sol* AP. It receives the incoming messages and delivers outgoing messages through a specific network access technology (Bluetooth or Wi-FI) or communication protocol (TCP, UDP). This aspect hides communication dependencies defining a high-level interface to send and receive messages to and from different communication technologies. For each network interface and protocol the agent can access, this aspect is in charge of instantiating the corresponding device and API-dependent functionality. For example, in SunSPOT sensor motes use UDP sockets instead of the TCP sockets used by the other devices. Additionally, they do not need a special class for receiving messages from groups because new instantiations of the UDP socket for multicasting can be used for communicating with *Sol*. Moreover, these devices require a discovery process, while other implementations do not. Finally, the Bluetooth side of the AP is not included for SunSPOT sensor motes and there are some differences between the implementations for Android devices and mobile phones with MIDP/CLDC profile, mainly related to how the API sets and establishes the connection.

Internally, this communication concern is divided in two parts (see fig. 2), one to access the AMS, DF and GMS services of the platform (*SolAPInterface*) and the other for the communication through the platform (*SolPlugin*, *MulticastPlugin* and *BluetoothPlugin*). The *SolAPInterface* realizes the *FIPAAgentPlatform* inter-

[5] `http://www.aosd.net`

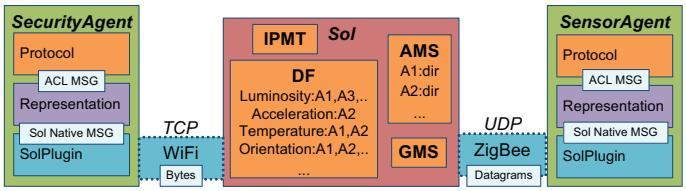


Figure 3: Schema of the communication in *Sol* agent platform

face. This interface is common to all the distribution aspects and allows a uniform access to FIPA compliant APs. The *SolAPInterface* extends the services of *FIPAAgentPlatform* with services to allow the joining and leaving of groups.

The *SolPlugin* and the *BluetoothPlugin* classes extend the *Distribution* aspect and permit sending messages to the *Sol* AP using a specific transport or access technology. They are implemented as threads that listen to messages. In the case of the *SolPlugin*, from TCP socket connection established with the *Sol* AP, and in the case of the *BluetoothPlugin* from a RFCOMM Bluetooth connection to a service also running in the AP. The case of *MulticastPlugin* is special because it is a thread only for listening to messages targeted at multicast group. As stated before, agents can ask the *Sol* AP to join a group and this is done by means of the *SolAPInterface*. If the technology for the connection is IP then the *Sol* AP answers the request with a multicast IP address and a transport port. With this information a new *MulticastPlugin* is instantiated and added to the agent architecture. If an agent wants to send a message to a group, this is done via *SolPlugin* or *BluetoothPlugin* and using as receiver the identifier of the group. When the message arrives in *Sol*, this sends the message using the corresponding *GroupConnection* as described in Section 2.2.

As stated before, aspect orientation gives the Self-StarMAS agents the possibility of using different APs and mechanisms for communication. But as a counterpart it also requires the transformation of the messages during sending and reception in order to compose messages in the format used by the underlying AP. This task can be defined as a translation, which is encapsulated in the *Representation* aspect [3]. For our case study, *SensorAgent* and *SecurityAgent* are involved in an interaction ruled by a protocol that consists of *SensorAgent* periodically sending reading light to the *SecurityAgent*. The *SensorAgent* sends an ACL message that is transformed by the *Representation* aspect to a *SolMessage* and is sent to the *Sol* AP as a datagram by *SolPlugin*. When the message arrives at the AP, this is sent by means of the *TCPConnection* that sends the message as a stream of bytes to the *SecurityAgent*.

3 Validation

As stated before, the AP presented here has been designed to support the interoperation of a family of self-configuring agents called Self-StarMAS. These agents can accomplish different tasks for self-configuring: (T1) change the sampling frequency of a monitoring component/aspect; (T2) change the distribution aspect at

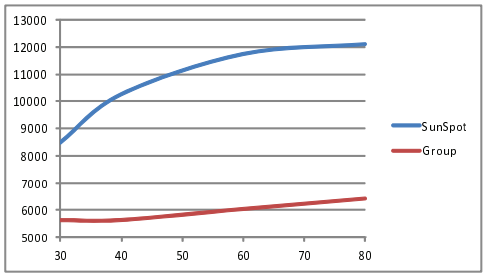


Figure 4: Times (in milliseconds) for the T3 task for different numbers of agents.

runtime; and (T3) require a new service provider of data in case of failure of the service provider. T1 and T2 are related with the internal architecture of the agent and are beyond the scope of this paper, but T3 is directly related with the work of the *Sol* AP presented here. In [5], we evaluate this task considering that the agent each time it loses the service provider, queries the DF of the *Sol* AP in order to get a new service provider. Then, it sends request messages to all the service provider agents that can provide the required service and chooses one sending a confirm message and discarding the others sending a refuse message.

In this section, the set of service provider agents will be replaced by a group (*GroupConnection*, see section 2.2) and we see the benefits of the group mechanism comparing it with the classical unicast messaging (*TCPConnection*, *SunSpotConnection* or *BluetoothConnection*). Specifically, we compare one *GroupConnection* with a set of *SunSpotConnections* to see benefits for the T3 self-configuring task. The tests have been performed in a Samsung Galaxy Nexus[6] mobile phone and SunSPOT sensor motes. All the experiments has been repeated fifty times and the average and the standard deviation has been calculated.

In order to evaluate T3 with the two communication mechanisms (a single *GroupConnection* and a set of *SunSpotConnections*), we have deployed different scenarios of the IM, mainly composed of one *SecurityAgent* (executing in an Android enabled phone) and a variable number of *SensorAgents*. The number of *SensorAgents* can vary from just one sensor (for instance providing luminosity data) to various tens. All the *SensorAgents* of the system are registered as service providers of the specific service which fails and causes the T3 reconfiguration. The average of the results of this experiment are depicted in figure 4 and the standard deviation is approximately 2 seconds for all cases. The results show that time for self-configuring scales up with the number of agents following a logarithmic curve in the case of the set of *SunSpotConnections* (*SunSpot* label). However, the resulting times are affordable and the scalability of the self-configuring task using this mechanism is good, the benefit of using the mechanism for group-based communication (*Group* label) is clear. The time using the *GroupConnection* is lower in all the scenarios. Additionally, note that when the number of sensors and agents increases by more than 250%, the time required for self-managing increases 140% in the case of the set of *SunSpotConnections* and 120% in the case of the *GroupConnection*.

[6] `http://www.google.es/nexus/`

4 Related work

There are different approaches to develop IoT applications, the majority of them based on middlewares [4], services [8] or frameworks [13]. These solutions, as does the work presented here, integrate technologies such as RFID, smartphones or sensors and offer mechanisms to build applications based on them. Mainly, they offer a transparent access to the remote resources of the system alleviating the problematic of building distributed applications or in other cases they provides a communication infrastructure for heterogeneous devices. An interesting approach in this trend is the MundoCore middleware [2], that not only provides a transparent access to services, additionally it supports the communication using different transport protocol (e.g. TCP/IP, Bluetooth,...) and communication paradigms (e.g. peer to peer, publish/subscribe,...). These approaches are different to agent approaches, since they support different application layers. Middleware are at the infrastructure level, while agent-based solutions are at the application level and can use services provided by the middleware in order to fulfill their goals. The work presented here has similar goals to MundoCore middleware (support for communication and services) but adapted to the agent paradigm.

There are different agent approaches that can be used to develop IoT applications. However, there are no specific solutions for IoT systems, so this section mainly shows agent approaches for AmI/ubiquitous computing area, which can be used for the development of IoT systems. Among these approaches, we can find agent technologies that support heterogeneous devices and approaches for a single type of target device. In this section we are going to show and compare our approach with some of them.

One of the most important approaches that tries to tackle the heterogeneity in MAS is Agent Factory Micro Edition (AFME) [14, 15]. This is a framework that seeks to address the constraints and heterogeneity of AmI environments. AFME provides a set of tools that support the execution of a deliberative agent architecture that are able to execute on desktop computers, mobile phones with CLDC/MIDP profile and SunSPOT sensor motes. However, this approach does not support the communication between agents using TCP/IP based communications (agents in mobile phones and desktop computers) and agents that uses ZigBee (agents in SunSPOT sensor motes), as our approach does. Jade-Leap [7] is a lightweight implementation of the Jade AP for desktop computers, mobile phones with MIDP/CLDC profile and Android enabled devices. This proposal has been used in different AmI projects [11, 17], but does not have support for more lightweight devices such as sensors, so important in IoT environments. Another approach that has tried to tackle the heterogeneity of ubiquitous environments is 3APL-M [10] that enables the execution and communication of BDI agents in mobile phones and desktop computers. Java Standard Edition (J2SE) has the connectivity technology to develop IoT applications based on TCP or UDP and additionally, it can be executed in desktop computers and more lightweight devices such as PDAs. Therefore, agent approaches implemented in J2SE can be used to develop IoT applications based on agents for a limited set of heterogeneous devices, this is the case of Jack [9] and FIPA-OS [16].

As stated before, there are agent technologies that support one single type of

target device but they are useful for the development of IoT systems because their target devices are typical of IoT systems. This is the case of Andromeda [1], an agent approach only for Java for Android and the case of other approaches for SunSPOT sensor motes, such as MAPS or MASPOT [12].

Finally, our AP has support for agents running on desktop computers, mobile phones with CLDC/MIDP profile, Android enabled devices and SunSPOT mote sensors, which makes this approach the agent proposal which covers the greatest number of devices. Moreover, our AP enables the communication between agents that use different transport protocols, which is not the case of other similar approaches.

5 Conclusions

The IoT envisions a world in which heterogeneous set of devices (i.e objects) are interconnected and collaborate using the Internet in order to provide valuable services for users. For the developer, the deployment of applications and services for the IoT requires managing a heterogeneous set of devices, communication protocols and underlaying networks, in order to resolve interoperatility issues due to the heterogeneity of the IoT nodes. More than other technologies, agents seem to have the necessary characteristics to support the development of applications and services for the IoT. However, agent technologies have to address a set of challenges in order to be used to develop IoT systems. They have to support device and communication heterogeneity (C1 and C2), to have a flexible communication infrastructure (C3) and to have efficient group-based data multicast (C4).

To address these challenges, our approach defines the *Sol* AP, which facilitates (1) the communication and interoperation of agents running in a heterogeneous set of devices (such as SunSPOT sensor motes, Android-based lightweight devices and mobile phones) and even using different communication protocols; (2) group message delivery (one-to-many communication) in an efficient manner. The *Sol* AP supports the native communication protocols of each device (e.g. ZigBee, WiFi) and acts as a gateway, performing specific functions in order to ensure interoperability. At the agent level, we provide agents with the self-configuration of their internal architecture in order to use different communication protocols taking into account the context and the necessities of the application. This flexibility inside the agent's internal design also makes the simultaneous use of different message distribution mechanisms easier.

We have illustrated the benefits of our approach with several scenarios in an Intelligent Museum, and we have shown the feasibility of this approach in terms of the response times of reconfiguration, and wireless data exchange, so important in the IoT. In previous works [5, 6] we have presented and validated the internal architectures of our self-managed agents for heterogeneous devices, while the presented work has focused in the underlying AP. As future work, we plan to extend the *Sol* AP to more lightweight devices such as Tiny OS sensor motes and validate our proposal with more experiments and application scenarios.

Acknowledgments.

This work has been supported by the Spanish Ministry Project RAP TIN2008-01942 and the regional project FamWare P09-TIC-5231.

References

[1] J. Agüero, M. Rebollo, C. Carrascosa, and V. Julián. Model-driven development for ubiquitous mas. In *ISAmI 2010*, volume 72 of *Advances in Intelligent and Soft Computing*, pages 87–95. 2010.

[2] E. Aitenbichler, J. Kangasharju, and M. Mhlhuser. Mundocore: A light-weight infrastructure for pervasive computing. *Pervasive and Mobile Computing*, 3(4):332 – 361, 2007.

[3] M. Amor and L. Fuentes. Malaca: A component and aspect-oriented agent architecture. *Inf. Softw. Technol.*, 51(6):1052–1065, June 2009.

[4] L. Atzori, A. Iera, and G. Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[5] I. Ayala, M. Amor, and L. Fuentes. Self-configuring agents for ambient assisted living applications. *Personal and Ubiquitous Computing*. Accepted for publication.

[6] I. Ayala, M. Amor, and L. Fuentes. Self-management of ambient intelligence systems: a pure agent-based approach. In *Proc. of AAMAS*. IFAAMAS, 2012. To appear.

[7] F. Bergenti and A. Poggi. Leap: A fipa platform for handheld and mobile devices. In *Intelligent Agents VIII*, volume 2333 of *LNCS*, pages 436–446. 2002.

[8] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio. Interacting with the soa-based internet of things. *IEEE Transactions on Services Computing*, 3(3):223 –235, july-sept. 2010.

[9] N. Howden, R. Rnnquist, A. Hodgson, and A. Lucas. Intelligent agents - summary of an agent infrastructure. In *Proc. of AAMAS*, 2001.

[10] F. Koch, J.-J. Meyer, F. Dignum, and I. Rahwan. Programming deliberative agents for mobile services: The 3apl-m platform. In *PROMAS*, volume 3862 of *LNCS*, pages 222–235. 2006.

[11] T. C. Lech and L. W. M. Wienhofen. Ambieagents: a scalable infrastructure for mobile and context-aware information services. In *Proc. of AAMAS*, pages 625–631, New York, NY, USA, 2005. ACM.

[12] R. Lopes, F. Assis, and C. Montez. Maspot: A mobile agent system for sun spot. In *10th ISADS*, pages 25 –31, march 2011.

[13] M. Mitchell, F. Sposaro, A.-I. Wang, and G. Tyson. Beat: Bio-environmental android tracking. In *2011 IEEE RWS*, pages 402 –405, jan. 2011.

[14] C. Muldoon, G. O'Hare, R. Collier, and M. O'Grady. Agent factory micro edition: A framework for ambient applications. In *ICCS*, volume 3993, pages 727–734. 2006.

[15] C. Muldoon, G. O'Hare, M. O'Grady, and R. Tynan. Agent migration and communication in wsns. In *Proc. of PDCAT 2008.*, pages 425 –430, dec. 2008.

[16] S. Poslad, P. Buckle, and R. Hadingham. The fipa-os agent platform: Open source for open standards. In *Proc. of PAAMS*, volume 355, page 368, 2000.

[17] N. Sánchez-Pi, J. Carbó, and J. Molina. Jade/leap agents in an aml domain. In *Hybrid Artificial Intelligence Systems*, volume 5271 of *LNCS*, pages 62–69. 2008.

[18] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, and P. Doody. Internet of things strategic research roadmap. *Internet of Things: Global Technological and Societal Trends*, page 9, 2009.

An Approach for the Qualitative Analysis of Open Agent Conversations

Emilio Serrano[1], Michael Rovatsos[2], and Juan A. Botía[1]

[1] *University of Murcia* *, {emilioserra,juanbot}@um.es

[2] *University of Edinburgh,* michael.rovatsos@ed.ac.uk

Abstract

This paper presents an approach for the qualitative analysis of data obtained from past communicative interactions in an open multiagent system. Such qualitative analysis focuses on the use of high-level agent communication languages to infer theories about agents with mental states which are normally not accessible for the outside observer. The inference of these theories, or context models, is based on logging semantic data available from protocol execution traces and using this information as samples for the application of data mining algorithms. These context models can be applied both by system developers and agents themselves at run-time for various tasks, e.g. to predict future agent behaviour, to support the process of ontological alignment in communication, or to assess the trustworthiness of agents. An implementation of the approach presented is also given, the ProtocolMiner tool, which automates the building of context models from arbitrary protocol executions.

Keywords: Agent communication languages, interaction protocols, interaction analysis, data mining, agent-oriented software engineering.

1 Introduction

The interaction among autonomous, rational agents is one of the essential characteristics of a multiagent system (MAS). For this reason, most MAS platforms offer various mechanisms to support such interaction, including an infrastructure for sending and receiving messages, establishing conversations, following interaction protocols, sharing vocabularies using ontologies, etc.

Most of the analysis tools included in MAS development frameworks fall into two categories: (1) analysis of agents' mental states, and (2) analysis of the interactions among agents. The analysis of mental states usually assumes that the

*Acknowledgments: This research work is supported by the Spanish Ministry of Science and Innovation under the grant AP2007-04080 and in the scope of the Research Projects TSI-020302-2010-129, TIN2011-28335-C02-02 and through the Fundación Séneca within the Program 04552/GERM/06. Facultad de Informática, Campus Universitario de Espinardo, 30100 Murcia, Spain.

agents' implementation is known and available. This constraint is very restrictive in open MAS that run on interoperable agent interaction platforms where different agents can be added to the system by different users at runtime. On the other hand, methods for interaction analysis often remain very superficial and address only fixed syntactic elements that can be observed in message exchanges (e.g. performative, sender and receiver, successful/unsuccessful completion of the protocol, etc). This paper covers these shortcomings employing the semantics of interactions to perform a *qualitative analysis* which is able to infer theories about agents with hidden mental states.

To appreciate the utility of a qualitative analysis, let us take the FIPA Contract-Net protocol as example [1]. This protocol describes an agent (the *Initiator*) who wants one or more agents (the *Participants*) to perform a task. In this protocol, there are a large number of analysis tasks that can be performed following a *quantitative analysis*: (1) obtaining the number of conversations in which each agent has participated; (2) conversations in which an error occurred in the flow of messages defined by the protocol; (3) number of agents which rejected a proposal; (4) number of tasks that a participant was unable to perform after committing to them; etc.

Although this quantitative analysis can be very useful for a developer, more interesting information is usually captured in the specific semantics of messages, i.e. *qualitative properties*. The developer of the Initiator agent, for example, may be interested in which tasks are usually rejected by agents or, more specifically, how the process used by a Participant to accept or reject a task is implemented (in the sense of a decision rule in the agent's reasoning mechanism). Of course, this information is hidden if the Participant agents' implementation is unknown by the Initiator. Nonetheless, considering a concrete past execution of the protocol, the Initiator agent can easily recognise if a particular task has been accepted or rejected by a specific Participant. What is more, after several executions of the protocol, the Initiator can generalise the individual cases to build a more general theory that explains participants' behaviours.

We call theories which allow a developer to perform a qualitative analysis *context models*. As illustrated in the example, these models correlate the status of logical constraints attached to interaction protocol specifications to perceived agent behaviour. In other words, they map the conditions under which a certain behaviour occurs to the resulting behaviour itself.

Construction of these context models is based on capturing regularities in previously observed interactions by using data mining techniques. Context models are able to reveal implicit causal relationships between states of the system and the reasoning and decision-making mechanisms of all agents involved. These models can be used for various purposes: (1) to make predictions about future behaviour; (2) to infer the definitions other agents apply when validating logical constraints during an interaction; and (3) to analyse the reliability and trustworthiness of agents based on the logical coherence of their utterances.

In addition to the definition, construction and use of the context models; the contribution of this paper is to present an approach to automatically generate these models and an implementation of this approach, the *ProtocolMiner* tool.

The remainder of the paper is structured as follows. After reviewing related

work in section 2, we introduce the formal approach in section 3. The ProtocolMiner tool is presented in section 4. Section 5 gives empirical results obtained in a case study. Finally, section 6 concludes.

2 Related work

Many tools for run-time multiagent systems analysis address the testing, debugging, validation or verification of these systems. For example, the *Tracer Tool* [9] provides a semi-automated solution for agent software understanding, using high-level agent concepts instead of detailled execution traces and programming data structures. The tool proposed in [14] for the JADEX agent platform can be used to verify the consistency of internal events and message declarations and to obtain an overall communication structure as a three-dimensional graph. The inspector tool [6] for the the *Agent Factory Agent Programming Language* (AFAPL) provides support for the inspection of the internal states of agents, and for monitoring the performance of the underlying agent components. Similarly, INGENIAS [8] provides a visual debugging tool to inspect agents' mental states. These approaches are, however, only suitable for systems in which the mental states of the agents can be inspected by the designer, i.e. effectively only for systems whose code has been disclosed *a priori*, or who have been designed by the user performing the analysis themselves.

In contrast to this, there are also methods aimed at design-time (static) analysis of multiagent systems properties such as MABLE [15]. This imperative programming language uses the SPIN model checker to automatically verify properties of the system. While valuable, these approaches can only verify certain properties of agent interactions (based on observed interactions or on design-time specifications). However, they cannot derive any *additional* and *explicit* knowledge about the emergent behaviour of the system apart from whether agents are behaving correctly or not.

The only exception to this is some work that has recently emerged in the area of ontology matching [3, 4]. In these contributions, hypotheses about the possible meanings of unknown terms used by the other agent are filtered based on structural knowledge of the protocols. This is achieved by either (1) looking at the ontological relationships between candidate concepts based on the terms that appear earlier in the same dialogue or in previous dialogues, or (2) by reasoning about the overall syntactic structure of the protocol. However, this kind of reasoning is only used to resolve ontological conflict. On the other hand, the approach presented in this paper is able to infer more general emergent properties of interactions.

The notable limitation of all (but the latter) existing work is that it does not consider semantic elements of interactions for analysis, e.g. the constraints used by the agents while they are executing protocols, and which cause a concrete protocol execution to unfold in a particular way. Also, they fail to induce compact, explicit theories about the ways in which interaction evolves in a system, and which could be useful for the design of adaptive agents.

3 Formal approach for a qualitative analysis

Beyond the presentation of the ProtocolMiner tool, which uses a specific protocol specification language, the analysis provided by this tool is based on a formal approach. This approach for qualitative interaction analysis is independent of the specific development platform used. This section formalises: how to define protocols that are semantically annotated; how to obtain a context model from the execution of these protocols using data mining techniques; and how to build the training data set for these techniques.

3.1 Defining a protocol and its context

In brief, our framework is based on defining a *protocol model* as a graph $G = (V, E)$. In this graph, each node $v \in V$ is labelled with a message $m(v) = q(X, Y, Z)$ with performative q (a string) and sender / receiver / content variables X, Y, and Z. Besides, each edge is labelled with a (conjunctive) list of (say, n) constraints $c(e) = \{c_1(t_1, \ldots, t_{k_1}), \ldots, c_n(t_1, \ldots t_{k_n})\}$. Each constraint $c_i(\ldots)$ has arity k_i, head c_i and arguments t_j which may contain constants, functions or variables (in general the label of an edge could be an arbitrary formula $\phi \in \mathcal{L}$ of a logical language $\mathcal{L}$). All variables that occur in such constraints are implicitly universally quantified. The framework also assumes that all outgoing edges of a node result in messages with distinct performatives, i.e. for all $(v, v'), (v, v'') \in E$, $(m(v') = q(\ldots) \wedge m(v'') = q(\ldots)) \Rightarrow v' = v''$. Therefore, each observed message sequence corresponds to (at most) one path in G by virtue of its performatives. Figure 1 shows an example protocol model in this generic format for illustration purposes.

The *semantics* of a protocol model G can be defined by looking at the pairs $\langle \pi, \theta \rangle$ which specify the path and variable substitution that any message sequence $\mathbf{m}$ corresponds to in protocol model G. With this, we can define the *context* of $\mathbf{m}$ as $c(G, \langle m_1, \ldots, m_n \rangle) = \bigwedge_{i=1}^{n-1} c(e_i)\theta$ where $G(\mathbf{m}) = \langle \pi, \theta \rangle$. Crucial to our view of qualitative interaction analysis is the assumption that for any observed message sequence $\mathbf{m}$, the conjunction of edge constraints described by the context $c(G, \langle m_1, \ldots, m_n \rangle)$ was logically true at the time of the interaction.

3.2 Obtaining a context model by data mining

The basic method for applying data mining methods to protocol interactions is as follows: Consider a protocol model G, and message sequences $\mathbf{m}$ obtained from past executions of G. Any such sequence can be translated to a pair $G(\mathbf{m}) = \langle \pi, \theta \rangle$ as defined above. This approach assumes that only sequences allowed by G occur (if necessary, G can be modified on the fly to accommodate unexpected messages by adding constraint-free edges and message nodes). Assuming that a set of such substitution-annotated paths are used as a training data set D, an inductive learning algorithm $L : \mathcal{D} \rightarrow \mathcal{H}$ can be used to map any concrete data set $D \subseteq \mathcal{D}$, where $\mathcal{D}$ is the set of all possible observations, to a learning hypothesis $h \in \mathcal{H}$ taken from the hypothesis space of the machine learning algorithm in question [10].

This paper proposes to *augment* the learning data by the logical context of the data samples, i.e. to include the logical formula $c(G, \mathbf{m})$ in the data samples, which

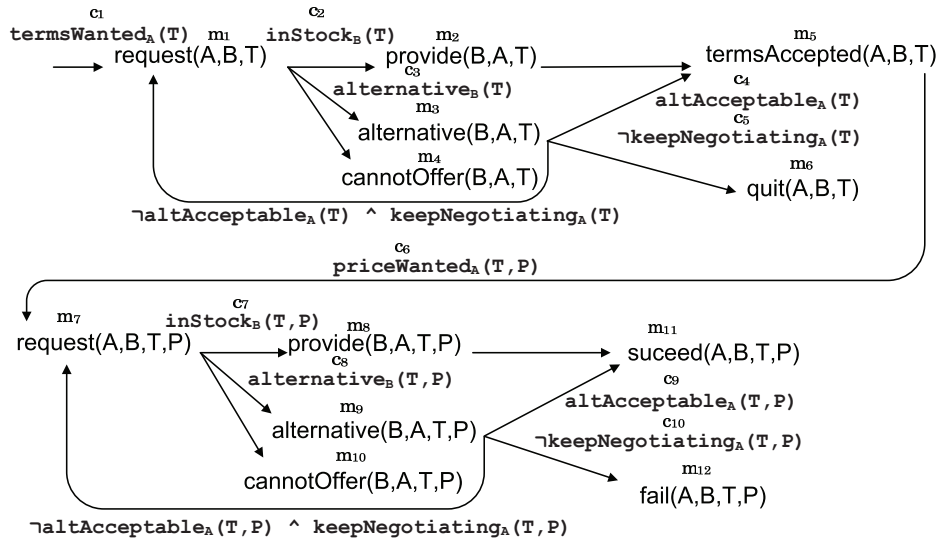


Figure 1: A simple negotiation protocol model: A decides the terms of a desired product, and requests T from B. The initial response from B depends on availability: if terms T cannot be satisfied, A and B go through an iterative process of negotiating new terms for the item, depending on the *keepNegotiating*, *altAcceptable*, and *alternative* predicates. In case of acceptance, the process of negotiation is repeated to decide the price P for the product. Edge constraints are annotated with the variable representing the agent that has to validate them (subscript A or B). Additional (redundant) shorthand notation c_i/m_j is introduced. Different out-edges represent XOR if constraints are mutually exclusive, and OR else.

can be directly inferred using the logical constraints provided by the definition of G. The model obtained with the learning algorithm using the context as training data is what we call a *context model*.

Determining the most suitable learning algorithms for a particular context mining task is beyond the scope of this paper, our method does not depend on the use of a specific algorithm. For example, figure 2 shows the context model for the constraint $altAcceptable_A$ of the protocol described in figure 1 when used in a car trading system. In this case, a decision tree algorithm has been employed to learn a specific constraint in the protocol.

3.3 Preparing the training data set

Due to the nature of multiagent interaction, additional design decisions have to be made before standard data mining machinery can be used, which are to do with the details of how exactly training data is constructed from raw protocol execution traces. We discuss these in the following sections.

```
persons = 2: F (158)
persons = 4: F (158)
persons = more
|   lug_boot = small
|   |   doors = 2: F (8)
|   |   doors = 3: F (7)
|   |   doors = 4: F (8)
|   |   doors = 5-more: T (105)
|   lug_boot = med
|   |   doors = 2: F (13)
|   |   doors = 3: F (8)
|   |   doors = 4: F (13)
|   |   doors = 5-more: T (120)
|   lug_boot = big: T (402)
```

Figure 2: Context model of the $altAcceptable_A$ constraint, obtained using the J48 decision tree algorithm after 1000 negotiations using the protocol described in figure 1. The notation `a =v : T/F` denotes that "if `a` has value `v` the target predicate has value `T/F`". Every leaf includes the number of instances classified under a certain path in parentheses.

3.3.1 Dealing with different agents

In defining the datasets to be used for protocol mining, one important design decision is how to deal with the presence of multiple agents. If all messages and contexts that occur in observed interactions were treated as features of learning samples this would amount to an attempt to derive globally valid interaction patterns. This learning strategy would imply that a shared theory regarding logical constraints and a shared ontological understanding of all terms used in communication exists among agents. In many cases, however, the purpose of interaction mining is to infer definitions of constraints or behavioural patterns that are specific to an agent or a group of agents.

To be able to make these distinctions, we need a method for filtering data obtained from protocol executions according to individual agents or groups of agents. Assume an assignment $\sigma : Var \to Ag$ where Var is the set of all variables occurring as sender/receiver variables in nodes of the graph, and Ag the set of agent names. Then for any agent $a \in Ag$, $V_\sigma(a)$ are the nodes that correspond to messages sent by agent a under role assignment σ, and $E_\sigma(a)$ are the *incoming* edges to those messages (formally, $E_\sigma(a) = \{(v, v') \in E | v' \in V_\sigma(a)\}$). We generalise these notions to $V_\sigma(A)/E_\sigma(A)$ for $A \subseteq Ag$ by taking the union over the respective sets for agents.

As an example, consider the protocol model depicted by the graph in figure 1. Assuming a set of agents $Ag = \{a_1, a_2\}$, a role assignment $\sigma = \{[A/a_1], [B/a_2]\}$, and the `request-provide-termsAccepted` path, $V_\sigma(a_1)$ would contain the `request` and `termsAccepted` messages and $E_\sigma(a_1) = \{termsWanted(T)\}$ as the only constraint of the incoming edges to utterances performed by a_1.

The most cautious form of data filtering in this setting would be to reduce the path π of every sample to those nodes and edges that pertain to the learn-

ing agent a_i. Only contextual information $c(G_{a_i}, \mathbf{m})$ from the restricted graph $G_{a_i} = \langle V_\sigma(a_i), E_\sigma(a_i)\rangle)$ would be considered because a_i can safely verify "own" constraints along π. With this strategy, all logical constraints verified by other agents are dropped. Note, however, that the path π and substitution θ used in the learning sample are still based on the full graph, as the observed messages were objectively perceived, i.e. $G(\mathbf{m}) = \langle \pi, \theta \rangle$.

At the other end of the spectrum, if a_i fully trusts the other agent(s) and can safely assume that all agents' ontologies and logical theories are fully aligned, it can use the entire path information as part of each learning sample. This strategy assumes that the definitions of constraints are common to all agents *and* that every agent verifies the constraints reliably and honestly.

3.3.2 Dealing with paths, loops, and variables

Standard data mining algorithms assume a fixed number of attributes (features) and values. Because of this, in our approach to qualitative analysis a number of issues arise that require certain further design decisions to be made.

Firstly, when collecting different paths for inclusion in the training dataset, their labels (messages/constraints) and the set of variables contained in these labels may differ. This is not a problem in principle, as data samples can be "padded" with "unknown" values for all messages and context constraints that do not occur in them, but this can be computationally wasteful. In many practical cases, it will be more appropriate to create a different data set for each observed path π. This is because any model learnt over such path-specific training data captures better the precise circumstances under which it occurs.

Moreover, at a domain-specific level, one can merge data across different paths into a single set while only observing a fixed set of certain messages and constraints. Different messages along the path can even be ignored introducing a single path label (or path group label) for each path to predict interaction outcomes. For example, an artificial boolean label *success* can be attached to a number of paths, effectively classifying different paths into two categories (where paths with *success* = *true* belong to the category of successful interactions, and all other paths are deemed unsuccessful).

Secondly, the results of the constraints functions in the context (but not the attributes in the arguments of these constraints) should be removed when the learning algorithm tries to predict the "outcome" value. This information is explicit in the definition of the protocol and is reflected in the path models it provides. Moreover, including these results may hinder learning techniques from relating the details of constraint argument values (note that the definition of the constraints in the protocol is still necessary) to the overall outcome of the protocol.

Thirdly, many common interaction protocols (e.g. negotiation protocols like auction and bargaining protocols) involve iterations of sub-sequences that can be repeated an arbitrary or number of times. The existence of a loop in a protocol means that variables occurring in a logical constraint or messages used in the loop can have several constants as ground instantiations in the same execution $\{g_1, g_2 ... g_n\}$, where n is the number of iterations in the loop. Moreover, n may vary depending on the number of iterations occurred in each run. Different strate-

```
1  a(participant, B) ::
2  request(X) <= a(initiator, A) then
3  (agree(X) => a(initiator, A) <- consider(X) then
4      (informDone(Y) => a(initiator, A) <- performed(X,Y))
5      or
6      (failure() => a(initiator, A))
7  ) or (refuse()  => a(initiator, A))
```

Figure 3: LCC implementation of the "participant" role in the FIPA Request Interaction Protocol

gies can be employed to obtain fixed-length samples for use in the training dataset. (1) If N is the maximum number of iterations observed in a protocol across various runs, N "copies" of each variable can be kept (with copies of messages and context predicates that contain it as attributes in the learning samples). This, however, introduces a lot of redundancy in paths with fewer than N iterations of the loop which will have to use a value of "unknown". (2) Considering only the first/last ground term g_1/g_n for a specific variable V in a loop. In many cases (e.g. negotiations) keeping two copies of each variable, one for the first and one for the last value will suffice as intermediate steps are less important for the outcome of the interaction.

4 ProtocolMiner

ProtocolMiner is a prototypical tool that provides comprehensive functionality for qualitative protocol mining. While the tool itself is designed for use by a human designer, an application programming interface (API) is also provided to allow agents to exploit emerging knowledge extracted from past interaction.

We first briefly introduce the definition of semantically annotated protocols that ProtocolMiner is designed to operate on. This is followed by details about how ProtocolMiner implements the formal approach described in Section 3.

4.1 Defining protocols in ProtocolMiner

The ProtocolMiner tool is a plugin for the OpenKnowledge platform [13], a protocol specification and execution platform designed for large-scale heterogeneous multiagent systems. ProtocolMiner automates the construction of context models. This includes the registration and recovery of the training data for any protocol implemented and executed in OpenKnowledge. OpenKnowledge uses a protocol definition language called the *Lightweight Coordination Calculus* or LCC [11], a language that uses declarative Prolog-like constraints in protocol specifications. As an example, an implementation of the "participant" role in the FIPA Request Interaction Protocol [2] in LCC is shown in figure 3. This specification defines a role `a(participant,B)` (binding the concrete agent identifier to variable `B` at runtime when the agent adopts the role) in terms of the message sequences allowed for that role. Incoming/outgoing messages are denoted by double arrows `<=` and

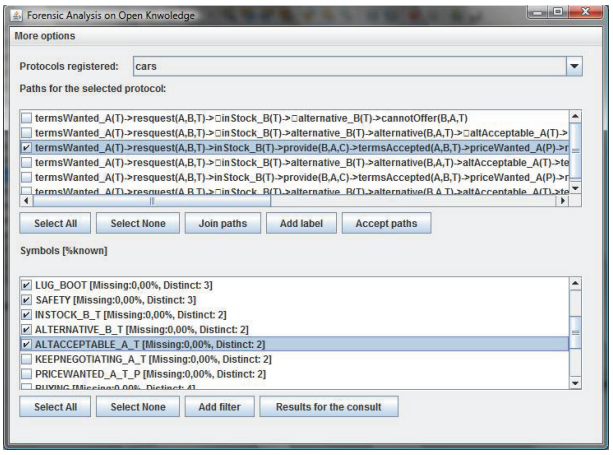


Figure 4: ProtocolMiner user interface

`=>` from/to another role identifier, and guards (preconditions) on messages appear on the right hand side of a message exchange, prefixed by a single arrow `<-` (the language also allows for postconditions prefixed by `->`, these are not used in the above example). Sequential concatenation, disjunction, and iteration are captured by keywords `then`, `or` and Prolog-like recursive calls of role clauses, respectively. In this specific example, the constraint $consider(X)$ is used to determine whether to `agree` or `refuse` the request for X.

Although ProtocolMiner has been implemented to use LCC based protocol specifications, other interaction platforms may be used as data sources as long as their specification mechanism can be translated to the protocol-model graph structure defined in section 3.1.

4.2 ProtocolMiner features

The ProtocolMiner user interface is shown in figure 4 after logging executions of the case study presented for evaluation purposes in the following section. As explained above, ProtocolMiner allows a developer to define a protocol which meets the requirements specified in section 3.1 using LCC language. ProtocolMiner also integrates the Weka [5] algorithms allowing the tool to use a large number of data mining algorithms to obtain the context models describes in section 3.2. The strategies to build a training data described in section 3.3 are also implemented in the tool. The ProtocolMiner GUI, shown in figure 4, allows a developer or agent to select the agents considered in the dataset through SQL expressions. For example, if the developer is only interested in agents a_1 and a_2, the *"add filter"* option can be used to introduce `A='a1' and B='a2'`. Additionally, the list of selected attributes in the dataset allows for arbitrary selections of subsets of constraints (or edges) to be considered by the data mining procedure. The tool also allows a developer or agent to select the paths considered in the data set and to label them with an "output attribute" by means of the option *"add label"*. Moreover, several paths that only differ in the number of iterations of a loop can be selected and merged

to a single path using the *"join paths"* option. Besides, the tool can be configured to (1) log all the values given to an attribute in a path (every value is stored in a new attribute), (2) to log only the first and last values along a path, or (3) to log only the last value given to an attribute at the end of a protocol execution.

5 Case study

To illustrate the usefulness of our approach, we have analysed data generated in a car selling domain, where agents negotiate over cars using the protocol shown in figure 1.

5.1 Description of the system under test

To make the discussion concrete, we apply our system to a well-known database for car evaluation [7]. This database includes the technical characteristics and prices which are used in the system. More specifically, a potential customer (role A) is requesting offers from a car selling agent (role B) where T specifies the technical characteristics including number of doors, capacity in terms of persons to carry, the size of luggage boot and estimated safety of the car. For the interactions analysed in the case study, we assume that the values for car characteristics are given as a tuple $T = (\mathit{doors}, \mathit{persons}, \mathit{lug\_boot}, \mathit{safety})$.

After negotiating the car's technical features, the agents use the protocol to negotiate the price and maintenance terms (below we refer to these as "price terms"). Specifically, the potential customer (role A) requests price terms P from a car selling agent (role B) for the negotiated features T. Price terms are given as a tuple $P = (\mathit{buying}, \mathit{maint})$, where the two attributes refer to the cost of purchase and maintenance.

We define ten customer agents C_i, where $1 \leq i \leq 10$, with associated mental states, i.e. different preferences regarding T and P that determine what offers they will accept, where $C_i := MS_{i \bmod 5}$. Therefore, agents C_1 and C_6 have mental state MS_1, C_2 and C_7 have mental state MS_2, and so on.

For the purposes of this case study, we assume that a single seller (S) is analysing the system evolution from its local point of view, aiming to predict the different outcomes of its interactions based on perceived regularities regarding the observed behaviour of the customers[1].

5.2 Strategy to build the training data

In converting raw sequences of message exchanges to training data samples (see section 3.3), we make use of the simplest, most general data generation method that ProtocolMiner offers.

Firstly, we consider the agent $B = S$ (S is the seller agent name) who performs the analysis to obtain knowledge about the others agents' (opaque) mental states.

[1] Due to space limitations, this section does not detail the possible values of T, P, the mental states for the customers, and the decisions made by customers and sellers to follow the protocol described in figure 1. An extended version of this evaluation can be found at `http://ants.dif.um.es/staff/emilioserra/QA/EE.pdf`, and provides the necessary detail to reproduce our results.

Therefore, the learning input is restricted to $V_c(A)$ and $E_c(A)$, where c is the customer role in the protocol and A is any agent participating in this role.

As far as variables occurring in constraints are concerned, we uniformly record all attributes contained in "terms" descriptions T and P, including a "?" (unknown) value for those not mentioned in a given execution trace. Our strategy to deal with loops is to only record the last value of every variable occurring in multiple iterations. We introduce a variable $outcome \in \{S, F, N\}$ to denote Successful completion of the negotiation (path ended with m_{11}, see figure 1), Failure to unsuccessful (paths m_4 and m_6) and Neutral for a "neutral" outcome (the remaining paths).

Three open source implementations of data mining techniques are employed using their default parameters to illustrate the impact of using different algorithms in an exemplary way, and also to show that our method does not depend on the use of a specific learning algorithm. More specifically, we use the *J48* decision tree algorithm (an implementation of the C4.5 algorithm), the *NNge* classification rules algorithm (Nearest neighbor like algorithm) and the *BayesNet* technique (a Bayesian network classifier) [5].

5.3 Learning a context model for a constraint

In our first experiment, the seller tries to learn a context model for a single constraint, $c_4 = altAcceptable_A(T)$, and a single customer agent, C_1.

The output of the J48 algorithm after 1000 protocol executions is shown in figure 2. The time taken to build the model (on a 2.4 Ghz 4GB RAM machine) is 0.01 seconds and a tree with 15 nodes is obtained as the hypothesised mental state that corresponds to the logical formula

$$\begin{aligned} altAcceptable_{C_1}(T) \Leftrightarrow & (persons(T) = more \wedge lug\_boot(T) = small \wedge doors(T) = 5\text{-}more) \vee \\ & (persons(T) = more \wedge lug\_boot(T) = med \wedge doors(T) = 5\text{-}more) \vee \\ & (persons(T) = more \wedge lug\_boot(T) = big) \end{aligned}$$

This formula is logically equivalent to the mental state implemented by the customer, $MS_1(T)$, and, therefore, the constraint has been learned completely by the seller using context models even without having access to the customer's implementation.

5.4 Protocol outcome prediction by context models

To conduct a more exhaustive evaluation, a seller tries to learn a context model for the overall outcome of the protocol. Figure 5 shows the average model accuracy across 100 repeated experiments. The accuracy of the context models is evaluated using cross-validation across 100 experiments with 10000 negotiations each.

The experiments demonstrate that accurate context models can be built using contextual information extracted from concrete executions of a protocol to predict its final outcome. More specifically, after an average of 200 total negotiations (i.e. 20 per customer), the models classified at least 80% of all instances correctly. Therefore, as the following section shows, an agent can build a context model after only a few negotiations, consider a specific context as input for the model,

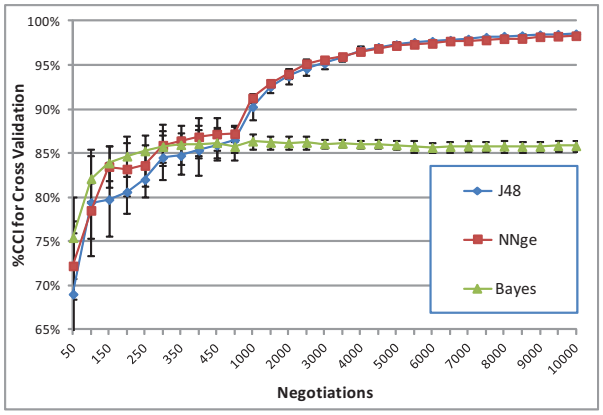


Figure 5: Average model accuracy (based on cross-validation) shown across total number of negotiation (100 experiments). Learning algorithms: J48, NNge and BayesNet. Error bars show standard deviation.

and predict the outcome of the protocol without actually interacting with another agent.

5.5 Agents predicting interactions by context models

In this section, we show how agents can use context models directly to improve their own performance in communicative exchanges with others. For this, we enable customers to build and use context models to choose a good seller for the concrete context (including the product) they are looking for. Assume that we have three sellers, S_1, S_2 and S_3, who are able to offer products which satisfy the different mental state models used by customers.

We compare the prediction accuracy of our system against two alternative analysis strategies: (1) *Random* and (2) *Quantitative*. For (1), the seller is chosen randomly – this provides a baseline for the minimum performance that could be achieved without any use of context models. An optimal strategy is not included, as 100% success constitutes the upper bound of what can be achieved in this scenario (we ensure that there are always sellers in the system who can provide the requested items). For (2), the seller is chosen using a distance function based on the number of past successes and failures with them in the customer's personal experience. The function used is $D(s, f) = 1-(1+\frac{s}{2f+1})^{-1}$, where s is the number of successes and f the number of failures with a particular seller, and the seller to interact with is chosen with a probability proportional to $D(s, f)$ [12].

As figure 6 shows, after 100 negotiations (10 negotiations per customer) the use of context models (independently of the data mining technique used) greatly outperforms the random and quantitative strategies. Besides, the use of decision trees converges faster to optimal performance than the other two learning techniques considered. In these experiments, using context models built with J48 or NNge, customers reach over 90% of successes after only 750 negotiations (75 negotiations

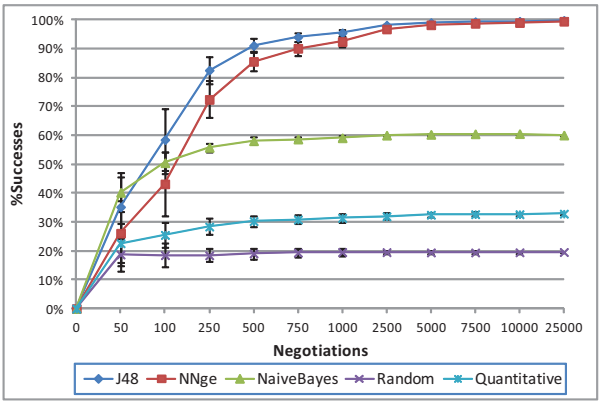


Figure 6: Average number of successful negotiations against number of total negotiations (100 experiments); error bars show standard deviation. Learning algorithms: J48, NNge and BayesNet.

per customer) and over 99% after 7500 conversations.

6 Conclusions and future work

In this paper, we have presented a novel mechanism to exploit *qualitative* information provided by high-level ACLs and interaction protocols. In these protocols, messages are associated with logical constraints, which can be used as "semantic" annotations of communication in a natural way. This work is motivated by shortcomings in existing multiagent systems analysis methods which mostly ignore this rich source of *contextual information* when analysing run-time multiagent interactions. The main advantage of using contextual information is that data mining methods can be used to infer qualitative information from observed message exchanges.

A formal approach has been detailed with the aim of making interaction data available for qualitative data mining. In this approach, information about the shared protocol models has been used as background knowledge. As part of the approach presented, this paper has discussed different alternatives for dealing with the specific nature of agent interaction protocols when converting interaction experiences to training data. This involves addressing issues such as the presence of multiple agents, variable-length execution paths, and loops that are commonly present in common multiagent interaction protocols. Subsequently, an implementation of our formal approach, ProtocolMiner, has been presented. Finally, a case study has been described (with an extended version available on-line) to hint at the potential of applying data mining in real-world multiagent systems.

In the future, we aim to apply our analysis methods to more real-world examples in order to extract guidelines for making appropriate choices when selecting training data extraction strategies and appropriate data mining algorithms. We would also like to explore the use of more advanced machine learning methods to learn logical

theories of, for example, the internal ontological conceptualisations agents use, and to rate their competence and trustworthiness based on the knowledge they appear to have based on their interaction behaviour. We believe these to be promising practical avenues for addressing one of the fundamental problems of open systems, which is to be able to derive knowledge of the internal workings of other agents without being able to observe their internal state.

References

[1] FIPA Contract Net Interaction Protocol Specification. SC00030, 2002. Foundation for Intelligent Physical Agents.

[2] FIPA Request Protocol Specification. SC00026, 2002. Foundation for Intelligent Physical Agents.

[3] M. Atencia and W. M. Schorlemmer. I-SSA: Interaction-Situated Semantic Alignment. In *OTM'08*, volume 5331 of *Lecture Notes in Computer Science*, pages 445–455. Springer, 2008.

[4] P. Besana and D. Robertson. Probabilistic Dialogue Models for Dynamic Ontology Mapping. In *URSW'08*, volume 5327 of *Lecture Notes in Computer Science*, pages 41–51. Springer, 2008.

[5] R. R. Bouckaert, E. Frank, M. Hall, R. Kirkby, P. Reutemann, A. Seewald, and D. Scuse. *Weka manual (3.7.1)*, June 2009. http://prdownloads.sourceforge.net/weka/WekaManual-3-7-1.pdf?download.

[6] R. W. Collier. Debugging Agents in Agent Factory. In *PROMAS'06*, pages 229–248, 2006.

[7] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[8] J. J. Gómez-Sanz, J. Botia, E. Serrano, and J. Pavón. Testing and Debugging of MAS Interactions with INGENIAS. In *AOSE'08*, pages 199–212, Berlin, Heidelberg, 2009. Springer-Verlag.

[9] D. N. Lam and K. S. Barber. Comprehending agent software. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *AAMAS*, pages 586–593. ACM, 2005.

[10] T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[11] D. Robertson. A lightweight coordination calculus for agent systems. In *DALT'04*, pages 183–197, 2004.

[12] E. Serrano, A. Quirin, J. A. Botía, and O. Cordón. Debugging complex software systems by means of pathfinder networks. *Inf. Sci.*, 180(5):561–583, 2010.

[13] R. Siebes, D. Dupplaw, S. Kotoulas, A. P. De Pinninck, F. Van Harmelen, and D. Robertson. The OpenKnowledge system: an interaction-centered approach to knowledge sharing. In *OTM'07*, pages 381–390, Berlin, Heidelberg, 2007. Springer-Verlag.

[14] J. Sudeikat, L. Braubach, A. Pokahr, W. Lamersdorf, and W. Renz. Validation of BDI Agents. In *PROMAS'06*, pages 185–200, 2006.

[15] M. Wooldridge, M. Fisher, M.-P. Huget, and S. Parsons. Model checking multi-agent systems with MABLE. In *AAMAS'02*, pages 952–959, New York, NY, USA, 2002. ACM.

An Assistance Infrastructure to Inform Agents for Decision Support in Open MAS*

Pablo Almajano[1], Maite Lopez-Sanchez[2], and Inmaculada Rodriguez[2]
[1] *Artificial Intelligence Research Institute (IIIA-CSIC),* palmajano@iiia.csic.es
[2] *WAI research group, University of Barcelona,* {maite,inma}@maia.ub.es

Abstract

Organisations are an effective mechanism to define the coordination model that structure agent interactions in Open MAS. Execution infrastructures mediate agents interactions while enforcing the rules imposed by the organisation. Although infrastructures usually provide open specifications to agents, understanding this specification and participating in the organisation could result a difficult task to agents, specially when the system is hybrid (i.e participants can be both human and software agents) and its specification becomes more and more complex. In this paper we further formalise a two layered *Assistance Infrastructure* in order to enable and evaluate different *Assistance Services* to agents in MAS. We also formalise the *Information Service* and evaluate it using the case study of a water market. Experiments results show that the information service increases agents satisfaction and helps the system meets its organisational goals. In addition, different information services may support individual agents in their decision processes when they follow alternative strategies.

Keywords: Decision Support, Assistance Infrastructures, Assistance Services

1 Introduction

Usually, multi-agent systems (MAS [15]) design and implementation involves the specification of a coordination model and the development of an infrastructure in charge of enacting it. In Open MAS, systems are populated by heterogeneous agents trying to achieve particular and/or collective goals. These agents are developed by third parties so that the number and the cognitive abilities of agents that may participate in an Open MAS is unknown at development time, and varies at runtime [16]. Organisation Centred MAS (OCMAS [12]) have proven to be an effective mechanism to define the coordination model that structures agent interactions in MAS, and infrastructures give support to their execution by imposing

*This work is partially funded by EVE (TIN2009-14702-C02-01 / TIN2009-14702-C02-02), AT (CONSOLIDER CSD2007-0022) and TIN2011-24220 Spanish research projects, EU-FEDER funds.

interaction rules between participants. Although OCMAS infrastructures usually provide open specifications to agents [11] [14], understanding these specifications and participating in the organisation could result a difficult task to agents, specially as its specification becomes more and more complex. If we take the humans in the loop and consider hybrid systems, where agents may be humans or software agents, the complexity increases and facilitating agent participation becomes a mandatory issue [3] [19].

This paper focuses on the challenge of improving agents' participation in the organisation by means of an *Assistance Infrastructure*. Certain data about the organisation and its environment require complex computational processes in order to be useful for agents. Therefore, agents would improve their participation in the organisation if the infrastructure could provide them with some assistance mechanisms that facilitate such processes. Our aim is to help agents in achieving their goals, and, when they are aligned with global goals, lead to a better system's global performance [18].

In this paper we further formalise an extension of the *Assistance Infrastructure* defined by Campos et al. [7] in order to enable and evaluate assistance in Open OCMAS systems. Our framework is composed by: i) the extension of the *Organisational Layer*, which regulates the participation of the agents in the system and manages the historical information about the organisation and its execution state; and the addition of an *Assistance Layer*, populated by *Personal Assistants* which provide general *Assistance Services* to agents in the organisation. In a previous work [1], we have preliminarily formalised four main categories of *Assistance Services*. Here, we further formalise the *Information Service*, provide a specific example, and evaluate it in a prototype based on *amWater* [1], which implements a water market in the agriculture domain.

In our experiments we first execute a *base-line* configuration without assistance. Then, data from this initial configuration are used in three alternative configurations that provide information assistance about *Runtime Properties*. An *Assistance Quality of Service* measure evaluates whether the assistance is an useful decision support tool for participant agents (i.e helps agents to satisfy their goals) and helps the systems to meet its goals (when they are aligned with participants' goals).

The paper is structured in the following parts. First, section 2 summarizes the related work. Second, section 3 provides definitions and notation in order to formalise both the proposed *Assistance Infrastructure* and the *Information Service*. Next, section 4 empirically evaluates information service. Finally section 5 gives some conclusions and future work.

2 Related work

There are two main lines of active research in assistance to MAS provided by *Software Agents*: organisational assistance services [8] [5] [6], and agent assistance services [10] [17] [9]. Regarding organisational assistance, Centeno et al. [8] defined an incentive mechanism (*Incentivators*) which induces individual participants to follow organisational goals by learning their preferences and doing modifications in the environment. On the other hand, Bou et al. [4] defined an Electronic Insti-

tution with autonomic capabilities that allows it to adapt its regulations to comply with institutional goals despite varying agent's behaviours (Autonomic Electronic Institutions). In a preceding work, they also applied Case-Based Reasoning (CBR) to reason about the process of adapting the norms of an Electronic Institution when certain system-wide measures differ from the expected ones [5]. Finally, the Two Level Assisted MAS Architecture (2-LAMA [6]) also provides organisational assistance services. It is composed by two levels. In the domain-level (DL) agents perform domain specific activities. On top of it, a distributed meta-level (ML) is in charge of providing assistance to the DL. This assistance is performed by changing the norms of the organisation and it is provided to groups of not overlapped and fixed clusters of agents. Our approach goes in the line of agent assistance service, so it differs from previous ones in organisational perspectives.

Other works focus on agent assistance services. Electric Elves [10] is a system that applies agent technology in service of the day-to-day activities of the Intelligent Systems Division of the University of Southern California Information Sciences Institute. Chalupsky et al. developed specific Software Personal Assistants (*SPA*) for project activities coordination and external meetings organisation. Since our proposal is general for MAS our *Assistance Layer* can include such kind of services.

These and other proposed *SPA*'s abilities were evaluated in a conceptual framework that simulated human behaviour in different MAS structures [18]. In this research, Okamoto et al. evaluated the impact that *SPA*s have on the individual performance and on the collective performance of the organisation as a whole. They built a computational model of human organisations and analysed two types of agent's organisational structures: hierarchical and horizontal. One *SPA* measured ability that is close to our proposal is the decision support (see section 3.2). They concluded that supporting decision tasks in human organisations increases the success rate (i. e., to meet the deadline with higher probability) and the speed performance average (i. e., to meet the deadline more rapidly), this is particularly the case in organisations with hierarchical structure.

A recent work presented a generic assistant agent framework in which various applications can be built [17]. As a proof of concept application, it implemented a coalition planning assistant agent in a peacekeeping problem domain. A more general framework for organising MAS [9] contains *Informative Organisational Mechanisms* and *Regulative Organisational Mechanisms*, a generalisation of the *Incentivators* [8]. As mentioned approaches, we also propose a general framework. Moreover, we propose to offer planning as an advice service in our infrastructure (see section 3.2) as in the former. The latter, *Informative Organisational Mechanisms*, is a generalisation of our *Agent's Assistance Services*.

Existing OCMAS infrastructures already offer some kind of information about the organisation to a participant. In the $\mathcal{S} - \mathcal{M}oise^+$ [14] middleware, the OrgManager provides useful information for the organisational reasoning of agents and its organisational coordination. In this model, an agent is allowed to know another agent information if their roles are linked by an acquaintance relation (defined in the social level). Moreover, the OrgManager also informs actors about the new permissions, obligations, and goals they can pursue when a new state is reached in the organisation. Our framework provides similar information services and elaborated ones, e.g., statistics on information that may be unknown for participants.

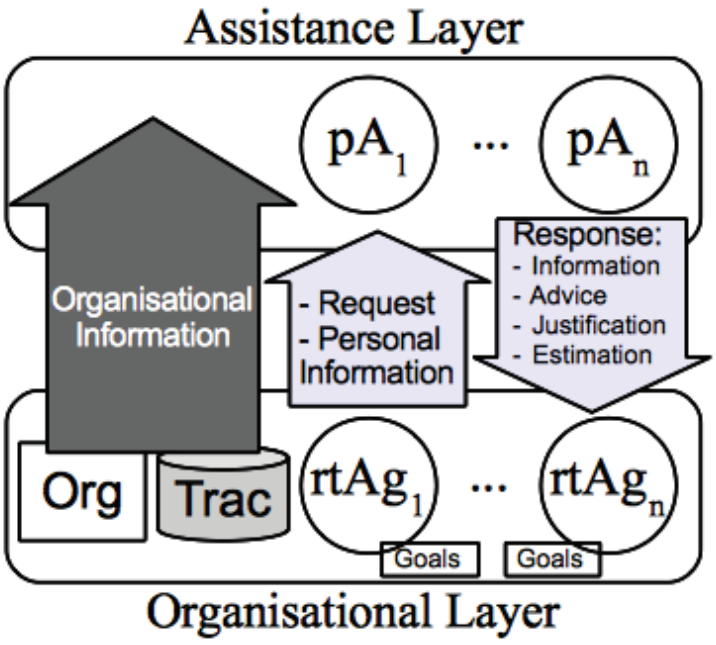


Figure 1: Assistance Infrastructure overview

3 Assistance Infrastructure formalisation

With the aim of assisting –further than enabling– agent coordination, we propose the *Assistance Infrastructure* depicted in Figure 1. It is composed by *Organisational Layer* (OL) and *Assistance Layer* (AL). The OL contains i) a set of running agents ($rtAg$), ii) the specification of the organisation (Org) and iii) historical information from previous system executions ($Trac$). The AL contains a set of personal assistants (pA) which provide support to $rtAg$. Three arrows show the private communication channel between both layers in Figure 1. First, pA use organizational information (i.e. Org and $Trac$ data) in order to provide assistance. Second, $rtAg$ may reveal personal information (e.g., their goals) and request help to pA in order to be adequately assisted. The last one is devoted for pA to provide the services (response) to $rtAg$. We illustrate the formalisation in *amWater* scenario, a water market where participants negotiate water rights. A *transaction* is the result of a negotiation where a seller settles with a buyer to reallocate (part of) the water from its water rights for a fixed period of time in exchange for a given amount of money[1].

3.1 Organisational Layer

We propose an *Organisational Layer* (OL) specification for Open MAS extended from the definition of Campos et al. [6]. Our main contributions in this layer are:

[1]We give a further explanation in section 4.1.

i) the extension of the *Organisation Specification* (Org) to include elements related to assistance and ii) the addition of the *Organisational Trace* ($Trac$).

3.1.1 Organisation Specification

Eq. 1 presents our proposed Organisation Specification (Org) (extensions appear in bold text).

$$Org = \langle \mathbf{O}, SocStr, SocConv, \mathbf{DomP}, Goals, \mathbf{AssQoS} \rangle \quad (1)$$

$$SocStr = \langle AgP, Rol, Rel \rangle \quad (2)$$

$$SocConv = \langle Prot, \mathbf{Activ}, \mathbf{ActivRel}, Norms \rangle \quad (3)$$

$$\mathbf{Goal} = \langle OrgGoal, \mathbf{AgSat} \rangle \quad (4)$$

–**Ontology. O** is a conceptualization of the domain (actions and possible domain concepts). For example, in amWater *StartRound* corresponds an action and *Transaction* (*trans*) is a concept with some properties (e.g., *trans.quantity*).

–**Social Structure.** $SocStr$ is a *social structure* (Eq. 2) consisting of: i) a set of agents properties (AgP), common for all participants; ii) a set of roles (Rol), where each role $\mathbf{rol} \in Rol$ contains the set of properties, **RolP**, characterizing it, $\mathbf{rol} = \langle \mathbf{RolP} \rangle$ and iii) Rel, the relationships among roles.

In general, we consider a property as a triple $\langle name, type, v \rangle$ with a $name$[2], a *type* and its *visibility* (v). *Visibility* implements the natural privacy policy of any organisation. A *private* property can only be accessed by its owner. An agent $rtAg$ perceives a property defined as *restricted* whenever it deploys role rol such as $restricted = \langle RoleV \rangle$ and $rol \in RoleV$. Finally, *public* denotes it is completely visible within the organisation.

In amWater, the different roles are irrigator – with buyer and seller subroles ($inheritance \in Rel$) –, market facilitator and basin authority. The property *trans.quantity* is public, agent owned water rights are private and auction's starting price is restricted to the market facilitator role.

–**Social Conventions.** $SocConv$ stands for the *social conventions* agents should conform to and expect others to conform to. It is defined in Equation 3 and is composed by a set of *Protocols* ($Prot$), a set of *Activities* (**Activ**), their relationships **ActivRel** and a set of Norms ($Norms$).

First, a *protocol* $\mathbf{prot} \in Prot$ is defined as $\mathbf{prot} = \langle \mathbf{ProtSpec}, \mathbf{ProtP} \rangle$, where **ProtSpec** specifies the interaction mechanism between agents and **ProtP** is the set of its properties. Second, an *activity* $\mathbf{activ} \in \mathbf{Activ}$ is defined as $\mathbf{activ} = \langle \mathbf{ActivP}, \mathbf{ActivProt} \rangle$, where **ActivP** is a set of properties and $\mathbf{ActivProt} \subseteq Prot$ is a set of protocols that can perform it. One *activity* can be instantiated multiple times and each instance can have one different protocol associated. Third, **ActivRel** defines the *activities relationships* as a work-flow specifying the possible transitions between different *activities*. Finally, $Norms$ are applied by the organisation with the aim of further regulating the *activities*. $Norms$ could be specific of one *activity* or apply to the overall system. Note that norms can be violated by agents whereas protocols' rules can not. In our example scenario, we have

[2]We use the notation $x.name$ to refer to property $name$ of component x (e.g., $buyer1.credit$ denotes the property $credit$ of agent $buyer1$).

one *Auction* activity, following a *multi-unit Japanese* auction protocol to negotiate transfers. As an example of a norm is *"buyers can acquire a maximum of 40% of the total quantity of water"*. A relationship between activities in our model is: after registering rights in a *Registration* activity, a seller may move to *Waiting And Information* activity.

–**Domain properties**, defined as $\mathbf{DomP} = \langle \mathbf{OrgP}, \mathbf{EnvP} \rangle$, are differentiated between: organisation internal properties (**OrgP**), such as the list of transactions (*Trans*); and environmental properties (**EnvP**), such as the water needs of a cultivation.

–**Goals** (**Goal**) are defined in Eq. 4 as a duple containing a set of *organisational goals* (*OrgGoal*) and a method to calculate *agent satisfaction* (**AgSat**). *OrgGoal* describe the organisation design purpose in terms of desired values for certain properties. **AgSat** is a method (may be asking the agents) to obtain the participants' degree of satisfaction at runtime. In *amWater*, one *organisational goal* is to minimise the auction time and the degree of *agent satisfaction* for a buyer can be related to the quantity of water bought.

–**Assistance Quality of Service** (**AssQoS**) mainly evaluates whether the assistance layer helps agents to satisfy their goals (*AgSat*) and, when they are aligned with organisational goals (*OrgGoal*) led to a better system performance. In section 4.3, we have tested *Information Services* defined in section 3.2.2 by comparing the values that *AgSat* and *OrgGoal* take in different configured executions.

3.1.2 Organisation Historical Information

At runtime, agents enter the organisation, interact trying to achieve their goals and finally exit. As the result of these (inter)actions, the state of the organisation changes. The Organisational Layer keeps the sequence of *Execution States* (S) and the agent *Runtime Actions* (RtA) within the *Organisational Trace* ($Trac$).

–**Execution States.** An state S contains current runtime (Rt) values of non-static organisational elements. Eq. 5-11 specify these elements based on Eq. 1-4. The applicable norms at runtime, $RtNorms$, specify (see Eq. 9): i) a set of active norms ($RtAct$); ii) the list of norm violations ($RtVio$); and iii) the list of consequences due to both norm applications or norm violations ($RtCon$).

$$S = \langle RtSocStr, RtSocConv, RtDomP, RtGoals, RtAssQoS \rangle \tag{5}$$

$$RtSocStr = \langle RtAg \rangle; rtAg = \langle RtAgP, RtRolP \rangle, rtAg \in RtAg \tag{6}$$

$$RtSocConv = \langle RtActiv, RtNorms \rangle \tag{7}$$

$$rtActiv = \langle RtActivP, RtProtP \rangle, rtActiv \in RtActiv \tag{8}$$

$$RtNorms = \langle RtAct, RtVio, RtCon \rangle \tag{9}$$

$$RtDomP = \langle RtOrgP, RtEnvP \rangle \tag{10}$$

$$RtGoal = \langle RtOrgGoal, RtAgSat \rangle \tag{11}$$

–**Runtime Actions**, $RtA = \{rtA_1, \ldots, rtA_n\}$, is the set of agents' actions performed at runtime, where n is the number of agents of the institution and rtA_i is the action performed by an agent i. We assign the *idle* action to rtA_i ($rtA_i = idle$) when agent i does not perform any action.

–**Organisational Trace**, $Trac$, contains the trace of the organisation (Eq. 12) specified by: i) a set of *time stamps*, T; ii) the sequence of *execution states* at each *time stamp*, S; and iii) a set of agent *actions performed* at each *state*, RtA. S_0 is the initial state, S_c is the current state and in general S_i is the organisation execution state at step i, RtA_i is the set of actions that take place at such step and T_i indicates the time at which S_i occurred. S only keeps information about changes in the organisation.

$$Trac = \{\langle T_0, S_0, RtA_0\rangle, \ldots, \langle T_c, S_c, RtA_c\rangle\} \tag{12}$$

Table 1 shows an example of $Trac$ in amWater. The property *"state"* of one auction activity at step occurred at time t ($rtActiv_i \in S_t.RtSocConc.RtActiv$) has the value *opened*, therefore the auction has not yet started. It just starts when agent $rtAg_j$ enacting market facilitator role ($RtRol_{rtAg_j} = MarketFacilitator$) performs the illocution *"StartRound"*. Then, the property *"state"* is updated and S changes at step *t+1*.

T	S	RtA
t	$rtActiv_i.state = opened$	$StartRound(rtActiv_i, rtAg_j)$
$t+1$	$rtActiv_i.state = running$	

Table 1: Example of amWater Organisational Trace at time t and *t+1*

3.2 Assistance Layer

The *Assistance Layer* depicted at the top in Figure 1 is in charge of providing decision support to agents in the *Organisational Layer*. It is populated by the so-called *Personal Assistants*, a set of organisational agents offering a set of *Assistance Services* to participants in the *Organisational Layer*.

We propose the Assistance Layer to provide the following services ($AsServ = \{Info, Adv, Just, Est\}$): i) refined *information* ($Info$), ii) *advice* (Adv), iii) *justification* ($Just$) of either action consequences or applied constraints when performing an action and iv) *estimation* (Est) of action consequences.

3.2.1 Personal Assistant Agents

We define PA as the set of *Personal Assistant Agents* which belong to the organisation. One personal assistant $pA_i \in PA$ provides direct and sole support to one agent[3] $rtAg_i \in RtAg$. We have formalised two assistance communication processes[4]:

$$AsServ : Org \times Trac \rightarrow Res \tag{13}$$

$$AsServReq : Req \times Org \times Trac \rightarrow Res \tag{14}$$

Since *Personal Assistants* are organisational agents, they are allowed to use organisational information[5] in order to provide assistance. However, they only provide

[3]An agent, as an autonomous entity, can freely use the given support in its decision process.

[4]The private communication channel is depicted as arrows in Fig. 1.

[5]We assume in this definition that the infrastructure sends information about the organisation specification (Org) and its execution state ($Trac$).

responses (Res) concerning properties that respect *visibility*[6] constraints. First, Eq. 13 defines the process by subscription ($AsServ$). It can be provided at different frequencies of subscriptions: each time the information changes, e.g. when a new water right has been registered to transfer; at concrete moments, e.g. the first time entering the organisation the agent receives a "welcome pack"; and never, e.g. subscription disabled. Second, Eq. 14 defines the process under request ($AsServReq$). The request, Req, can include personal information of the agent (e.g., its goals) which can decide whether to reveal it or not[7]. The specification of both Req and Res illocutions contains first the sender agent, second the receiver agent and a set of parameters in the request, $Req = \langle rtAg_i, pA_i, Par \rangle$, or a set of values in the response, $Res = \langle pA_i, rtAg_i, Val \rangle$.

As previously introduced, we propose to establish a private communication channel between a personal assistant pA and its assisted agent $rtAg$ in order to preserve the privacy of the information in the communications. To ensure the use of private information in the defined terms and conditions, a service contract may be signed between pA and $rtAg$. On one hand, this contract commit pA to keep personal information private, to not exploit it, and to offer services pursuing $rtAg$'s private information following social conventions. On the other hand, $rtAg$ is responsible for the use it makes of the services rendered based on the personal information revealed by it and can decide when the personal information should be erased by the assistants. As a pA is designed to use organisational trace and personal information in a private way, we consider that agents should have confidence using them.

3.2.2 Information Services Specification

We consider 3 main reasons for agent needing of information. First, the specification of the organisation could be difficult to understand for some participants. Second, runtime data could not be perceived by participants, because, even though it is visible to them, they were not notified about it. Finally, data from previous executions may require to be processed in order to be useful for agents' decisions, e.g., the weighted average value of transactions' prices in *amWater* along a previous market execution. Therefore, *Information Service* can be divided into three subcategories of services: i) *Organisation Specification*, ii) *Runtime Organisation Specification* and iii) *Runtime Properties*. Next we define the different parameters of the request (Par) and the values provided in the response (Val) for each subcategory.

Organisation Specification (Os) is information about the organisation as defined at design time. The only parameter of the request is the name of a component[8] in the organisation specification (Org). The response contents the value of such a component at design time. For example, let imagine that in amWater an agent Pablo ($rtAg_{Pablo}, rtRol_{Pablo} = seller$) wants to participate in a registration activity following protocol $prot_j$ and require the roles allowed to participate in such a protocol to his personal assistant: $Req_{Os}(rtAg_{Pablo}, pA_{Pablo}, prot_j.Rol)$,

[6] *Properties visibility* is further explained in section 3.1.1

[7] We stress that the more relevant information revealed, the better services will be provided.

[8] It should be specified by using the access variable values operators mentioned in section 3.1.1

obtaining as response $Res_{Os}(pA_{Pablo}, rtAg_{Pablo}, \{seller, marketFacilitator\})$.

Runtime Organisation Specification ($RtOs$) is information of the organisation specification based on the current situation of the agent within the organisation at runtime. In this case, one parameter $par \in \{actions, location, destination\}$ indicates the type of specification requested: i) the allowed *actions*, ii) the current *location* or iii) the possible *destinations* in the organisation. The values in the response can be i) a set of permitted *actions*; ii) a *location loc* expressed as an activity identifier or an activity's relationship, $loc \in \{Activ, ActivRel\}$; or iii) a set of *destinations*, where a destination *des* is expressed in the same way as a location ($des \in \{Activ, ActivRel\}$). Following with the example in amWater, once Pablo has joined the registration activity $rtActiv_j$, he could ask for the actions he is allowed to perform at current state. The request will be $Req_{RtOs}(rtAg_{Pablo}, pA_{Pablo}, actions)$ and one response could be $Res_{RtOs}(pA_{Pablo}, rtAg_{Pablo}, \{register, exit\})$.

Runtime Properties (Rt) is (processed) information about historical values of the observable properties. The parameters of the request are the name of a property in S, and two timestamps defining a period: t_{ini}, indicating the beginning, and t_{fin}, indicating the end. The response will content the corresponding values of such a property at runtime between the specified timestamps. Continuing with the previous example, Pablo can request for a low transactions price in order to decide the starting price of his water rights in the negotiation. One possible request could be about transactions of a previous auction activity k between rounds with time stamp t_1 and t_{80}: $Req_{Rt}(rtAg_{Pablo}, pA_{Pablo}, \langle rtActiv_k.Trans.lowPrice, t_1, t_{80}\rangle)$. One possible response could be $Res_{Rt}(pA_{Pablo}, rtAg_{Pablo}, 9.00)$.

4 Assistance Evaluation

This section is devoted to evaluate our proposed *Assistance Infrastructure* by testing different information service configurations in a water market case study.

4.1 amWater: a Demonstrator of Assistance Scenario

As a case study, we have enacted an *a*ssisted electronic *m*arket of *Water* rights (*amWater*[9]) based on *mWater* [13]. The *Organisational Layer* corresponds to an *Electronic Institution* (EI [2]) and the *Assistance Layer* improves the market by providing *Runtime Properties Information Service*. Considering an agricultural context, *amWater* is associated to an irrigation basin which is divided in geographical areas of interconnected lands (and their associated water rights). Water transfers between lands are possible by using available basin's infrastructures.

External agents join this water market enacting either *buyer* or *seller* subroles while *market facilitator* and *basin authority* are designated for staff agents (*Rol* and *Rel* in Eq. 2). Specifically, agents can join three activities (*Activ* in Eq. 3): *Registration*, where the market facilitator is in charge of registering sellers' rights; *Waiting and Information*, where irrigators can ask for information about auctions to the market facilitator; and *Auction*, where the negotiation of water rights takes place. We have selected a multi-unit Japanese Auction protocol (*Prot* in Eq. 3)

[9] Notice that this domain is naturally structured and requires organisation.

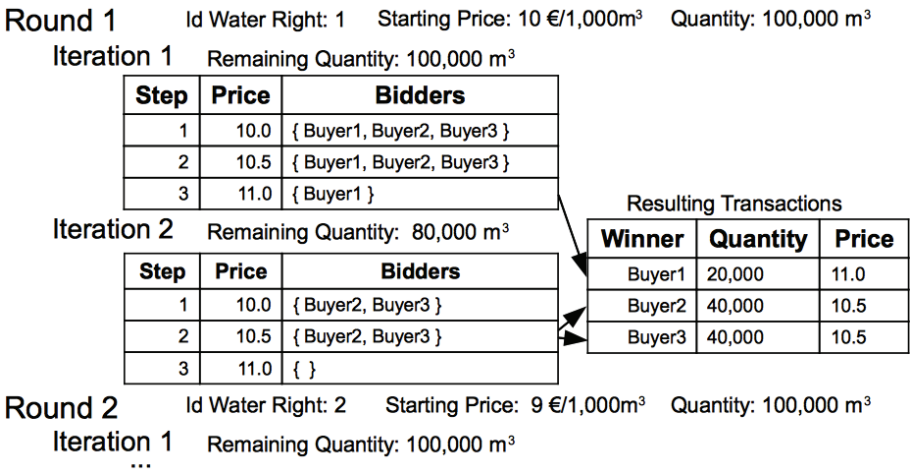


Figure 2: An example execution of a *Multi-unit Japanese* protocol

because it is suitable for divisible and perishable goods, in our case, water. In this protocol, the *market facilitator* conducts the auction of registered water rights –composed by several m^3 of water– in a *round-iteration-step* schema. Figure 2 shows an example execution of such a protocol. *Rounds* are divided in several *iterations* which are further divided in several *steps*. The *market facilitator* starts a new *round* for each registered water right at the starting price established by the *seller*. Subsequent *iterations* follow these four rules. First, one *iteration* is composed by several *steps* where the price increases in regular increments (0.5€ in Fig.2). Second, *buyers* are allowed to place bids for consecutive *steps*. Third, the *iteration* ends when i) just one buyer bids at current *step* and becomes the winner or ii) no bids are performed, so the winners are determined to be the *buyers* that bid at previous *step*. Last, winners request the amount of water desired (e.g. *Buyer1* buys 20,000m^3). If there is more than one winner (e.g. *Buyer2* and *Buyer3* in second iteration), then the water is assigned by following a proportional allocation algorithm (i.e. 40,000m^3 for each buyer). Once an *iteration* is finished, the *basin authority* validates the result(s) and announces the resulting transaction(s). Afterwards, if it remains water in the right, then a new *iteration* starts. No bids then would imply the ending of both the *iteration* and the *round*. The negotiation is over when all rights have been traded.

4.2 Experiment configuration

In order to asses the benefits (in terms of system performance and quality of service) of our information service, we have configured four alternative experiments. The first one does not use it and, thus, acts as a base-line for comparison purposes. The other configurations use the service (which gathers trace information from the base-line) by issuing different information requests.

Agents in the *base-line* configuration include staff agents, which follow a predefined and fixed behaviour, and an heterogeneous population of 100 buyers and 100

sellers. All buyers and sellers ($rtAg_i, rtRol_i = buyer|seller$) aim at buying/selling the same fixed water quantity ($rtAg_i.quantity = 100{,}000$). The variation in their behaviour is modelled in terms of the purchasing/sale price of water rights. Specifically, we assume buyers have an inner maximum purchasing price whose value is normally distributed, $\mathcal{N}(\mu, \sigma^2)$, with $\mu = 12$ (i.e. 12 €/*1,000m*^{3}) and $\sigma^2 = 2$. As for sellers, their starting price is low enough to ensure sales and follows a normal distribution $\mathcal{N}(4, 1)$. In order to preserve similar market conditions, just sellers'

Configuration	**Parameters of request** (Par)	**Value of response** (Val)
base-line	no information	–
low	$\langle Trans.lowPrice, t_1, t_{80}\rangle$	$Trans.minPrice - k$
medium	$\langle Trans.medPrice, t_1, t_{80}\rangle$	$Trans.wAvePrice - k$
high	$\langle Trans.highPrice, t_1, t_{80}\rangle$	$Trans.maxPrice - k$

Table 2: Request performed in each configuration and the subsequent response

price strategies are changed among the other test configurations. As Table 2 shows, sellers request for a different information service in different configurations. These services represent a decision support tool for setting seller's starting price. Thus, if the seller strategy is to set low starting prices, the corresponding information service will provide the minimum transaction price in the trace with a k decrement. Similarly, the seller can request a medium or a high starting price (Par), and the values of the responses (Val) will be a value k below the weighted average ($wAvePrice$) or the maximum historical price respectively. Equation 15 details the $wAvePrice$ computation over a set of transactions ($Trans$), where $trans_i.quantity$ and $trans_i.price$ denote the water quantity and price of transaction i.

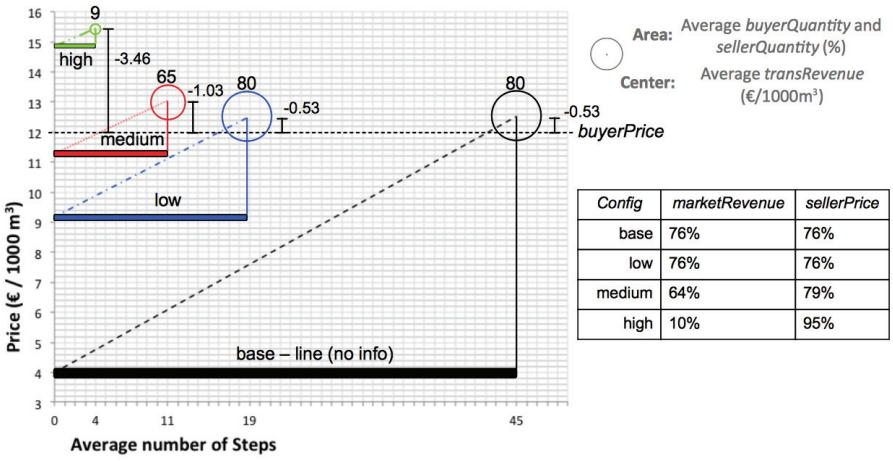
$$wAvePrice(Trans) = \frac{\sum_{i=1}^{|Trans|} (trans_i.quantity \times trans_i.price)}{\sum_{i=1}^{|Trans|} trans_i.quantity} \tag{15}$$

4.2.1 Goals

We have defined three *Organisational Goals*, and four *Agent Satisfaction* parameters ($OrgGoal$ and $AgSat$ in Eq. 4), two for buyers and two for sellers:

$$OrgGoal = \{transPerform, transRevenue, marketRevenue\}$$
$$AgSat = \{buyerQuantity, buyerPrice, sellerQuantity, sellerPrice\}$$

On one hand, $OrgGoal$ contains the following components: i) *Transaction Performance* ($transPerform$), inverse to the average number of steps to complete a transaction (the more steps, the worse performance); ii) *Transaction Revenue* ($transRevenue$), transactions' average price (€) per water unit (*1,000 m*^{3}) (computed as the $wAvePrice$ in Equation 15); and iii) *Market Revenue* ($marketRevenue$), percentage of the actual revenue (i.e. total transacted water quantity times weighted average price) over the maximum possible revenue (i.e. total auctioned quantity times maximum transactions' price).



Config	marketRevenue	sellerPrice
base	76%	76%
low	76%	76%
medium	64%	79%
high	10%	95%

Figure 3: Average *OrgGoal* and *AgSat* values of ten executions

On the other hand, *AgSat* is composed by: i) *buyerQuantity*, percentage of transacted water quantity over the total quantity that buyers aim to acquire (i.e. buyer satisfaction increases with water acquisitions); ii) *buyerPrice*, a price value which indicates the difference of the average of buyers' maximum purchasing price ($rtAg.maxPrice$) with respect to the actual average price ($wAvePrice$) (i.e. buyer satisfaction decreases when the price on the market exceed its inner maximum purchasing price); iii) *sellerQuantity*, the percentage of water quantity actually transacted over the total quantity registered by sellers (i.e. seller satisfaction increases with water sales); iv) *sellerPrice*, the percentage of $wAvePrice$ with respect to the maximum transaction price (i.e. sellers' satisfaction increases when the price on the market gets closer to the maximum historical price).

4.3 Assistance Quality of Service

In order to evaluate our information assistance infrastructure ($AssQoS$ in Eq. 1), we have conducted the experiments defined in section 4.2. Figure 3 shows the averaged results of executing ten times each of the four configurations[10]. The graphic on the left contains thick horizontal bars which correspond to the number of steps required to close a transaction (the inverse of $transPerform$). Thin vertical bars go from the starting prices (responses of each information request) to the weighted average transaction prices ($transRevenue$). The latter becomes the centre of a circle whose area represents both, buyers' and sellers' quantity satisfaction (*buyerQuantity* and *sellerQuantity*)[11]. The area value labels the circle. *buyerPrice* value is represented by a vertical bar just on the right of each circle. It starts in $transRevenue$ and ends in the average buyers' maximum purchasing price, in our case 12. Additionally, the table on the right completes the results by providing the *marketRevenue* and

[10]For these experiments, we have fixed k to the 10% of the respective $Trac$ value

[11]Note that we have $buyerQuantity = sellerQuantity$ because $rtAg_i.quantity = 100{,}000$ for all buyer and seller agents.

sellerPrice. Since the information services under evaluation target seller agents, we will focus on their satisfaction and the overall system performance.

As we can observe, the values obtained for *low* configuration get a *sellerPrice*, *sellerQuantity*, *marketRevenue* and *transRevenue* that are as high as the *baseline* configuration, with the advantage that they are reached in far less time (see the average number of steps). Thus, we can argue that the *low* information service drastically improves performance ($transPerform$). Accordingly, if we assume that system and agent goals are aligned, a seller agent can use this service to reduce the time of its sales without affecting any of its other satisfaction attributes.

On the other hand, seller pricing strategies may be more aggressive and pursue a higher *sellerPrice*. The results of the *medium* and *high* configurations show us that if the seller agent follows the corresponding services' advice when setting the starting price, then the *sellerPrice* can be increased close (95%) to the maximum sold price. Nevertheless it takes the risk of not being the one who is actually selling at the desired higher price (since the proportion of transactions, *sellerQuantity*, decreases down to 9 and the *marketRevenue* moves down to 10%). Moreover, taking higher risks has the additional positive effects of further improving $transPerform$ and *transRevenue* (i.e. transactions are performed faster and at higher prices).

Overall, we can conclude that these experiments show that system performance and agent satisfaction (and thus, the quality of service) increase with the addition of information services. Furthermore, different services can be useful for decision support processes being carried out by following alternative strategies.

5 Conclusions and future work

In this paper we have formalised both an *Assistance Infrastructure* and an *Information Service* to support agents participation in Open MAS. The *Assistance Infrastructure* is composed by two layers. First, the *Organisational Layer* is composed by a set of *Running Agents*, an extension of an *Organisation Specification* and the addition of the *Organisational Trace*, which keeps historical information of previous organisation executions. Second, the *Assistance Layer* has been populated with one *Personal Assistant* per each *Running Agent*. *Information Services* offers, in addition to basic organisational information, more elaborated information, e. g. specific statistics on data that might be unknown to participants.

In order to illustrate and evaluate our approach we use an OCMAS example scenario that implements an electronic market of water rights. In the tests performed, we have compared the values that different agent satisfaction parameters and system goals take when agents request for different information services, using as a base-line a configuration without enabling assistance services. The experiments show that system performance and agent satisfaction (and thus, the quality of assistance service) increase with the addition of information services. Furthermore, individual agents following alternative strategies can use different services as an useful decision support tool. As future work, we plan to enable and evaluate both the *Os* and *RtOs* information services, the rest of services (advice, justification and estimation) as well as include and evaluate assistance for human participants by means of a 3D Virtual World interface.

References

[1] P. Almajano, M. Lopez-Sanchez, M. Esteva, and I. Rodriguez. An assistance infrastructure for open mas. In *CCIA '11*, volume 232, pages 1–10. IOS Press, 2011.

[2] J. Arcos, M. Esteva, P. Noriega, J. Rodríguez-Aguilar, and C. Sierra. An integrated development environment for electronic institutions. In *Software Agent-Based Applications, Platforms and Development Kits*, pages 121–142. 2005.

[3] A. Bogdanovych, M. Esteva, S. Simoff, C. Sierra, and H. Berger. A methodology for developing multiagent systems as 3d electronic institutions. volume 4951 of *LNCS*, pages 103–117. Springer Berlin / Heidelberg, 2008.

[4] E. Bou, M. López-Sánchez, and J. Rodríguez-Aguilar. Towards self-configuration in autonomic electronic institutions. volume 4386 of *LNCS*, pages 229–244, 2007.

[5] E. Bou, M. López-Sánchez, and J. A. Rodríguez-Aguilar. Autonomic electronic institutions' self-adaptation in heterogeneous agent societies. In *Organized Adaption in Multi-Agent Systems*, volume 5368, pages 18–35. Springer, 2009.

[6] J. Campos, M. Esteva, M. López-Sánchez, J. Morales, and M. Salamó. Organisational adaptation of multi-agent systems in a peer-to-peer scenario. *Computing*, 91:169–215, 2011.

[7] J. Campos, M. López-Sánchez, and M. Esteva. Coordination support in multi-agent systems. In *AAMAS'09*, pages 1301–1302, 2009.

[8] R. Centeno and H. Billhardt. Adaptive regulation of open mas: an incentive mechanism based on online modifications of the environment (extended abstract). In *AAMAS'11*, pages 1243–1244, 2011.

[9] R. Centeno, H. Billhardt, R. Hermoso, and S. Ossowski. Organising mas: a formal model based on organisational mechanisms. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 740–746, 2009.

[10] H. Chalupsky, Y. Gil, C. A. Knoblock, K. Lerman, J. Oh, D. V. Pynadath, T. A. Russ, and M. Tambe. Electric elves: Applying agent technology to support human organizations. In *IAAI'01*, pages 51–58. AAAI Press, 2001.

[11] M. Esteva. *Electronic institutions. from specification to development*. PhD thesis, Universitat Politecnica de Catalunya, 2003.

[12] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: An organizational view of multi-agent systems. In *Agent-Oriented Software Engineering IV*, volume 2935, pages 443–459. 2004.

[13] A. Garrido, A. Giret, and P. Noriega. mwater: a sandbox for agreement technologies. In *CCIA'09*, pages 252–261, 2009.

[14] J. Hübner, J. Sichman, and O. Boissier. $\mathcal{S} - \mathcal{M}oise^{+}$: A middleware for developing organised multi-agent systems. In *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913, pages 64–77. 2006.

[15] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38, 1998.

[16] I. J. Jureta, M. J. Kollingbaum, S. Faulkner, J. Mylopoulos, and K. Sycara. Requirements-driven contracting for open and norm-regulated multi-agent systems. Technical report, 2007.

[17] J. Oh, F. Meneguzzi, K. Sycara, and T. J. Norman. Prognostic agent assistance for norm-compliant coalition planning. In *ITMAS'11*, pages 126–140, 2011.

[18] S. Okamoto, K. Sycara, and P. Scerri. Personal assistants for human organizations. In V. Dignum, editor, *Organizations in Multi-Agent Systems*. 2009.

[19] T. Trescak, M. Esteva, and I. Rodriguez. Vixee an innovative communication infrastructure for virtual institutions. In *AAMAS'11*, pages 1131–1132, 2011.

Behaviour Driven Development for Multi-Agent Systems

Álvaro Carrera, Jorge J. Solitario, and Carlos A. Iglesias
Dpto. de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid, UPM
Madrid, Spain
{a.carrera,jjsolitario,cif}@dit.upm.es

Abstract

This paper presents a testing methodology to apply Behaviour Driven Development (BDD) techniques while developing Multi-Agent Systems (MAS), so called BEhavioural Agent Simple Testing (BEAST) methodology. It is supported by the developed open source framework (BEAST Tool) which automatically generates test cases skeletons from BDD scenarios specifications. The developed framework allows testing MASs based on JADE or JADEX platforms and offers a set of configurable Mock Agents which allow the execution of tests while the system is under development. BEAST tool has been validated in the development of a MAS for fault diagnosis in FTTH (Fiber To The Home) networks.

Keywords: agent, test, simple, management, development, BDD, multi-agent systems, MAS, mock-agents, JADE, JADEX.

1 Introduction

Understanding stakeholders requirements and fulfilling their desired functionality is considered as the most important aspect for a software project to be considered successful [4]. Thus, requirements engineering plays a key role in the development process. The main challenges of requirements engineering are [17]: (i) improving the communication between the stakeholders and the implementation team and (ii) understanding the problem.

Nevertheless, the process of eliciting requirements and communicating them is still an issue and some authors consider it the *next bottleneck to be removed from the software development process* [2]. The main reasons for this communication gap between stakeholders and the development team are that [2] (i) imperative requirements are very easy to misunderstand; (ii) even the obvious aspects are not so obvious and can be misinterpreted and (iii) requirements are overspecified, since they are expressed as a solution, and focus on what to do and not why, not allowing the development team whether discuss if those requirements are the best way to achieve stakeholders' expectations.

In order to bridge this communication gap, the agile movement has proposed to shift the focus of requirements gathering. Instead of following a contractual approach where the requirements documents is the most important goal, they put emphasis on improving the communication among all the stakeholders and developers to have a common understanding of these requirements. Moreover, given that requirements will have inconsistencies and gaps [3], it has been proposed to anticipate the detection of these problems by checking the requirements as soon as possible, even before the system is developed. In this line, Martin and Melnik formulated the equivalence hypothesis: "As formality increases, tests and requirements become indistinguishable. At the limit, tests and requirements are equivalent" [18].

As a result, they have proposed a practice so called *agile acceptance testing*, whose purpose is improving communication by using real-world examples for discussion and specifications of the expected behaviour at the same time, which is called Specification by Example (SBE). Different authors have proposed to express the examples in a tabular form (Acceptance Test Driven Development (ATDD)[1] with Fit test framework [19]) or as scenarios (BDD [25] with tools such as JBehave [24] or Cucumber [30]). In this way, requirements are expressed as acceptance tests, and these tests are automated. When an agile methodology is followed, acceptance tests can be checked in an automated way during each iteration, and thus, requirements can be progressively improved. Most frameworks provide a straight forward transition from acceptance tests to functional tests based on tools such as the xUnit family [14]. Agile acceptance testing complements Test Driven Develpoment (TDD) practices, and it can be seen as a natural extension of TDD practices, which have become mainstream in among software developers. In this way, software project management can be based not only on estimations but on the results of acceptance and functional tests. In addition, these practices facilitate to maintain requirements (i.e. acceptance tests) updated along the project lifespan.

In the multi-agent field, there have been several efforts in the testing of multi-agent systems. Multi-agent systems testing present several challenges [20], given that agents are distributed, autonomous and it is interesting not its individual behaviour but the emergent behaviour of the multi-agent system that arises from the interaction among the individual behaviours. A good literature review of MAS testing can be found in [20, 21, 16]. To the best of our knowledge, there is no previous work dealing explicitly with acceptance testing in Agent Oriented Software Engineering (AOSE). Thangarajah et al [28] propose to extend the scenarios of the Prometheus methodology in order to be able to do testing of scenarios as part of requirements or acceptance testing. The work describes also a novel technique for integrating agent simulation in the testing process. Nevertheless, their proposal of acceptance tests seems targeted at technical users, given than the scenarios are described for based on percepts, goals and actions. Nguyen et al. [22] propose an extension of the Tropos methodology by defining a testing framework that takes into account the strong link between requirements and test cases. They distinguish external and internal testing. External testing produces acceptance tests for being validated by project stakeholders, while internal testing produces system and agent tests for being verified by developers. They focused on internal testing.

[1] A literate review of ATDD can be found in [15].

The research context of this article was a research project contracted by the company Telefonica. They requested us to develop a multi-agent system for fault diagnosing in their network. From a software engineering point of view, the main challenges were: (i) they required managing the project using the agile SCRUM methodology [26], (ii) the project involved integration with a wide range of external systems and the emulation faulty behaviour of network transmission and (iii) the development team was composed of students with different timetables, so they were not working together most of the time. After the first release, the main problems we encountered were communication problems between the development team and the customer (expert network engineers), communication problems within the development team, where agents were being developed in parallel, and lack of automation in the unit testing process, which involved to test physical connections with a manual and very time consuming process.

After analysing several AOSE proposals based on agile principles [8, 12], we have not found any proposal which covers *acceptance tests* and provides a good starting point for its application in an agile context. Thus, this research aims at bridging the gap between acceptance testing and AOSE. The key motivation of this paper is to explore to what extent acceptance testing can benefit MAS development, in order to provide support in the development of MAS in agile environments. This brought us to identify the need for an agile testing methodology for MAS.

The rest of the article is organised as follows. First, we propose a testing methodology for MAS based on BDD techniques in Sect. 2 which is supported by an open source tool. Sect. 3 presents a worked example of the application of the methodology and the tool. Then, Sect. 4 presents an evaluation of the benefits of the proposed approach. Finally, Sect. 5 presents some concluding remarks.

2 BEAST Methodology

In order to cover the problems identified in Sect. 1, we should identify which requirements should have the testing methodology. First, our primary concern is that it should help in improving the communication between the stakeholders and the development team. In addition, it should help to improve the communication between the development team. Another requirement comes from the overall methodology: it should be compatible and suitable for its application in combination with agile techniques. Finally, it should not be tied to a specific MAS tool, and it should be feasible to integrate with other MAS environments with low effort.

The BEAST methodology is intended to be used in agile environments, with special focus on providing traceability of stakeholder requirements. With this end, requirements are automated as *acceptance tests*, which are linked with MAS testing. The main benefit of this approach is that it improves the understanding of the real advance of the project from the stakeholders perspective, and, moreover, it provides a good basis for reviewing the objectives of each iteration (so-called *sprints* in SCRUM terminology). As a result, requirements negotiation and specification can be done in an iterative way, and can be adapted to the improved understanding of the desired system by both stakeholders and development team.

BEAST consists of four phases (Fig. 1): *System Behaviour Specification*, *MAS*

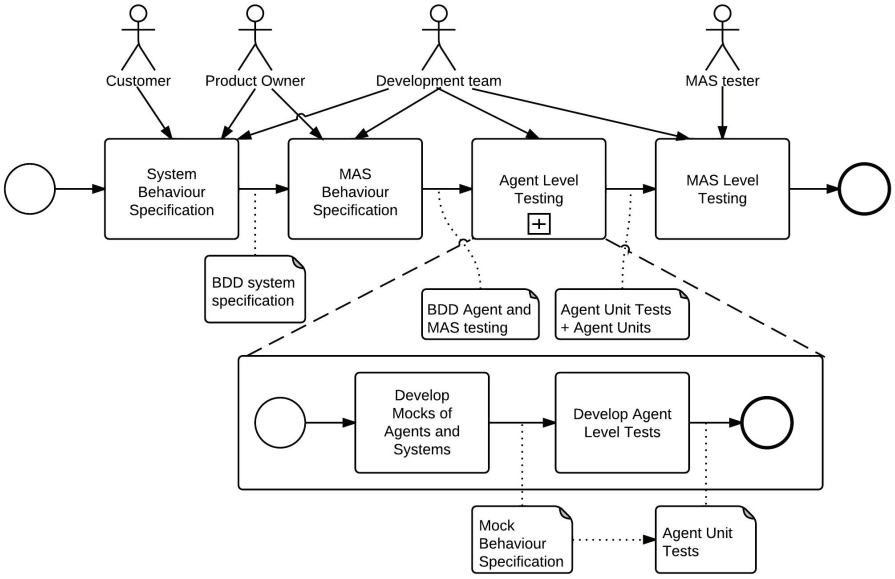


Figure 1: Beast Testing Methodology

Behaviour Specification, *Agent level testing* and *MAS level testing*, which are applied in each agile iteration.

2.1 System Behaviour Specification

The *System Behaviour Specification* phase aims at providing a communication bridge between the project stakeholders and the development team during requirements gathering. This phase follows the BDD technique [25]. System behaviours are derived from the business outcomes the system intends to produce. These business outcomes should be prioritized by the stakeholders. Then, business outcomes are drilled down to feature sets. Feature sets decompose a business outcome into a set of abstract features, which show what should be done to achieve a business outcome. These feature sets are the result of discussions between stakeholders and developers. Features are described using user stories. Then, user stories are described in scenarios for each particular instantiation of a user story. These scenarios are the basis of acceptance tests. Instead of using plain natural language, BDD proposes the usage of textual templates (Tables 2 and 3). These templates should be instantiated by the pertinent concepts. These concepts are part of the *ubiquitous language* [11] which establishes the common terminology used by stakeholders and developers. Thus, these terms will be used in the implementation, helping in reducing the gap between technical and business terminology.

```
[Story title] - description
As a [Role]
I want a [Feature]
So that [Benefit]
```

Figure 2: User Story template

```
Scenario [Scenario name]
Given [Context]
And [Some more contexts] ...
When [Event]
Then [Outcome]
And [Some more outcomes] ...
```

Figure 3: Scenario template [25]

2.2 MAS Behaviour Specification

This phase has the goal of architecturing the multi-agent system. Based on the features identified in the previous phase, the new features are realised with the MAS system. This can involve adding, removing or modifying behaviours developed in the previous iterations. In order to maintain traceability and improve communication within the development team, we have found useful to use the same approach than in the previous phase for specifying the MAS behaviour. Thus, *business benefits* are described by *features* which are assigned to *agent roles*. As previously, *features* can be obtained in different contexts which are described as scenarios, which can involve one or more agent roles in the case of *cooperative scenarios*. In the case of *emergent features* coming from emergent behaviour, they will be only verified when the full system has been developed. This kind of emergent behaviour will be specified at MAS level in the agent stories, instead of for a particular agent role.

This phase could be skipped and system behaviours could be directly translated into agent unit tests (Sect. 2.3). In fact, our first version of the framework did not include this step. Nevertheless, we have found it very useful in order to make explicit how stakeholders specifications are translated into MAS requirements, and provide better insight for developers about them.

2.3 Agent level testing

Based on the previous phase, agents are designed and developed. For this purpose, any of the available AOSE methodologies can be used for modelling and implementing agents. Since we are focused on testing aspects, this phase has two main steps (Fig. 1): (i) developing mocks of the external systems an agent interacts with and (ii) developing the unit tests of every agent.

The first step (Sect. 2.3.1) requires to simulate the intended behaviour of the external systems according to the scenarios described in the previous phase. In our methodology, an external system includes other agents different from the one we are developing.

The second step (Sect. 2.3.2) provides mapping rules from the specifications developed in the previous phases to executable code.

2.3.1 Mock development

There have been several research works developing the concept of using mock testing for agent unit testing. Coelho et al. [9] proposed a framework for unit testing of MAS based on the use of mock agents on top of the multiagent framework JADE [5]. They proposed to develop one mock agent per interacting agent role. Mock agents were programmed using script-based plans which collect the messages that should be interchanged in the testing scenarios. Tiryaki et al. [29] proposed the framework *SUnit* on top of the multiagent framework *Seagent* [10]. They extended *JUnit* testing in order to cope with agent plan structures testing. Zhang [31] generated automatically mock agents from design diagrams developed within the Prometheus methodology.

We propose to use a mock testing technique for simulating external systems, being agents or any other system. In addition, we need that the framework is valid for different MAS platforms, such as JADEX [7] and JADE [5]. In addition, the mocking framework should allow an easy configuration of the mock objects (or agents), with patterns such as *when(< some input >).thenReturn(< some answer >)*. After analysing available mocking frameworks, we have selected *Mockito* [1] framework, because of its easiness to be learnt, its popularity and its wide coverage of mocking functionalities. Thus, we have extended Mockito in order to be able to use it in MAS environments.

Given that we are interested in simulating the behaviour of agents, we have created several classes which provide a simple FIPA interface. In particular, we provide these three agent classes:

- *ResponderMockAgent:* mock agent that replies to predetermined incoming messages.
- *MediatorMockAgent:* mock agent that sends a message to a third agent based on predefined filters.
- *ListenerMockAgent:* mock agent that just receives messages.

Mock agents allow the specification of the simulated behaviour using Mockito constructions. Here follows an example.

```
when(mockAgent.processMessage(
  eq(‘‘REQUEST’’),
  eq(‘‘VoD Loss Rate’’)))
  .thenReturn(‘‘INFORM’’,‘‘Loss Rate=0.2’’)
```

2.3.2 Agent testing

Our approach to agent level testing has consisted of extending JUnit framework in order to be able to test MAS systems. Mapping rules have been defined in order to provide full traceability of acceptance tests defined previously. Thus, *JBehave* has been extended with this purpose. Mapping rules [27] provide a standard mapping from scenarios to test code. In JBehave, a *user story* is a file containing a set of *scenarios*. The name of the file is mapped onto a user story class. Each scenario

step is mapped onto a test method using a Java annotation. This Java annotation text provides the name for the test method.

In BEAST, a *stakeholder user story* (obtained in *System Behaviour Specification* phase) is a file that contains *agent stories* (obtained in *MAS Behaviour Specification* phase). These *agent stories* are files which contain a set of *scenarios*. BEAST Tool translates a "Given/When/Then" scenario to a test case class which extends JBehave JUnitStory class and contains three key methods which facilitates the connection to the MAS platform:

- *setUp* method. The "Given" scenario condition previously defined in natural language is translated into Java. This method typically initialises agents and configures their state (goals, beliefs, ...) as well as initialises the environment.
- *launch* method. The "When" scenario condition is implemented in Java. This method generates and schedules the trigger event to start the test.
- *verify* method. The "Then" scenario condition is checked here. The expected states (goals, beliefs, etc.) are checked in this method once test execution is over.

In order to provide MAS platform independence, a testing interface selector has been defined, so called *PlatformSelector* (see Fig. 4). This selector provides the proper platform access according to the MAS framework specified in the configuration file. This access consists of three interfaces that should be implemented in order to integrate a MAS platform:

- *Connector* interface provides an abstract interface to agent managing functions (launch platform, start an agent, etc.).
- *Messenger* interface declares methods for sending and receiving messages from the platform.
- *Agent Introspector* interface provides access to the agent model (such as goals, beliefs, etc.).

These interfaces have been implemented for JADE 4.0 [5] and JADEX 2.0 [7]. A requirement for integrating an agent platform is that it provides methods for external interaction. For example, the integration of JADEX 0.96 was too complex to be carried out because of the lack of these interfaces.

2.4 MAS level testing

The *MAS level testing* phase has two purposes. First of all, once agents have been developed, integration testing can be done replacing mocks by the real systems. Second, *emergent features* should be validated in the developed scenario Simulation techniques can complement this phase to simulate different system configurations.

Currently we do not provide specific facilities for this phase and will be developed as future work. Nonetheless, we would like to point out that once mocks objects have been replaced by the real entities they emulate, business requirements

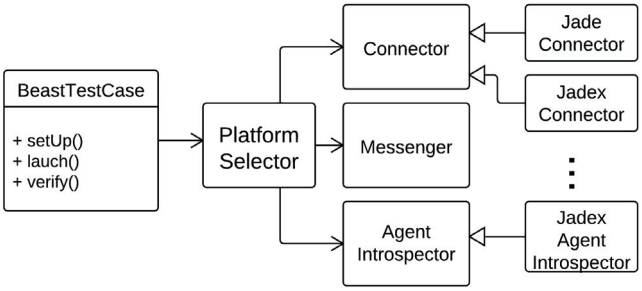


Figure 4: Beast Test Case

can be tested on the real system. Thus, acceptance testing is straight forward, and stakeholders' expectations can be checked without discussing about ambiguities or omissions in the requirements document. The main benefit of BDD techniques is the continuous validation of user requirements in each iteration. This helps for refining iteratively requirements based on the current project advance and available resources.

3 Case Study: MAS for fault diagnosis on Fiber To The Home (FTTH) scenario

To properly frame our proposed BEAST Technique, a network management project was chosen to examplify the use of BEAST. In this example, our stakeholder is a network operator company which wants a tool to reduce the management cost of FTTH networks.

The first step of the project is to write a high level project proposal and to explore different possible approaches to solve the problem. The result of this phase is that the solution that best fits the problem is a MAS architecture. The project and BEAST Methodology, assisted by the BEAST Tool, is used in the project. For exemplification purposes, it is exposed how tests can be implemented in JADEX [7] framework.

So, one of the next tasks that the development team has to do is to arrange a meeting with the stakeholder to specify a set of requirements. In this meeting, the main requirements of the stakeholder are written in BDD format (see Sect. 2). Table 1 shows an example of one gathered requirement.

Notice here that the stakeholder does not know anything about the solution, in this case, a Multi-Agent Systems (MAS). So, the written requirements do not refer at all to the final agents. These requirements are the *acceptance tests* of the project.

Once the developers have the requirements, they can write even more test cases in BDD format. Let us suppose at this point that the development team has identified two agent roles (Probe and Diagnosis) for implementing the requested behaviour as shown in Table 1).

The first role, *Probe Agent*, is responsible for monitoring the service quality,

```
As a operator network,
I want to have a system to diagnose faults root cause
So that time-to-repair is below the SLA with the customer.

Scenario: System diagnoses a QoS decreasing failure
Given a user that has a Video On Demand (VoD) service connected through
  a FTTH access network and the user requests a film from the streaming
  server,
When loss rate is higher to 1%, latency is higher to 150ms or jitter
  is higher to 30ms,
Then the system must diagnose the root cause of fault is
  'Damaged Fibre', 'Inadequate Bandwidth' or 'Damaged Splitter'.
```

Table 1: Stakeholder Requirements Example

```
As a Diagnosis Agent,
I want to diagnose failures when I receive a symptom
So that I am able to report the diagnosis result to other agents.

Scenario: Diagnosis Agent diagnoses Damaged Splitter
Given a 'high loss rate' symptom is received from a Probe Agent,
When two or more geographically close users have loss rate higher to 1%,
Then the Diagnosis Agent must diagnose the root cause of the
  problem is 'Damaged Splitter'.
```

Table 2: Developers Test Cases

while the second role, *Diagnosis Agent*, is responsible for diagnosing faults. These two roles will be implemented in a distributed fashion. Thus, the previous *acceptance tests* is further refined and, as a result, several detailed test cases are obtained, shown in Table 2).

These tests can be further refined and more specific test cases can be defined for obtaining a suitable testing coverage of the desired agent behaviour.

At this stage, the developer has not developed yet *Probe* and *Expert* agents. Since they are required for implementing the aforementioned tests, a mocking facility of BEAST Tool will be used. As previously introduced, BEAST Tool includes several Mock patterns. In this case, the *Mediator Mock Agent* is suitable for implementing both *Probe* and *Expert* agents. Thus, this mock is configured for sending symptoms and network information, respectively.

about the network status around the location of the user. Table 3 shows a sample of an implementation of the *setUp* method in the BEAST test case class.

```
public void setUp() {
 startAgent("DiagnosisAgent","DiagnosisAgent.agent.xml");
 AgentBehaviour mockBeh =  mock(AgentBehaviour.class);
 when(mockBeh.processMessage(eq(INFORM),
      eq("Generate High Loss Rate Symptom")))
  .thenReturn("DiagnosisAgent", INFORM, "Loss rate=0.15");
 MockConfiguration mockConf = new MockConfiguration();
 mockConf.addBehaviour(mockBeh);
 MockManager.startMockJadexAgent("ProbeMockAgent","MediatorMock.agent.xml",
   mockConf,this);
}
```

Table 3: setUp method implementation

```
public void launch() {
 sendMessageToAgent("ProbeMockAgent", INFORM, TRIGGER_EVENT);
 setExecutionTime(2000);// Waiting time in milliseconds
}
```

Table 4: launch method implementation

```
public void verify() {
 checkAgentsBeliefEquealsTo("DiagnosisAgent",ROOT_CAUSE,DAMAGED_SPLITTER);
}
```

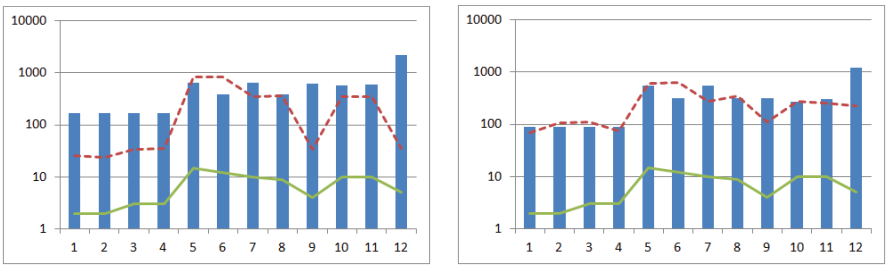
Table 5: verify method implementation

In a similar way, the other two methods of the test class are programmed. The *launch* method of the test case generates a trigger event that initiates the test and fixes the test duration(see Table 4). Then, the *verify* method consists of checking the expected status of the agent under test (see Table 5).

After this test coding task, tests can be launched using standard development tools, since it is a standard JUnit test.

4 Evaluation

The results of the proposed BEAST Methodology have been evaluated in a quantifiable way using source code metrics. In particular, we have measured the number of test code lines required to implement tests. One of the most important benefits of developed BEAST Tool is that automatically creates a wrapper of the MAS platform and allows developers to interact with a friendly interface simplifying the



(a) JADEX evaluation (b) JADE evaluation

Figure 5: Test code lines (Y axis) per Test Case (X axis) comparison for JADEX (a) and JADE (b)

implementation of tests. These metrics are strongly associated with the test implementation time that a developer consumes during this phase of development. The savings in number of code lines and in percentage are shown because they are quantifiable objective data, in comparison the time to develop a test that depend on the programming skills of the developer.

BEAST Tool is already adapted to test JADE [5] and JADEX [7] MAS and the evaluation process has been carried out for both platforms. To simplify the comparison, only twelve different test cases have been chosen for this evaluation. These test cases are quite different among them, some of them use Mock Agents, different number of agents are involved in each one, etc.

Notice that graphics shown in Fig. 5(a) and in Fig. 5(b) are in logarithmic scale. In both graphics, columns represent the code lines of agents under testing and lines represent the code lines required to implement the test with (solid line) and without (dashed line) BEAST Tool. Fig. 5(a) shows the benefits of BEAST Tool in number of lines required to implement the same test using BEAST Tool and without it for JADEX. The improvement is, in average, 247.91 lines per test, i.e. a saving of 97.22%. Fig. 5(b) shows the same comparison for JADE with similar test cases. The improvement in this case is, in average, 262,08 lines per test, i.e. a saving of 97,36%.

Nevertheless, the main advantages of the BEAST approach do not come from the saving in coding tasks. The main benefit of our approach is the significant increase in communication between all the stakeholders of the software project, thanks to the usage of an ubiquitous language and its formalisation using BDD templates. Moreover, this BDD technique is also used for developing MAS, which also improves communication among developers.

5 Conclusions and future work

Advances in MAS testing has been introduced in this paper using BEAST Technique and BEAST Tool. BDD perfectly fits its specification describing behaviours of a determined agent or of a set of agents. Furthermore, the use of BDD facilitates the communication between stakeholders and designers or developers which, usu-

ally, it is a gap between both of them. To solve this problem, BEAST Technique establishes that stakeholders must generate a set of behaviour specifications that describes the whole system. Later, MAS designers must generate the set of agent behaviour specifications that fits the solution of the problem. These behaviours in BDD format are translated automatically with BEAST Tool to JUnit test cases. During this process, text plain in natural language is always available to facilitate the specification compression and communication between both parts.

Other common issue in MAS development is the need of other agents to test the behaviour of an Agent Under Test (AUT). Since these agents are not developed yet, BEAST uses Mock Agents to allow developers to ensure the correct behaviour of an AUT. To add flexibility to Mocks, Mockito framework is integrated with BEAST Tool to allow the use of Mock Agents, Mock Web Services, Mock Java Objects, etc.

Besides, the use of MAS testing techniques or methodologies are commonly strongly connected to a specific MAS platform [9, 13, 23]. BEAST Tool are easily adaptable for any MAS framework. Currently, JADE and JADEX testing are supported.

For future work, we will study in depth the use of simulations for MAS Level Testing (see Sect. 2) in order to cover all possible test like non-functional tests, for example, performance of all agents working together or the achievement of high level goals that can be only in a collaborative way. For this purpose, MASON framework will be explored.

We also plan to improve BEAST Tool to support other MAS frameworks, like JASON [6], to maximize the scope of the developed tool.

Finally, other interesting issue is to evaluate other non-BDD approaches for system specifications provided by a stakeholder, like FIT [19], that support the specification of test cases with concrete examples that provide real data. This first step of the methodology is really important and the capability for stakeholders to choose the format of system specifications can be a key point for a successful project.

References

[1] Mockito Framework. `http://mockito.org`. Accessed March 25, 2012.

[2] G. Adzic. *Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing*. Neuri Limited, United Kingdom, 2009.

[3] G. Adzic. *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications, 2011.

[4] N. Agarwal and U. Rathod. Defining success for software projects: An exploratory revelation. *International Journal of Project Management*, 24(4):358–370, May 2006.

[5] F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*, volume 5 of *Wiley Series in Agent Technology*. Wiley, 2007.

[6] R. H. Bordini, M. Wooldridge, and J. F. Hübner. *Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)*. John Wiley & Sons, 2007.

[7] L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A BDI-Agent System Combining Middleware and Reasoning. In R. Unland, M. Calisti, M. Klusch, M. Walliser, S. Brantschen, and T. Hempfling, editors, *Software Agent-Based Applications, Platforms and Development Kits*, Whitestein Series in Software Agent Technologies and Autonomic Computing, pages 143–168. Birkhäuser Basel, 2005.

[8] N. Clynch and R. Collier. Sadaam: Software agent development-an agile methodology. In *Proceedings of the Workshop of Languages, methodologies, and Development tools for multi-agent systems (LADS007), Durham, UK*, 2007.

[9] R. Coelho, U. Kulesza, A. von Staa, and C. Lucena. Unit testing in multi-agent systems using mock agents and aspects. In *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*, SELMAS '06, pages 83–90, New York, NY, USA, 2006. ACM.

[10] O. Dikenelli, R. Erdur, and O. Gumus. Seagent: a platform for developing semantic web based multi agent systems. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1271–1272. ACM, 2005.

[11] E. Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[12] I. García-Magariño, A. Gómez-Rodríguez, J. Gómez-Sanz, and J. González-Moreno. Ingenias-scrum development process for multi-agent development. In *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, pages 108–117. Springer, 2009.

[13] J. Gómez-Sanz, J. Botía, E. Serrano, and J. Pavón. Testing and Debugging of MAS Interactions with INGENIAS. In M. Luck and J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering IX*, volume 5386 of *Lecture Notes in Computer Science*, pages 199–212. Springer Berlin / Heidelberg, 2009.

[14] P. Hamill. *Unit test frameworks*. O'Reilly, first edition, 2004.

[15] B. Haugset and G. Hanssen. Automated acceptance testing: A literature review and an industrial case study. In *Agile, 2008. AGILE '08. Conference*, pages 27 –38, aug. 2008.

[16] Z. Houhamdi. Multi-agent system testing: A survey. *International Journal of Advanced Computer*, 2011.

[17] A. Marnewick, J.-H. Pretorius, and L. Pretorius. A perspective on human factors contributing to quality requirements: A cross-case analysis. In *Industrial Engineering and Engineering Management (IEEM), 2011 IEEE International Conference on*, pages 389 –393, dec. 2011.

[18] R. C. Martin and G. Melnik. Tests and requirements, requirements and tests: A möbius strip. *IEEE Software*, 25(1):54–59, 2008.

[19] R. Mugridge and W. Cunningham. *Fit for developing software: framework for integrated tests*. Prentice Hall, 2005.

[20] C. D. Nguyen. *Testing Techniques for Software Agents*. PhD thesis, 2009.

[21] C. D. Nguyen, A. Perini, C. Bernon, J. Pavón, and J. Thangarajah. Testing in multi-agent systems. In *Proceedings of the 10th international conference on Agent-oriented software engineering*, AOSE'10, pages 180–190, Berlin, Heidelberg, 2011. Springer-Verlag.

[22] C. D. Nguyen, A. Perini, and P. Tonella. Goal oriented testing for mass. *Int. J. Agent-Oriented Softw. Eng.*, 4(1):79–109, Dec. 2010.

[23] D. Nguyen, A. Perini, and P. Tonella. A Goal-Oriented Software Testing Methodology. In M. Luck and L. Padgham, editors, *Agent-Oriented Software Engineering VIII*, volume 4951 of *Lecture Notes in Computer Science*, pages 58–72. Springer Berlin / Heidelberg, 2008.

[24] D. North. JBehave. A framework for Behaviour Driven Development (BDD). `http://jbehave.org`. Accessed March 28, 2012.

[25] D. North. Introducing: Behaviour-driven development. `http://dannorth.net/introducing-bdd`, 2007. Accessed March 28, 2012.

[26] K. Schwaber. Scrum development process. In *Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA*, pages 117–134, 1995.

[27] C. Solis and X. Wang. A study of the characteristics of behaviour driven development. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 383 –387, 30 2011-sept. 2 2011.

[28] J. Thangarajah, G. Jayatilleke, and L. Padgham. Scenarios for system requirements traceability and testing. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '11, pages 285–292, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.

[29] A. Tiryaki, S. Öztuna, O. Dikenelli, and R. Erdur. Sunit: A unit testing framework for test driven development of multi-agent systems. *Agent-Oriented Software Engineering VII*, pages 156–173, 2007.

[30] M. Wynne and A. Hellesy. Cucumber. Behaviour driven development with elegance and joy. `http://cukes.info`. Accessed March 28, 2012.

[31] Z. Zhang, J. Thangarajah, and L. Padgham. Automated testing for intelligent agent systems. In M.-P. Gleizes and J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering X*, volume 6038 of *Lecture Notes in Computer Science*, pages 66–79. Springer Berlin / Heidelberg, 2011.

Dynamic Monitoring for Adapting Agent Organizations

Juan M. Alberola, Luis Búrdalo, Vicente Julián, Andrés Terrasa, and Ana García-Fornes

Departament de Sistemes Informàtics i Computació,
Universitat Politècnica de València, Camí de Vera s/n. 46022, València. Spain,
{*jalberola,lburdalo,vinglada,aterrasa,agarcia*}*@dsic.upv.es*

Abstract

Multiagent technologies are usually considered to be suitable for constructing agent organizations capable of running in dynamic and distributed environments, and being able to adapt to changes as the system runs. The necessary condition for this adaptation ability is to make agents aware of significant changes in both the environment and the organization. This paper presents Trace&Trigger, an adaptation framework for agent organizations consisting of a monitoring mechanism, which helps agents detecting adaptation requirements dynamically at run time, and an adaptation assistant, which provides organizational agents with information related to the costs and benefits of carrying out an adaptation process at each moment of the execution. This framework intends to overcome some of the problems which are present in other approaches by allowing the dynamic specification of the information that has to be retrieved by each agent at each moment, avoiding the transference of useless information for adaptation deliberation. This framework has been integrated in the Magentix2 multiagent platform and, in order to test its performance benefits for any agent organization, an example based on a market scenario is also presented.

Keywords: monitoring, organization, adaptation, events.

1 Introduction

Nowadays, one of the goals of multiagent systems is to construct systems capable of autonomous and flexible decision-making, and of cooperating with other entities within a society. In these scenarios, dynamic agent organizations which are able to adjust themselves in order to gain advantage in their current environments are likely to become increasingly important [15]. Dynamic agent organizations have to modify/adapt their structure and behavior by adding, removing or substituting components while the system is running and without bringing it all down. In these cases, as pointed out by Dignum et al. in [10], changes in the environment of agents are the ones which trigger reorganization and thus, this dynamic adaptation demands that systems can evaluate their own health in order to find out when an

adaptation is needed. Being able to monitorize an agent organization is important in order to determine *why* and *when* an organization needs to be adapted. According to [12], monitoring is essential in order to be able to detect undesirable behavior that needs to be corrected. However, detecting these changes in the environment is not trivial.

Current approaches for agent organization adaptation propose different techniques for monitoring the organization in order to figure out when an adaptation is required. Most of them have in common that both the internal adaptation logic for deciding when an adaptation is required and the information required to be monitored are usually predefined at design time and cannot be modified during the execution. This restriction assumes that requirements associated to the adaption process are always known in advance. However, as stated in [1], adaptive systems may cause monitoring requirements to change throughout the agent organization's life-span and thus, the information required to be monitored can also change during the execution depending on the current requirements of the system.

Assuming that monitoring needs are static and known in advance at design time makes it difficult to develop dynamic applications which can adapt at run time. It is necessary to count on an adaptive approach which allows to overcome these monitoring limitations imposed by static designs. Thus, an adaptive approach should apply not only to the behavior and structure of the system, but also to the design of the monitoring system [17], specially when dealing with the management of complex systems over long periods of time.

This paper presents Trace&Trigger, an agent organization adaptation framework which consisting of a monitoring mechanism and an adaption assistant. The monitoring mechanism helps agents detecting adaptation requirements dynamically at run time and also feeds the adaption assistant, so that it can provide organizational agents with information related to the costs and benefits of carrying out an adaptation at each moment of the execution.

The rest of the paper is organized as follows. Section 2 details previous work by other authors in the field of monitoring and adapting multiagent systems. Section 3, describes in detail the adaptation framework itself. Section 4, shows an example of how to incorporate the framework to an adapting agent organization and in Section 5, the performance of the organization is evaluated. Finally, Section 6 presents the main conclusions of this work.

2 Monitoring Agent Organizations for Adaptation

Current approaches for organization adaptation propose different techniques for monitoring the organization in order to figure out that an adaptation is required. Some of these approaches define the monitoring process as an automatic response to events that cause an adaptation such as the addition or deletion of a new role, agent, etc. These events cause the agent organization to make the required adjustments in order to fulfill its goals or to improve its performance. Other approaches provide an implicit mechanism for reasoning about the current estate of the organization so that it can decide that an adaptation is required.

Weyns et al. presents in [19] a middleware that provides support for the man-

agement of organization adaptation. In this approach the adaptation is carried out in a distributed way triggered by external events (e.g. when an agents stops playing a role) and changes in the environment (e.g. when the traffic state in a traffic monitoring application changes). Adaptation purposes are specified by means of rules (called laws), which describe how agents can join and leave the organization, as well as how to restructure the organizations by merging and splitting organizations. However, the specification of these rules is carried out at design time. A similar problem is presented in other adaptation approaches such as [9, 18, 13], which define adaptation requirements as rules that are specified at design time and cannot be modified at run time.

Other works provide a more proactive decision mechanism for detecting that an adaptation is required. An example of this support can be viewed in the work of Campos et al. [7, 6]. In this adaptation approach the monitoring process is carried out in a distributed way between assistant agents. Each assistant perceives partial information about a cluster of agents and this information is later shared with other assistants in order to make the decisions. However, in spite of being less reactive than the above referred works, this is still a static monitoring approach, since the information to be retrieved from agents and the other assistants is predefined and cannot be changed while the system is running. In a similar way, the adaptation approach presented in [14] allows the implementation of different agents that are in charge of determining when an adaptation is required. As the monitoring is implemented by agents, these can enter dynamically in the system or they can learn. However, the monitoring mechanisms are not defined by adaptation approach level and should be implemented by the system designer providing his own methods and tools for the specific application.

Static mechanisms that do not consider changes regarding which information has to be monitored may result useful in small application domains with a priori well known organizational structures, but they would not be suitable for large-scale or complex systems. As the number of agents in the organization and their complexity grows, much more information than required is exchanged between agents. Most of this information is not useful at every moment of the execution and only contributes to considerably increase the traffic in the system, specifically in approaches in which a middleware or centralizing entity is the responsible of adaptation deliberation or implementation such as in [8]. Also, the internal logic of agents can become very complex and costly. And finally, situations to trigger an adaptation will not be suited to the system's performance, preventing it from correctly adapt to changing situations.

This work proposes the use of event tracing as a dynamic monitoring mechanism in order to allow agents to specify which information they want to receive at each moment of the execution. In this way, the amount and type of information which each agent has to deal with to determine if an adaptation has to be done will depend on the actual state of the organization and it will also vary as the system adapts.

3 The Trace&Trigger Framework

The Trace&Trigger framework presented in this work has been designed to run on the Magentix2 multiagent platform [11], a platform for open multiagent systems, developed in Java. This platform provides a specific mechanism to obtain the information necessary for any adaptation (Section 3.1). Also, specific mechanisms have been incorporated to let agents determine the costs and benefits of performing an adaptation at run time (Section 3.2), as well as the mechanisms required to carry on the necessary actions to perform that adaptation (Section 3.3).

3.1 Magentix2 Event Tracing Support

In addition to the message-based communication layer, Magentix2 also provides an event tracing based communication layer, which allows agents to throw and receive trace events at run time. These event tracing facilities have been incorporated to the platform according to the TRAMMAS abstract model and architecture [5].

The model establishes a *publication/subscription* mechanism, by which (1) any agent can publish the types of events which it is able to generate (in advance of generating them), and (2) any agent can subscribe to those trace events on which it is interested (before starting to receive them). This publication/subscription mechanism is dynamic, in the sense that, at any time during the execution, agents can change their publications and subscriptions. In order to give support to this publication/subscription mechanism, trace events are offered to agents in the system as *tracing services*, in a similar way to traditional services offered in the multiagent system. Agents have to send an ACL message to an specific agent, called *Trace Manager* (TM), whenever they want to publish or unpublish their available tracing services, as well as when they want to subscribe to a tracing service or to unsubscribe from it. The TM agent interacts with the Magentix2 communication layer so that trace events thrown by an agent are only injected to the network if there is any agent interested in receiving them and no trace event is received by any agent unless it has previously requested it.

3.2 Organization Management Module

Magentix2 provides support to virtual organizations by means of the THOMAS architecture [16], which defines flexible services that can be used by agents. This architecture has been used to define the organization's management, as well as the services provided by agents. The THOMAS architecture is composed by a *Service Facilitator* (SF) and an *Organization Management System* (OMS). The SF allows for the registration and search of services provided by internal or external entities by following Service-Oriented Architectures guidelines. The OMS is in charge of the management of the organizations, taking control of their underlying structure, services provided by the agents and their relationships. This module allows for the development of open and dynamic multiagent systems, where agents are able to dynamically enter and leave the system, change their services, their relationships or the roles that they play in the organizations. A special agent, which is called manager agent, is defined for managing the execution of each organization. This

agent has complete information about the current state of the organization and has permission to interact to the SF and OMS for changing it.

3.3 Adaptation Module

Being in charge of coordinating every adaptation process in an agent organization, the manager agent estimates the impact for each potential change. This impact represents the costs/benefits that the application of an individual change (such as the addition or deletion of a service) would cause, not only to those components involved in the change, but also to other components in the organization. Furthermore, it also shows the cost for carrying out the application of this change.

The *Reorganization Facilitator service* (RF) [2] is the service in charge of calculating at any time which adaptation has the lowest impact for the system. Individual impacts which are calculated by the organization manager agent are transferred to the RF in order to calculate the organization adaptation. This service implements an adaptation mechanism based on organization transitions in order to obtain the best adaptation from a current organization.

This process finds the organization whose transition impact is the lowest and the sequence of steps required to achieve it. Several changes can be considered by using the Multi-Transition Deliberation Mechanism (MTDM) presented in [4]. This mechanism calculates transitions in different dimensions (roles, services, relationships, agent population) to other organizations with high expected utility based on the cost for transition to these organizations. The MTDM decides which transition is finally implemented and provides the sequence of changes required to carry out the transition.

3.4 Adaptation Life-Cycle

Figure 1 shows how agents in the multiagent system interact with each other and make use of the different facilities provided by the Trace&Trigger framework in order to evaluate the state of the system at run time, calculate the costs and benefits of any potential adaptation and carry on that adaptation.

The organization manager needs to obtain certain information referred to the organization performance at run time. This organizational knowledge which the manager agent possesses is used in order to estimate the adaptation impacts of individual changes. The monitoring of the organization behavior is carried out by means of the event tracing support provided by Magentix2. To share their relevant information, agents in the organization publish their tracing services by sending an ACL message to the TM. Those agents in the system which are interested in that information, subscribe to those tracing services by requesting it to the TM via an ACL message too. Since information required for adaptation deliberation can change at run time, the organization manager agent sends requests to the TM agent for subscribing or unsubscribing dynamically, according to these requirements. In this way, the organization manager agent retrieves all the information it needs at each moment in a transparent way for the rest of the agents.

Organizational agents may also require some runtime information regarding the organization. These agents can also subscribe to tracing services and unsubscribe

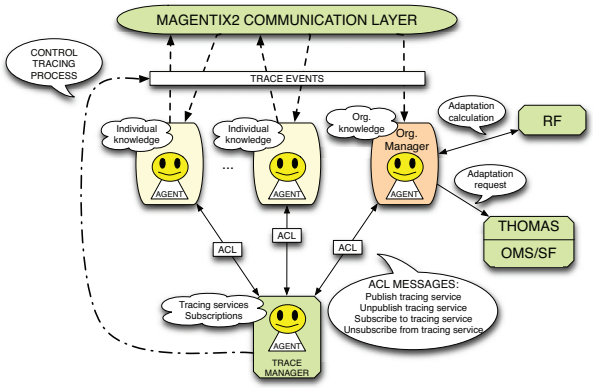


Figure 1: Trace&Trigger framework

from them. In this way, organizational agents can carry out tasks that do not affect other agents in the organization and that do not require the supervision of the organization manager. With the information received from the system, the organization manager has to determine which specific changes can be carried out. In order to do so, the organization manager has to interact with the RF service, which provides a sequence of changes that could be applied in order to improve the organization performance. If there is any promising adaptation that could be applied, the organization manager can interact with the OMS and the SF services in order to carry out this adaptation.

4 Case of study

To help on the demonstration of how the Trace&Trigger framework can improve the adaptation capabilities of an agent organization, a case of study based on a market domain has been implemented. The following conceptual elements are considered in this domain: factories, which generate products, and enterprises, which are able to sell these products to consumers as well as providing stock to other enterprises. In this case of study, the set of enterprises are designed as an agent organization which objective is to make as much profit as possible, attending to the products that are being sold by the enterprises. The organization can improve its performance at run time by means of adapting to the needs and demands of the market.

The agent organization has been modelled following the notation presented in [3]. At a given moment t, the agent organization is composed as a set of agents $A^t = \{a_1 \dots a_n\}$, which represent the different enterprises. Each agent a_x is able to provide a set of services $S^t(a_x)$, which are a subset of all of the services provided by the organization: $S^t(a_x) \subseteq S^t = \{s_1 \dots s_p\}$. Each agent a_x that provides a service s_y at a given moment t is represented as $provider^t(a_x, s_y)$ and has associated a current stock $stock^t(a_x, s_y)$. This stock is the maximum number of products that a_x can sell or provide to other agents at time t. Each agent is connected to other agents by acquaintance relationships which allow agents to share their stock to

other agents. An acquaintance relationship $acquaintance^t(a_z, a_x)$ allows an agent a_z being a stock provider of service s_y for agent a_x at time t. An agent a_x that provides a service s_y at time t has associated a list of stock providers, represented as $SP^t(a_x, s_y) = \{a_z, \ldots, a_n\}$.

At each time step, the organization passes through three states: *Serve* (state S), *Restock* (state S') and *Reorganize* (state S''). At the beginning of each time step, the organization remains on a state S for receiving consumer requests. In this state, each agent a_x receives a number of requests for each service s_y it provides, represented as $requests^t(a_x, s_y)$. Sales of a product s_y carried out by a provider a_x at time t is represented as:

$$sales^t(a_x, s_y) = \begin{cases} requests^t(a_x, s_y) & \text{if } stock^t(a_x, s_y) > requests^t(a_x, s_y) \\ stock^t(a_x, s_y) & \text{otherwise} \end{cases}$$

These sales cause the stock to be reduced: $stock^t(a_x, s_y) - sales^t(a_x, s_y)$.

After receiving the consumer requests, the organization reaches the state S', in which agents have to restock their products depending initially on the sales that were carried out in the previous state: $restock^t(a_x, s_y) = sales^t(a_x, s_y)$. Each agent tries to restock their services through one of their stock providers.These providers are requested sequentially until one of them agrees restocking the required amount. If agent a_x and stock provider a_z reach an agreement, the required stock is transferred from a_z to a_x, incurring in a transportation cost for each individual product represented as $tcost(a_z, a_x, s_y)$. If any of the stock providers is able to restock its demand, finally the agent requester a_x can restock the product directly from the specific factory $F(s_y)$ as a last resort. This requires a higher transportation cost that is represented as $tcost(F(s_y), a_x, s_y)$.

The agent a_x may in turn receive requests from other agent a_w that requests restock for a service s_y. In this case, the agent a_x only agrees to restock a_w if the amount requested is less than its current stock: $stock^t(a_x, s_y) \leq restock^t(a_w, s_y)$; and if a_x can restock in turn this amount from its provider (being a_z or $F(s_y)$ according to the above paragraph). If the restock is agreed, then the available stock of a_x is reduced to $stock^t(a_x, s_y) - restock^t(a_w, s_y)$ and the restock required by a_x is increased according to the stock required to be transferred $transf^t(a_x, s_y)$, denoting this later term the amount of products transferred to other agents.

Once all the restock agreements are carried out, each agent a_x restocks all the services from one of its stock providers or from the factory. Finally, the last state S'' represents the state of adaptation deliberation. In this state, the organization manager tries to distribute the services to agents in order to improve the organization's profit. This profit is measured as the sales carried out by each agent a_x for each service s_y depending on the sale price of the service $price(s_y)$ and the transportation cost required to restock the sold amount:

$$P^t(O) = \sum_{a_x \in A^t} \sum_{s_y \in S^t} sales^t(a_x, s_y) \times (price(s_y) - tcost(provider(a_x, s_y), a_x, s_y)$$

The changes that are considered in this example are the addition and deletion of services. Therefore, as we stated in Section 3.3, the manager must estimate the impact of these changes.

4.1 Adaptation Impact Estimation

The impact estimation involves the benefits caused by the adaptation, the costs associated to the adaptation, and how this adaptation would influence all the components of the organization. Here we omit some details of the costs and benefits specification [3] and only show the representation of the adaptation impact in the market example. The addition of a new service in an agent a_x implies to reduce the maximum stock of the rest of the other services provided by this agent, from $\frac{SMAX}{n}$ to $\frac{SMAX}{n+1}$, being n the number of services provided by the agent and $SMAX$ a constant defined for all the agents. We can estimate how the addition of a new service would have affected the sales of each service s_y during the previous period between $t-1$ and t. These sales would have been limited to the new stock, causing a sales opportunity cost defined as follows:

$$o\_sales^t(a_x, s_y) = \begin{cases} sales^t(a_x, s_y) - \frac{SMAX}{n+1} & \text{if } sales^t(a_x, s_y) > \frac{SMAX}{n+1} \\ 0 & \text{otherwise} \end{cases}$$

This opportunity cost represents the sales of service s_y that would not have been carried out if another service would be added. The profit associated to this cost $P(o\_salest^t(a_x, s_y))$ depends on the price of the service and on whether these sales have been restocked from a stock provider or from factory. Apart from this cost, the addition of a new service would have affected the transferences to other agents $transf^t(a_x, s_y)$. Similar to the sales opportunity cost, a transference opportunity cost can be defined, which is also limited to the new stock:

$$o\_transf^t(a_x, s_y) = \begin{cases} restock^t(a_x, s_y) - \frac{SMAX}{n+1} & \text{if } restock^t(a_x, s_y) > \frac{SMAX}{n+1} \\ 0 & \text{otherwise} \end{cases}$$

The transference opportunity cost represents the restock to other agents of service s_y that could not be carried out if another service would be included. As we stated previously, the restock of a service s_y includes the sales of this service and the transferences to other agents. The profit associated to this cost $P(o\_transf^t(a_x, s_y))$ depends on the difference between the transference cost from a stock provider agent and the transference cost from factory $F(s_y)$.

In addition, if agent a_x would have provided the new service s_n, some estimated sales $estimated^t(a_x, s_n)$ would have been carried out. These sales have an associated profit $P(estimated^t(a_x, s_n))$, which can be measured by considering the transportation cost from factory, which represents the worse case. Furthermore, other agents that also provide this service s_n could be negatively affected if a new agent is providing the same service. This can be represented as a sales opportunity cost associated to these other agents $o\_sales^t(a_z, s_n) \forall a_z, s_n \in S^t(a_z)$. This cost represents the possible sales lose in these agents. Finally, the addition of a new service has an associated fixed cost for setting up this service, that can be represented as $up(a_x, s_n)$. Aggregating all this information, the impact of adding a new service s_n to an agent a_x is represented as $I_A(a_x, s_n)$:

$$P(estimated^t(a_x, s_n)) - \sum_{s \in S^t(a_x)} \left(P(o\_sales^t(a_x, s)) + P(o\_transf^t(a_x, s))\right) - \sum_{a_z \in A^t} P(o\_sales^t(a_z, s_n)) - up(a_x, s_n)$$

In contrast than the addition of a service, the deletion of a service implies to increase the maximum stock of the rest of the services, from $\frac{SMAX}{n}$ to $\frac{SMAX}{n-1}$, being n the

number of services provided by the agent. Thus, if the stock of a service s_y during the period between $t-1$ and t have been dropped to 0, the agent manager could estimate that a higher number of extra sales would be carried out with a bigger stock. This value is represented as $estimated^t(a_x, s_y)$, which has an specific profit associated $P(estimated^t(a_x, s_y))$.

Furthermore, if an agent deletes a service s_p already provided, the sales associated to this service $sales^t(a_x, s_p)$ as well as the transferences to other agents $transf^t(a_x, s_p)$ would have not been carried out. Therefore, the specific profit associated to these can be estimated depending on whether the service s_p has been restocked from factory or not. In addition, other agents that also provide this service s_p could be positively affected if an agent stops providing this service. This can be represented as a negative opportunity cost associated to these other agents $o\_sales^t(a_z, s_p) \forall a_z, s_p \in S^t(a_z)$. This represents the possible sales gain in these agents. Finally, the deletion of a service has an associated fixed cost for turning off this service, that can be represented as $off(a_x, s_p)$. Therefore, the impact of deleting a service s_p already provided by agent a_x is represented as $I_D(a_x, s_p)$:

$$\sum_{s \in S^t(a_x)} P(estimated^t(a_x, s)) - \big(P(sales^t(a_x, s_p)) + P(transf^t(a_x, s_p))\big) - \sum_{a_z \in A^t} P(o\_sales^t(a_z, s_p) - off(a_x, s_p)$$

Depending on the amount of services provided by each agent, the manager considers the possibility of adding a service if the agent provides less than δ services and any of the services provided are restocked with more amount that the actually needed. This would mean that it could be beneficial that the agent could add other service and reduce the stock of the current ones. In contrast, the deletion of a service is considered if the agent provides δ services or more, and the stock of any of these services has dropeed to 0. This would mean that it could be beneficial to delete some of the current services in order to increase the stock capacity of this high demanded service. In order to deal with how all the information required to adaptation deliberation is retrieved, in the following section we show the use of monitoring mechanism by event tracing.

4.2 Event Tracing Specification

By using event tracing, agents can publish, request, and cancel subscriptions dynamically, in order to send and retrieve only the interesting information at each moment. To allow every agent to know the stock that is available in its stock providers, each agent a_x publish the stock of each service s_y provided at the beginning of each time step. This information is published by means of the $STOCK\_AVAILABLE$ event tracing. All the agents that are interested in receiving this information, i.e. the agents that have associated a_x as a provider of the service s_y, request a subscription to this event. Thus, each agent a_z only receives the specific information that is required at each moment according to the following restriction:

$$TE.type = STOCK\_AVAILABLE \wedge TE.sender = a_x, a_x \in P^t(a_z, s_y)$$

In order to manage the information required for organization adaptation deliberation, the manager requests subscriptions to different events depending on the services provided by each agent. On the one hand, the manager is interested in receiving information of those agents that can add a new service (those which provide

less than δ services) and have requested more restock than the actually required. This is implemented by publishing the $RESTOCK$ event tracing in the state S'. By using this event tracing each agent publishes the information of the restock that it is carrying out. Therefore, the manager is interested in receiving information of those agents that can add a new service and that have requested a restock amount that is higher than the required:

$$TE.type = RESTOCK \wedge TE.sender = a_x, |S^t(a_x)| < \delta \wedge TE.value > \tau$$

We define this restriction as a threshold τ, which represents the estimated stock required for the next time step in order to satisfy the sales and restocks received in the current time step, according to the stock that is still available: $\tau = \left(\frac{SMAX}{n} - stock(a_x, s_y)\right) - stock(a_x, s_y)$. Regarding the deletion of services, the manager is interested in receiving information about those agents that are able to delete a service (those which provide δ services or more) and have sold all the stock of any service. We can use the $STOCK\_AVAILABLE$ event tracing published by agents in order to obtain this information:

$$TE.type = STOCK\_AVAILABLE \wedge TE.sender = a_x, |S^t(a_x)| \geq \delta \wedge TE.value \leq \sigma$$

Similar to the previous event tracing we define a threshold, which could be modified at runtime, defined as $\sigma = 0$. Finally, the manager also needs to know how many restocks are carried out from factories in order to calculate the profit depending on the transportation costs. This is represented as events $FACTORY\_REQUEST$ that are published by agents and are sent when the agent carries out a request to an specific factory:

$$TE.type = FACTORY\_REQUEST \wedge TE.sender = any$$

5 Evaluation

In this section we show different experiments to measure the performance of the adaptation framework. For these experiments we define an organization composed of ten agents. The distribution of services between agents is randomly generated in which four agents provide two services s_0 and s_1 and six agents provide only one of these services. The acquaintance relationships between agents are also generated such that each agent has zero, one, or two different stock providers. The constant $SMAX$ used for dividing the stock between the services provided is 100, and the initial requests received at each agent are randomly distributed between 30 and 70 requests. Then, in a period of time between two consecutive time steps, each agent a_x receives requests for each service s_y according to the following formula:

$$requests(a_x, s_y)^{t+1} = requests(a_x, s_y)^t \times random(0.95, 1.05)$$

As can be observed, the number of requests received in two consecutive time steps may change in an interval of $\pm$ 5%. However, in a real scenario, demand of services may change at any time. Therefore, in order to simulate a more realistic and dynamic execution, we varied the demand of services during the 50 time-steps.

Specifically, at $t = 15$ the service s_1 becomes more demanded than s_0, while at $t = 30$ this demand changes again, becoming the s_0 the most demanded service.

Figure 2 shows a comparison of the organization profit of an adaptive organization and a static organization. The adaptive organization represents an organization that is able to change the services provided by agents according to the information monitored at runtime, while the static organization maintains the initial configuration during all the iterations. The profit drawn in the figure represents the mean profit of 10 executions and the profit of a single execution with the moments in which an adaptation is carried out.

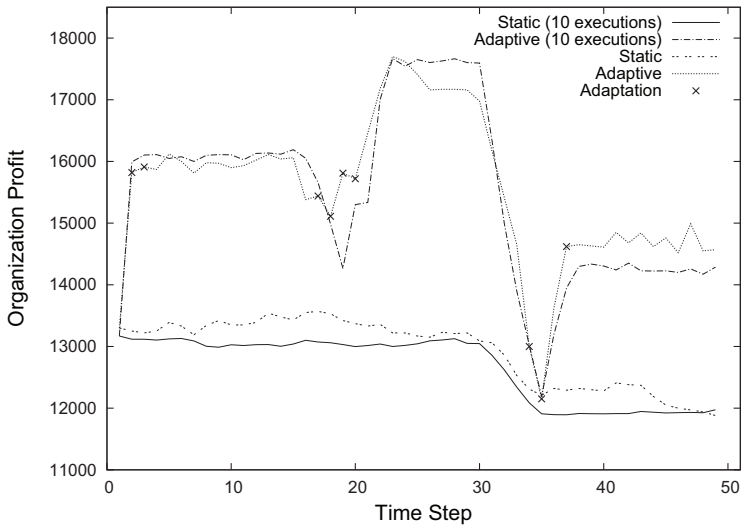


Figure 2: Organization profit

We can observe that the adaptive organization tries to achieve a better distribution of services during the initial iterations, until achieving a profit round to 16000, while the static organization profit remains at 13000. From $t = 15$ on, we can observe that the profit of the adaptive organization changes abruptly. This is caused because the distribution of services was focused to well-suit the demand monitored up to this moment. However, from this moment on, the demand of both services changes and the profit obtained by this distribution is not the best performing. As we can observe, the organization adapts itself in order to achieve a distribution of services more suitable according to the demand increase of s_1. These adaptations cause that the organization achieves a distribution that provides a profit around almost 18000. In contrast, the profit of the static organization remains similar due to it is not able to achieve a better distribution of services. The same situation occurs at $t = 30$, when the demand changes again. The former iterations from this moment on, show a high profit decrease in the adaptive organization, but this organization quickly achieves a better distribution for improving the profit. It can be easily observed that with different data demand, the adaptive organization should allways try to find a better distribution of services. However, the contribution of this paper is focused on how this information is obtained. Therefore, following we

evaluate we evaluate the overload caused by the information that is monitored by using the Trace&Trigger framework.

In order to do this, the number of messages and events that are received by the organization manager during the 50 iterations are shown in Figure 3 (a). In this figure, a dynamic and static monitoring comparison is shown. As can be observed, the messages received is greater without using event tracing, and the differences between both approaches become higher according to the population of agents increases. This proves that the information required to be monitored in dynamic systems such as this case study, may be different along the organization's life span. Therefore, a dynamic monitoring in which only the required information is retrieved, considerably reduces the traffic load in the system. Note that in the static approach, the information required to be monitored must be specified at design time and thus, a lot of information is transferred that finally is not used by the manager in order to consider adaptation. Furthermore, despite in this example we maintained the thresholds δ, σ, and τ constants, these could also be changed at runtime in a transparent way for the agents, without requiering to inform these agents every time the organization manager changes its monitoring requirements.

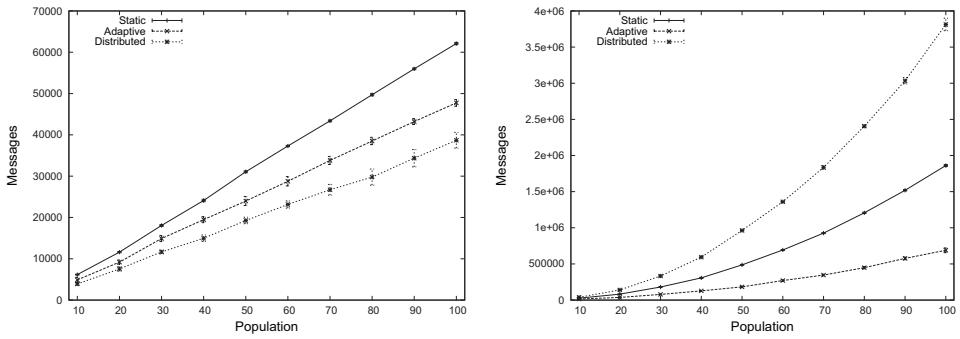


Figure 3: Messages/events received: (a) in a single agent; (b) in the whole organization

In this example, the manager is the agent that receives the biggest amount of information. Therefore, we also want to compare in Figure 3 (a) the performance of a distributed approach, in which the information is not centralized in an individual entity. This figure shows the messages received by a given agent in a distributed approach. It can be observed that in this case, the number of messages received is even lower. However, as there is not any entity that centralizes all the information required for adaptation deliberation, all of the agents should receive a similar number of messages in order to coordinate and synchronize the information. This is important because an individual agent cannot adapt the services offered without taking into account how this change can affect the whole organization.

In order to evaluate this, Figure 3 (b) shows the performance of the three approaches according to the messages exchanged in the whole organization. It can be observed that the dynamic monitoring clearly outperforms the rest of the approaches. We can observe that the distributed approach presents an even higher message traffic, which could overload the framework execution when the population of agents increases. As it can be observed, the number of messages required

to coordinate the adaptation in the distributed system is more than four times greater than the dynamic monitoring approach. In contrast, the differences shown in Figure 3 (a) are much more lower than the differences in the traffic of the whole organization.This could be a clear bottleneck when large systems are executed.

6 Conclusions

Monitoring an agent organization becomes essential to determine that an adaptation is required at an specific moment. Most monitoring techniques for agent organizations which can be found in the literature either rely on predefined rules, which cannot be at run time, or they assume that the information that has to be monitorized does not change as the system executes. As a consequence, these approaches are only appliable in small domains which involve a small number of agents and a priory well known organizational requirements, but cannot be applied in large-scale or complex domains.

The Trace&Trigger adaptation framework proposed in this work allows for the specification at run time of the information that has to be retrieved from the agent organization according to its state a each moment, so that the organization can adapt to those changes. This monitoring is complemented by an adaptation module, which allows to obtain the sequence of the most promising changes, as well as the potential improvement and cost of carrying those changes out in order to well-suit the organization to the current requirements.

Experimental results show how the performance of the agent organization improves when using the proposed framework for adapting to changes in service demands. The global profit obtained by the organization is higher when using the proposed framework. In addition to that, the scalability of the system is improved by reducing the traffic due to the monitoring process. Future lines of research would include deeper studies on agent organizations, considering adaptation not only to changes in services provided by each agent, but also considering changes in the agent population or in the relationships among these agents provided by the MTDM [4]. In addition to that, strategies for improving monitoring requirements attending to past experiences will be studied.

Acknowledgments

This work has been partially supported by CONSOLIDER-INGENIO 2010 under grant CSD2007-00022, and projects TIN2009-13839-C03-01 and TIN2011-27652-C03-01. Juan M. Alberola has received a grant from Ministerio de Ciencia e Innovación de España (AP2007-00289).

References

[1] H. Abdu, H. Lutfiyya, and M. A. Bauer. A model for adaptive monitoring configurations. In *Proceedings of the VI IFIP/IEEE IM conference on network management*, pages 37–1, 1999.

[2] J. M. Alberola, V. Julian, and A. Garcia-Fornes. Cost-aware reorganization service for multiagent systems. In *Proc. 2nd Int. Workshop ITMAS11*, pages 60–74, 2011.

[3] J. M. Alberola, V. Julian, and A. Garcia-Fornes. A cost-based transition approach for multiagent systems reorganization. In *Proc. 10th Int. Conf. on Aut. Agents and MAS (AAMAS11)*, pages 1221–1222, 2011.

[4] J. M. Alberola, V. Julian, and A. Garcia-Fornes. Multi-dimensional transition deliberation for organization adaptation in multiagent systems. In *Proc. 11th Int. Conf. on Aut. Agents and MAS (AAMAS12)*, page In Press, 2012.

[5] L. Búrdalo, A. Terrasa, V. Julián, and A. García-Fornes. TRAMMAS: A Tracing Model for Multiagent systems. In *First International Workshop on Infraestructures and Tools for Multiagent Systems*, pages 42–49, 2010.

[6] J. Campos, M. Esteva, M. Lopez-Sanchez, J. Morales, and M. Salamo. Organisational adaptation of multi-agent systems in a peer-to-peer scenario. *Computing*, 91:169–215, 2011.

[7] J. Campos, M. López-Sánchez, and M. Esteva. Assistance layer, a step forward in Multi-Agent Systems Coordination Support. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1301–1302, 2009.

[8] G. Carvalho, H. Almeida, M. Gatti, G. Vinicius, R. Paes, A. Perkusich, and C. Lucena. Dynamic law evolution in governance mechanisms for open multi-agent systems. In *2nd Workshop on Software Engineering for Agent-oriented Systems*, 2006.

[9] S. DeLoach, W. Oyenan, and E. Matson. A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, 16:13–56, 2008.

[10] V. Dignum, F. Dignum, and L. Sonenberg. Towards dynamic reorganization of agent societies. In *In Proc. of Workshop on Coordination in Emergent Agent Societies*, pages 22–27, 2004.

[11] R. L. Fogués, J. M. Alberola, J. M. Such, A. Espinosa, and A. Garcia-Fornes. Towards dynamic agent interaction support in open multiagent systems. In *Proc. of the 13th Int. Conf. of the Catalan Association for Artificial Intelligence*, volume 220, pages 89–98, 2010.

[12] Z. Guessoum, M. Ziane, and N. Faci. Monitoring and organizational-level adaptation of multi-agent systems. In *In Proc. of AAMAS04*, pages 514–521. ACM Press, 2004.

[13] M. Hoogendoorn and J. Treur. An Adaptive Multi-agent Organization Model Based on Dynamic Role Allocation. In *Proc. of the IAT '06*, pages 474–481, 2006.

[14] J. F. Hübner, J. S. Sichman, and O. Boissier. Using the MOISE+ for a Cooperative Framework of MAS reorganisation. In *SBIA'04*, volume 3171, pages 506–515.

[15] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. University of Southampton, 2005.

[16] M. Rebollo, A. Giret, E. Argente, C. Carrascosa, J. M. Corchado, A. Fernandez, and V. Julian. On the road to an abstract architecture for open virtual organizations. In *Proc. of the 10th Int. Work-Conf. on Art. Neural Networks*, pages 642–650, 2009.

[17] P. L. Ringold, J. Alegria, R. L. Czaplewski, B. S. Mulder, T. Tolle, and K. Burnett. Adaptive Monitoring Design for Ecosystem Management. *Ecological Applications*, 6(3):745–747, 1996.

[18] Z.-g. Wang and X.-h. Liang. *A Graph Based Simulation of Reorganization in Multiagent Systems*. 2006.

[19] D. Weyns, R. Haesevoets, A. Helleboogh, T. Holvoet, and W. Joosen. The MACODO middleware for context-driven dynamic agent organizations. *ACM Transaction on Autonomous and Adaptive Systems*, 5:3:1–3:28, February 2010.

Multi-Agent Oriented Reorganisation within the JaCaMo infrastructure

Alexandru Sorici[1,2], Gauthier Picard[2], Olivier Boissier[2], Andrea Santi[3], and Jomi F. Hübner[4]

[1] *"Politehnica" University of Bucharest , Dpt of Computer Science and Engineering , Bucharest, Romania,* alex.sorici@cti.pub.ro

[2] *Ecole Nationale Supérieure des Mines , FAYOL-EMSE, LSTI , F-42023 Saint-Etienne,* {picard,boissier}@emse.fr

[3] *DEIS, Alma Mater Studiorum , Universita di Bologna , 47521 Cesena (FC), Italy,* a.santi@unibo.it

[4] *DAS-UFSC , Federal University of Santa Catarina, CP 476, 88040-900 Florianópolis SC, Brazil,* jomi@das.ufsc.br

Abstract

Current IT applications are often meant to be used in complex and highly dynamic environments. Multi-Agent Oriented technologies and related programming languages offer high level abstractions that can ease the realisation of such applications. However, providing the means to handle adaptation and endogenous reorganisation, is still a challenging issue even if multiple state-of-the art works propose reorganisation models at the agent level. Adopting a multi-agent oriented approach, in this paper we define a dedicated artifact (tool) that agents use to design and implement transitions to new organisations. The agents' compliance with a specific organisation specification that we formulate helps regulate and coordinate this reorganisation process. We show how this proposal has been realised within the JaCaMo multi-agent oriented programming platform and how this constitutes a step toward really adaptive systems.

Keywords: multi-agent oriented programming, reorganisation, organisation centred multi-agent systems

1 Introduction

Current applications are more and more meant to be used in complex, distributed and highly dynamic environments. Among others, their features stress heterogeneity, context-awareness, anticipation of users' desires and absence of a central control. So, given this context, adaptation becomes the key to face the dynamics and the evolution of situations in which these applications operate. Multi-agent technologies and abstractions, and in particular organisation oriented ones, can provide concepts and tools that give possible answers to this issue. However, even if multiple works in the state-of-the art propose several reorganisation models at the agent level for dealing with system adaptation, it is still an open issue how to put these models in practice: performing the changes at the right time, in the right order, without disrupting the functioning of the entire organisation.

In this paper we introduce a proper infrastructure for managing reorganisation using a multi-agent oriented programming approach based on the JaCaMo [3] framework. The JaCaMo framework proposes a novel multi-agent oriented programming approach and a related development platform for engineering and executing distributed and open software systems by integrating three multi-agent programming dimensions, namely the agent, environment, and organisation levels in a synergistic way. In JaCaMo the use of organisation-centric abstractions (e.g. roles, groups, norms, missions) offers the possibility to express and impose cooperation schemes that govern the functioning of the autonomous agents in the environment. Currently agents have the means to inspect and reason about the organisations in which they partake. However, for being able to support endogenous adaptation [1] of systems by using systematic reorganisational processes, agents, besides abilities to reason on organisational constructs, must also have the proper means to coordinate and cooperatively modify the organisation as soon as they deem it to be improper for the current objectives. In this paper, we propose to realise such a support by using the different first class abstractions related to environment and organisation that are part of a multi-agent programming approach. Our contribution consists in a tool, the reorganisation artifact, and the definition of an organisational specification that regulates and guides agent activity during reorganisation (an *organisation for reorganisation*). We use and extend the JaCaMo platform with these different elements.

The remainder of the paper is organised as follows: In Section 2 we provide the reader with the required background for the JaCaMo multi-agent oriented programming approach upon which is realized our reorganisation proposal. Section 3 presents the rationale and general overview of the envisioned reorganisation process and Section 4 and Section 5 introduce the JaCaMo based infrastructure used to support it. In Section 6 we compare our approach with existing related works in literature. Finally, Section 7 concludes the paper.

2 Background

This section provides the required background needed for properly introducing our reorganisation infrastructure. First, we briefly present the JaCaMo framework and its programming model (Section 2.1), and then we specifically focus on organisation programming in JaCaMo (Section 2.2) for introducing the founding concepts used in the proposed multi-agent oriented programming of reorganisation.

2.1 The JaCaMo Multi-Agent Programming Framework

JaCaMo [3] is a newborn multi-agent oriented conceptual framework and platform, which provides high-level first-class support for engineering Multi-Agent Systems (MAS) taking into account agent, environment, and organisation dimensions in synergy. The framework integrates three existing agent-based platforms – i.e. *Jason* [4], CArtAgO [10], and *M*OISE [8] – by defining in particular a semantic link among concepts of the different programming dimensions at the meta-model and programming levels, in order to obtain a uniform and consistent programming model aimed at simplifying the combination of

[1] Endogenous adaptation refers to an adaptation process realized by the agents participating to the organisation themselves.

those dimensions when programming a MAS [3]. A JaCaMo multi-agent system (i.e., a software system programmed in JaCaMo) is given by an agent organisation programmed in Moise, organising autonomous agents programmed in *Jason*, working in shared and distributed artifact-based environments programmed in CArtAgO.

When developing a MAS in JaCaMo environment programming is exploited to define the computational layer encapsulating functionalities and services that agents can use/explore/share at runtime [14]. Being based on the A&A meta-model [10], in CArtAgO and hence in JaCaMo, such software environments can be designed and *programmed* as a dynamic set of computational entities called *artifacts*, collected into workspaces, possibly *distributed* among various nodes of a network. In order to be used by the agents, each artifact provides a usage interface composed by a set of *operations* and *observable properties*. Operations correspond to the actions that the artifact makes it available to agents to interact with such a piece of the environment; observable properties define the observable state of the artifact, which is represented by a set of information items whose value (and value change) can be perceived by agents as percepts.
Agents obviously encapsulate the control and decision-making part of the application. Based on *Jason* an agent in the JaCaMo framework is programmed as an entity composed of a set of *beliefs*, representing agent's current state and knowledge about the environment in which it is situated, a set of *goals* (either private or bound to organisational goals), which correspond to tasks the agent has to perform/achieve, and a set of *plans* which are courses of *action*, either internal or external (mapped into artifacts' operation), triggered by *events*, and that agents can dynamically compose, instantiate and execute to achieve goals.
Organisation programming specifies and manages the overall governance strategy of the system. The following section describes how it is programmed based on Moise.

2.2 Organisation Programming in JaCaMo

In JaCaMo, an organisation program based on the Moise Organisation Modeling Language (OML) consists in an *organisation specification* (OS) stating the global structure, functioning and norms to achieve the global purpose for which the organisation is defined. The OS ($OS = \langle SS, FS, NS \rangle$) consists in two independent specifications –structural specification (SS) and functional specification (FS)– bound together by the normative specification (NS) (see [8] for further details):

- The SS ($SS = \langle R, \sqsubset, rg \rangle$) declares a set of roles R possibly connected by an inheritance relation $\sqsubset$ and a root group rg specification. A group specification can define subgroups, the set of roles contained in the group and that could be connected to each other by communication, authority and/or acquaintance links. Role-compatibility relations, agent-cardinalities for role adoption or group participation are used to constrain this setting during the life cycle of the organisation.

- The FS ($FS = \langle M, G, S \rangle$) is focused on the definition of plan-based strategies (or social schemes, S) for the achievement of collective inter-dependent goals G grouped into missions[2] M. The organisation state evolves according to the missions the agents commit to and the goal achievements.

[2] A mission represents a consistent grouping of collective or individual goals.

- The NS defines a set of norms ($norm = \langle id, c, \rho, dm, m \rangle$) that binds SS and FS together by the way of roles and missions. When the norm conditions c holds, any agent playing a role ρ has the deontic modality dm to commit to mission m.

Given the OS the set of agents A build what we call an *organisation entity* (OE) which current state can be described as $OE = \langle OS, A, GI, SI, O \rangle$. GI is the set of running group instances, i.e. groups of agents playing roles according to the specifications stated by the SS. SI is the social schemes instances SI, i.e. goals under achievement by the agents given the FS specification, following the current set of obligations O resulting from the activations of the norms stated in the NS. In order to help the agents to manage in a distributed way the frequent changes of the OE, a set of dedicated artifacts have been define (GroupBoard and SchemeBoard) to provide the agents with the proper tools to manage the OE. Such *organisational artifacts* are created for each GI or SI in the OE.

- Each group instance of GI is associated to a dedicated GroupBoard artifact that manages its current state: current set of agents in the group, playing the roles that are specified as part of that group type. This artifact provides the agents with operation for *adopting or leaving a role*, *adding or removing a social scheme* within this instance of group. Any agent that focuses on this artifact may get observable properties or different events pertaining to the *life cycle* of this group (e.g. which agent plays which role inside the group, number of roles that could be adopted, etc).

- Each social scheme instance of SI is managed by a SchemeBoard artifact. This artifact is connected to the corresponding GroupBoard managing the group instance in charge of its achievement. Thus all the agents participating to that group instance may be involved in the achievement of the goals managed by that scheme instance according to the obligations created from the activation of the norms stated in the NS. Using the operations of that artifact, agents can commit to a mission, leave it when finished, or can state that a given goal has been achieved. The SchemeBoard's observable properties provide the agents with the evolution of the coordinated goal-solving process (achieved goals, violated obligations, etc).

The Moise-based specifications of the organisation themselves are part of the information made observable by organisational artifacts to agents. This means that there is the potential for agents that understand the Moise OML to reason about the organisations in which they partake and therefore to change them at runtime. This allows for complex on-the-fly re-structuring of computational systems to be done at very high level.

3 Organisation Specification for Reorganisation

It is reasonable to think that adaptation of an application behaviour, in response to changes in the environment, requires a carefully selected and coordinated sequence of specific actions like monitoring, logging, reorganisation plan design, selection and implementation [7]. Being considered as an endogenous process, each of these actions will be carried out by one or several agents of the system [3]. As it is done for the coordination of the

[3] Depending of the application, agents taking part to the reorganisation may be dedicated agents or domain agents embedding particular reorganisation knowledge and skills

application, a dedicated organisation specification can be defined to coordinate and supervise the reorganisation process itself. Following the organisation programming proposed in the JaCaMo framework, we program this organisation using the *M*OISE OML, installing a kind of meta-level control of the reorganisation process itself. For that purpose, we have extended the organisation for reorganisation proposed in [7]. We describe it below along the structural, functional and normative dimensions.

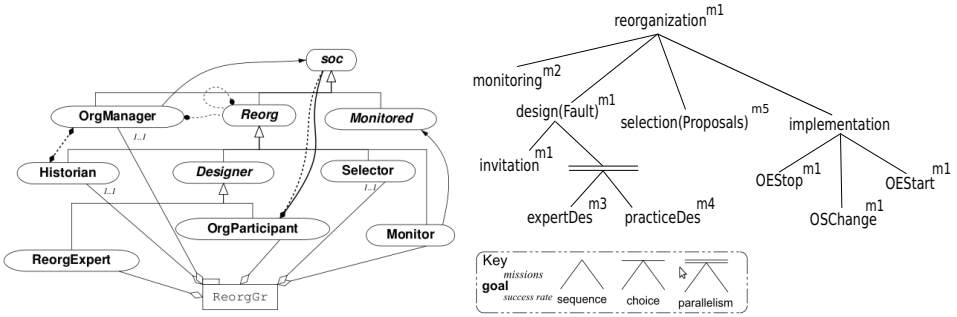


Figure 1: Graphical representations of the Reorganisation Organisation in terms of (i) Reorganisation Group and (ii) Reorganisation Social Scheme

3.1 Reorganisation Group

Figure 1-Reorganisation Group depicts the structural specification of the Reorganisation Group governing the reorganisation. We can immediately observe that, according to the names of the roles, the structure depicts the different coordination functionalities that are required to monitor, diagnose, design and install a new organisation from an existing one. All the roles are part of this group.

The OrgManager role is compatible with the Historian role. The Designer is specialized in a role that can be played by agents of the application domain (compatibility link between OrgParticipant with soc where soc is the root role of any structure of any organisation). It is also specialized into a ReorgExpert which is not compatible with any other role, forcing thus agents who adopt this role to have not adopted a role in another organisation. Let's note that the agents playing one of these roles may communicate with each other (reflexive communication link attached to the Reorg role and communication link between OrgManager and Reorg roles).

3.2 Reorganisation Social Scheme

The entire reorganisation process (see Figure 1 Reorganisation Social Scheme) consists in a monitoring phase followed by design, selection and finally implementation phases. In order to fully coordinate and implement the reorganisation process, we have extended the initial functioning presented in [7] by decomposing the *implementation* goal into *OEStop*, *OEChange* and *OEStart* subgoals, executed in sequence. The *OEStop* goal will be achieved when the *OE* will be stopped, i.e. targeted agents will stop pursuing collective goals or even leave their roles completely. The *OSChange* goal will be achieved when

all required modifications to the structural, functional or normative specifications will be done. Finally, the *OEStart* goal will be achieved when the new OE will be effectively created, informing targeted agents of their new assignments (what role(s) to play, what goal(s) to pursue etc).

3.3 Normative Specification for Reorganisation

The Normative Specification as depicted in Table 3.3 assigns the roles to missions with a set of norms consisting in obligations.

Condition	Deontic	Role	Mission
Application Specific	*obligation*	OrgManager	*m1*
Application Specific	*obligation*	Monitor	*m2*
Application Specific	*obligation*	ReorgExpert	*m3*
Application Specific	*obligation*	OrgParticipant	*m4*
Application Specific	*obligation*	Selector	*m5*

Table 1: Normative Specification of the reorganisation process. Conditions are not specified here since they are application dependent.

As stated by the *NS*, the OrgManager is the role that has to instantiate the organisation and supervise the whole process (in charge of mission *m1*). The Historian is in charge of keeping a list of all the changes that the organisation has gone through – a kind of useful information for the monitoring and design phases (e.g. role adoption, mission commitments, role creation, change in the cardinalities). The Designer contains the common properties for designers. The entire reorganisation process is expected to start when the agent(s) playing the Monitor role signal(s) a fault with the current organisation. Then, during the design stage, both agents playing Reorganisation Experts and Organisation Participants can come up with solutions for the identified problem. Agents playing the Selector separate role will then have to evaluate the best proposal out of those made by the designers. At the end, the agent in charge of the OrgManager will supervise the actual transition from the old organisation to the new one (cf. goal *implementation* within *m1*).

In the following section, we describe how we use the Artifact programming approach promoted by the JaCaMo platform to enrich the artifact-based organisation management infrastructure with a specific artifact to support the agents in the achievement of the *design* and *implementation* goals of the reorganisation social scheme.

4 Reorganisation Artifact

In the JaCaMo platform, organisation management is supported by a set of dedicated GroupBoard and SchemeBoard artifacts. Thus, the execution of the reorganisation process, as specified in the previous section, is de-facto managed by a GroupBoard for adopting/leaving roles defined in the *Reorganisation Group* and a SchemeBoard for committing to missions, achieving goals according to the specifications given in the *Reorganisation Social Scheme*. These artifacts provide the right operations for managing the execution of the generic and abstract reorganisation process. In order to complement this definition with an adapted *Design* and *Implementation* plan taking into account the real conditions in which the new organisation has to be implemented, we propose to enrich

this artifact-based organisation management infrastructure with a ReorgBoard artifact. It will help the agents to build and execute organisation implementation plans. This will constitute the base infrastructure that future applications can use in order to achieve adaptive behaviours.

After a global description of this artifact, we describe in detail the dynamic structure to handle the organisation implementation plan in the system and finaly give the description of the different operations that the agents are provided with.

4.1 ReorgBoard Artifact

The management of the organisation for reorganisation as defined in the previous section is set in practice in the JaCaMo framework with the creation of (*i*) a GroupBoard to manage the created ReorgGr instance and (*ii*) a SchemeBoard artifact to manage the execution of the reorganisation social scheme. The basic operations for changing the *OE* (e.g. adopting a role, committing to a mission, etc) are de facto provided to any member of a group instance (resp. scheme instance) via the corresponding operations offered by GroupBoard (resp. SchemeBoard).

One or several of these actions changing the *OS* and/or *OE* levels of the organisation must be coordinated: agents playing a role must leave that role at a certain time, agents committed to a mission also, so on and so forth. The sequence in which it has to be done is strongly dependent on the application, on the features of the current running organisation, etc. The achievement of the *design* and *implementation* goals of the reorganisation process (see Fig. 1) require thus a specific attention. In order to help the agents in this process, we define a new kind of artifact, called ReorgBoard (cf. Fig. 2). This artifact is created by the agents playing the OrgManager role, every time the *monitoring* is achieved, launching the achievement of the goal *design* for a particular *fault* (cf. Fig. 1).

Achieving the *design* goal consists in defining a coordinated set of reorganisation changes to be done both at the *OS* and/or *OE* levels in the organisation for each of the subgoal of the *implementation* goal. This coordinated set of changes builds what we call in the sequel an implementation plan ImplPlan. The ImplPlan is stored in the ReorgBoard artifact and is made accessible to the agents via a set of observable properties of this artifact. This way, the agents in charge of implementing the new organisation will get the implementation plan as a belief and will execute it. Complementary to this observable property, the ReorgBoard usage interface proposes two distinct sets of operations to help the agents in the achievement of those two goals (see Sec. 4.3).

In the following subsections, we will first describe the implementation plan structure and then describe the different operations that are provided by this artifact.

4.2 Implementation Plan

The different reorganisation changes that have to be executed to install a new organisation may be numerous and interdependent, addressing different group instances or social scheme instances. This is why we define a hierarchical structure (ImplPlan) taking into account precedence constraints and targeted GroupBoard or SchemeBoard artifacts on which they have to operate (changes on a group instance, on a scheme instance).

Each of the three subgoals *OEStop*, *OSChange*, *OEStart* of the *implementation* goal of the Reorganisation Social Scheme (cf. Fig. 1), will be refined by the agents into a specific

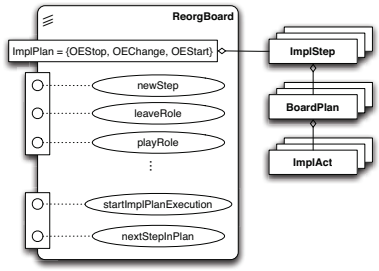


Figure 2: ReorgBoard Observable Properties (rectangles) and partial view of the operations (circles)

implementation plan ImplPlan according to the condition in which the new organisation has to be deployed. An implementation plan ImplPlan is a structured set of implementation actions ImplAct (see Fig. 2 for the implementation plan structure):

- An *OEStop* ImplPlan holds the coordinated set of actions to make the target agents stop their current work in the current organisation.
- An *OSChange* ImplPlan contains the coordinated set of actions that modify the OS which is governing the OE.
- An *OEStart* ImplPlan contains the coordinated set of actions that allow the agents playing the corresponding roles to re-instantiate a new OE according to the modified OS (e.g. telling target agents to play newly created roles, to commit to newly created missions, etc.)

An *implementation action* ImplAct defines an expression refering to an GroupBoard or SchemeBoard operation for changing the organisation. As mentioned, these different ImplAct actions that have to be executed to transit to a new organisation can be interdependent. To manage these dependencies, ImplPlan is decomposed in different implementation steps ImplStep which can be executed in sequence. An ImplStep can further be decomposed into board plans BoardPlans that group together implementation actions targeted at the same GroupBoard or SchemeBoard. For instance, implementation actions that are targeted to agents that play roles in the same group instance are grouped into the same BoardPlan corresponding to the GroupBoard that manages that group instance. Implementation actions that refer to changes in goals/missions are grouped in different BoardPlans according to the SchemeBoards that manage those goals/missions.

Implementation actions have different interpretations according to the organisation level they address:

- An OS level ImplAct contains an expression refering to an operation altering the OS (e.g. *addRoleObligation*, *changeRoleCardinality* etc), targeted to a GroupBoard or a SchemeBoard involved in the management of the running OE.
- An OE level ImplAct consists in an expression refering to an operation of a groupboard/schemeboard that a recipient agent has to perform (e.g. *leaveMission*, *playRole* etc). The execution of the action sends a request to the target agent to execute the change (cf. Sec. 5 for discussion on this notification mechanism).

OE level ImplAct actions are to be executed by autonomous agents. In order to control the way the action it refers to has to be executed by the agent, a "*strength*" modality is used to express the importance of the associated command and the delay with which it will have to be carried out:

- *regimentation*: immediate execution of the operation refered by the action. The agent it is addressed to is shunt: the action is directly executed in the corresponding GroupBoard or SchemeBoard.
- *obligation*: execution of the operation refered by the action as soon as the targeted agent has finished off any remaining goals from previous commitments.
- *permission*: execution of the operation refered by the action is permitted, but not required. It is mainly used when the new organisation does not depend on the successful completion of the operation.
- *interdiction*: interdiction to execute the operation refered in the action during the current reorganisation process.

4.3 Implementation & Design Operations

In order to build and manipulate the structure of the implementation plan, the ReorgBoard artifact provides the agents with a set of operations for the design and for the execution of such plans.

Design Operations. The operations for the design are used to create the structure of a ImplPlan, adding steps, boardplans. For instance, the operation `newStep(p, st)`: creates a new ImplStep *st* within the ImplPlan *p*, where $p \in \{OEStop, OSChange, OEStart\}$. This command binds the variable *st* with the step ID that can be used to reference the step in future commands. The table 2 shows the set of operations used for the addition of implementation actions. From the features of these actions, we can distinguish OS Level and OE Level ReorgBoard operations:

- An OS Level reorganisation design operation has the following syntax:

  $$operation(plan, step, target, type, object)$$

  It adds the implementation action "$operation(type, object)$" in the BoardPlan *target* belonging to ImplStep *step* of ImplPlan *plan*.
  For instance, the operation `addRole(p, st, A, G, ρ)` adds an implementation action for adding a new role ρ in the group definition `G`. It is included in the BoardPlan *A*, member of step `st`, part of the ImplPlan `p`.

- An OE Level reorganisation design operation has a syntax similar to the OS level design operation. The strength modality is added. Such an operation adds ImplAct in ImplBoard addressing OE Level changes.
  For instance, the operation `leaveMission(p, st, A, sm, ag, mi, sch)` adds to the BoardPlan *A* in step *st*, of the ImplPlan *p*, an implementation action that requests agent `ag` to stop working on the goals included in mission `mi` under execution in scheme instance of type `sch` managed by SchemeBoard instance `A` with the strength modality `sm`.

OS Level Operations	Description
addRole	add a role to a group specification
removeRole	remove a role from a group specification
addMission	add a mission to a scheme specification
removeMission	remove a mission from a scheme specification
addRoleObligation	add a normative assignment of a mission to a role
removeAllRoleObligations	clear a role of all its normative obligations
changeRoleCardinality	change the min..max interval of agents playing this role
changeSubgroupCardinality	change the min and max number for the instances of a subgroup
removeGroup	remove the given group specification from the organisation
removeScheme	remove the given scheme specification from the organisation
OE Level Operations	**Description**
leaveRole	target agent must leave the specified role
playRole	target agent must start playing the specified role
leaveMission	target agent must discontinue pursuing the goals in given mission
commitMission	target agent must commit to solving goals in given mission

Table 2: OE and OS Levels Reorganisation Design Operations

Implementation Operations. During the achievement of the *implementation* goal the ReorgBoard artifact acts as a coordinating entity. To this aim, it provides the following operations to the agent committed to the achievement of the implementation goals:

- `startImplPlanExecution(p)`: triggers the execution of the ImplPlan p, corresponding to the execution in sequence of the different *ImplStep* defining the plan.
- `nextStepInPlan(p)`: starts the execution of the next step within the *ImplPlan* p, once all the commands in the current step have completed. This last operation allows the OrgManager working with the ReorgBoard to start a new step once it has perceived the notifications that all the actions dictated by the previous commands have been performed.

Having described the internal structure of the ReorgBoard and the usage interface it exhibits, the next section will detail the interactions that exist between the ReorgBoard, the agents and the OrgArtifact instances to which it connects during the reorganisation process.

5 Multi-Agent based Reorganisation

In the previous section we have presented the ReorgBoard as an instrument that agents can use to achieve the *design* and *implementation* goals of the reorganisation process governed by the ReorgGr reorganisation group and the reorganisation social scheme described in Sec. 3. In this section, we bring together the pieces of the jigsaw and describe the use of this artifact in the global context of a reorganisation. We describe the different interactions that take place between the ReorgBoard, the agents involved in the reorganisation and GroupBoard or SchemeBoard instances that are impacted by the reorganisation (see. Fig. 3). We focus here on the design and implementation phases that are the core of the paper [4].

[4]For page limitation reasons, we cannot describe the full implementation and application. Interested readers may refer to `http://jacamo.sourceforge.net/?page_id=87` or [12]

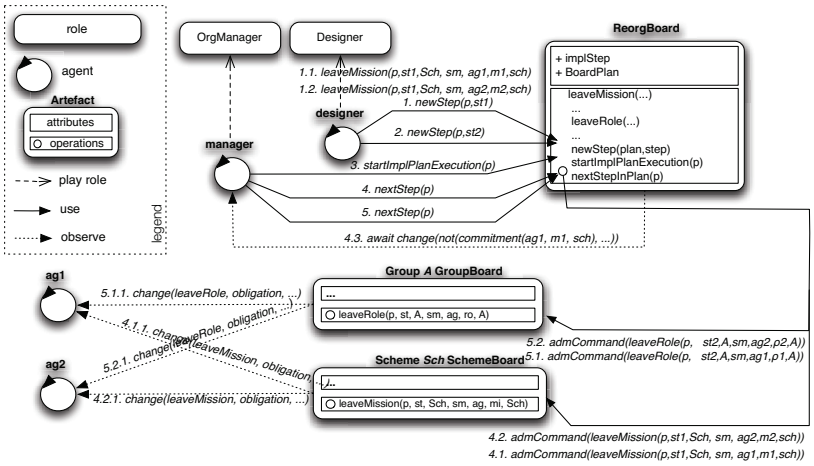


Figure 3: Collaboration diagram of the reorganisation process.

5.1 Reorganisation Design

In order to achieve the design goal, the agents playing the OrgManager role and the other agents playing the Designer role use the ReorgBoard's usage interface to construct the content of each implementation plan (steps, boardplans and implementation actions) by using different reorganisation design operations (cf. Table 2).

As an example, let's consider dynamic role (re)allocation in rescue teams. In such a use case, a *role shift* implies a sequence of actions ranging from the leaving of current roles (*OEStop*), to possible structural modifications (*OSChange* - e.g. change in minimum or maximum cardinalities of agents playing a given role) and adoption of the new roles (*OEStart*). Note that the knowledge and strategy (e.g. negotiation or machine learning) used to come up with the proposed role shift is particular to each Designer agent. The ReorgBoard and its reorganisation design operations act as the *mechanism* used by agents to implement the required changes.

To show the exact use of the artifact in the context of reorganisation, let's focus on the *OEStop* subgoal of a hypothetical role shift, where the agent in charge of the design decides that, in the current ImplPlan p, the agents (ag_1 and ag_2) belonging to group A have to leave their roles (ρ_1 and ρ_2).

Since, before leaving their roles, the agents must first leave their missions, the agent playing the Designer role defines two ReorgSteps (`newStep(p, st`$_1$`)` and `newStep(p, st`$_2$`)`), which will be executed in sequence: the first one contains an ImplAct for the given agents to stop their work on the goals contained within the missions they are responsible for, whereas the second one comprises actions that tell those two agents to leave their current role (see. 1 and 2 on Fig. 3).

The commands to leave the missions are included in the ImplStep referenced by st_1. Both actions are grouped under the BoardPlan that concerns the SchemeBoard instance (*Sch*) managing those missions. To do this, the designer agent invokes the following operations ((see. 1.1 and 1.2 on Fig. 3): `leaveMission(`p`, `st_1`, Sch, sm, `ag_1`, `m_1`, sch)` and `leaveMission(`p`, `st_1`, Sch, sm, `ag_2`, `m_2`, sch)`, which bare the same semantics as those described in section 4.3.

The commands to leave the roles are contained in the second ImplStep. The agents ag_1 and ag_2 being members of the same group, the actions are put together under the same BoardPlan that will be sent to the GroupBoard instance managing group *A*. The designer agent achieves this by calling `leaveRole(`p`, `st_2`, A, sm, `ag_1`, `ρ_1`, A)` and `leaveRole(`p`, `st_2`, A, sm, `ag_2`, `ρ_2`, A)` on the ReorgBoard artifact. With this, the design process for the considered *OEStop* implementation subgoal is completed [5]

5.2 Reorganisation Implementation

The achievement of the implementation goal consists in the direct interaction of the agent playing the OrgManager role with the ReorgBoard artifact. Using the different implementation operations, it ensures the proper and ordered execution of the implementation actions contained in each of the implementation plans produced during the achievement of the design goal.

Pursuing with our previous example of the *OEStop* implementation plan *p*, the agent OrgManager uses the `startImplPlanExecution(p)` operation to start the first step (st_1) of the ImplPlan. When handling the step, the ReorgBoard executes the list of BoardPlans, in the case of st_1, those directed to the SchemeBoard instance *Sch* managing missions m_1 and m_2. The ReorgBoard links itself to the receiving OrgArtifact instance in order to send all of the implementation actions included in the BoardPlan.[6]
When an OrgArtifact instance receives a ImplAct, it looks at the level this action is situated (OE Level or OS Level).

- In case of an OS level action, the changes contained within the action directly alter the internal specifications of the organisational artifact instance.
- In case of an OE level action, such as in our example, a first notification is done between the ReorgBoard artifact to the GroupBoard/SchemeBoard concerned by the change. A second notification is then done by this latter artifact, informing the targeted agent (e.g. ag_1, ag_2) of the pending change stated by this command. As part of the step st_1, agent ag_1 would, for instance, receive an expression like: $change(leaveMission, obligation, ag_1, m_1, sch)$ (see. 4.1, 4.1.1 and 4.2, 4.2.1 and 5.1, 5.1.1 and 5.2, 5.2.1 on Fig. 3).

Meanwhile, for implementation actions with a strength attribute lower than *regimentation* (like in the example above), the ReorgBoard uses the same observable events mechanism to inform the OrgManager of the perceived organisational statements he has to wait for before he can move on to the next step.

In our example the agent playing the OrgManager will receive, among others, a notification from the ReorgBoard of the form: $await\_change(not(commitment(ag_1, m_1, sch), ...))$, meaning the agent will have to wait until she perceives the organisational fact that agent ag_1 actually quit handling the goals included in mission m_1 from the scheme *sch*, before proceeding with other actions (i.e. calling the `nextStepInPlan(p)` operation on the ReorgBoard). The agent will be noticed of that fact thanks to the observable properties of the SchemeBoard Sch.

[5] For clarity reasons, we didn't add the call to these operations on the Fig. 3).

[6] Both GroupBoard and SchemeBoard artifacts provide an operation which allow for the handling of internal administration tasks. This operation, called `admCommand`, is exploited by the ReorgBoard to link to the required organisational artifact when sending a ImplAction.

When all steps in a plan have been executed, the agent playing the OrgManager role is again informed via an observable event sent out by the ReorgBoard that it is possible to proceed to the next plan. When the last plan has finished executing, the reorganisation process is complete and the agent playing the OrgManager role can discard the ReorgBoard artifact.

6 Related Works

There is a huge literature on reorganisation in multiagent systems. Some use an exogenous reorganisation process where the user or a dedicated MAS itself reorganise the whole system (e.g. [13, 6] or [5]). Others [9, 11], like our proposal or the one used as a starting point, use an endogenous approach where the agent themselves modify the organisation. However, we make clear and explicit the organisation controlling the reorganisation process itself. This is a clear difference with the other approaches promoting an endogenous approach where the reorganisation process is hard coded in the agents themselves.

An approach similar to ours can be found in the work by Alberola et al. [1]. The Reorganisation Facilitator Service they propose has a functionality and purpose which are close to those of the ReorgBoard. However, even if the approach in [1] allows for the finding of the minimal-cost transition from an initial organisation instance to a newly desired one (which can actually be viewed as a particular strategy), the work does not describe an explicit coordination and supervision mechanism for the implementation of the required transition operations.

Using the reorganisation artifact approach, we go a step further in the sense that we have proposed a set of tools that can be used to support and engineer this process. This is a strong added value with respect to the one proposed by Hubner et al. [7]. Thanks to this artifacted process, we have been able to contribute to different ways of changing the organisation as in [2] and to address the practical setting of organisation changes: when and how to stop, when and how to implement the changes, when and how to restart the organisation process.

7 Conclusions

This work has presented a clear mechanism that is used to engineer and support the process of reorganisation in multi-agent organisations. Its realization was made possible by means of the JaCaMo platform and it represents an extension of previous work on the modeling of the reorganisation process started in [7]. Specifically, the business logic of the developed ReorgBoard artifact helps extend the functional specification of the "organisation for reorganisation" with the explicit stages of the implementation phase: *stopping* the affected part of the organisation, *applying changes* to the organisational specifications themselves and *starting* the new instance of the organisation. The ReorgBoard offers an internal structuring and a set of operations to instrument both the *design* and the coordinated *implementation* of the reorganisation process.

The benefit of this extended model for reorganisation is that each involved step (*monitoring*, *design* and *implementation*) is clearly defined and controllable. By means of the artifact-based infrastructure offering the required coordination, agents can be seen to employ whichever application-specific adaptation strategies or heuristics, thus achieving a

separation of concerns between the underlying reorganisation mechanism and the considered adaptation policy.

8 Acknowledgments

This work has been supported by French Rhône-Alpes Région CMIRA and ERASMUS/SOCRATES funds.

References

[1] J. M. Alberola. A cost-oriented reorganization reasoning for multiagent systems organization transitions. In L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors, *AAMAS*, pages 1349–1350. IFAAMAS, 2011.

[2] H. Aldewereld, F. Dignum, V. Dignum, and L. Penserini. A formal specification for organizational adaptation. In M. P. Gleizes and J. J. Gómez-Sanz, editors, *AOSE*, volume 6038 of *Lecture Notes in Computer Science*, pages 18–31. Springer, 2009.

[3] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, (0):–, 2011.

[4] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using **Jason***. Wiley Series in Agent Technology. John Wiley & Sons, 2007.

[5] E. Bou, M. López-Sánchez, J. A. Rodríguez-Aguilar, and J. S. Sichman. Adapting autonomic electronic institutions to heterogeneous agent societies. In G. A. Vouros, A. Artikis, K. Stathis, and J. V. Pitt, editors, *AAMAS-OAMAS*, volume 5368 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2008.

[6] B. Horling, B. Benyo, and V. R. Lesser. Using self-diagnosis to adapt organizational structures. In *Agents*, pages 529–536, 2001.

[7] J. Hübner, J. Sichman, and O. Boissier. Using the moise+ for a cooperative framework of mas reorganisation. In *SBIA*, volume 2004, pages 506–515. Springer Verlag, 2004.

[8] J. F. Hübner, J. S. Sichman, and O. Boissier. Developing Organised Multi-Agent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels. *Agent-Oriented Software Engineering*, 1(3/4):370–395, 2007.

[9] S. Kamboj. Analyzing the tradeoffs between breakup and cloning in the context of organizational self-design. In C. Sierra, C. Castelfranchi, K. S. Decker, and J. S. Sichman, editors, *AAMAS (2)*, pages 829–836. IFAAMAS, 2009.

[10] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in CArtAgO. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2*. Springer, 2009.

[11] M. Sims, C. V. Goldman, and V. R. Lesser. Self-organization through bottom-up coalition formation. In *AAMAS*, pages 867–874. ACM, 2003.

[12] A. Sorici, O. Boissier, G. Picard, and A. Santi. Exploiting the jacamo framework for realising an adaptive room governance application. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, SPLASH '11 Workshops, pages 239–242, New York, NY, USA, 2011. ACM.

[13] M. Tambe, D. V. Pynadath, and N. Chauvat. Building dynamic agent organizations in cyberspace. *IEEE Internet Computing*, 4(2):65–73, 2000.

[14] D. Weyns, A. Omicini, and J. J. Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, Feb. 2007.

Providing Agents With Norm Reasoning Services

N. Criado[1], J.M. Such[1], and V. Botti[1]
[1] *DSIC, Universitat Politècnica de Valencia (Spain),*
{ncriado,jsuch,vbotti}@dsic.upv.es

Abstract

Norms are used in open Multi-Agent Systems as a formal specification of deontic statements aimed at regulating the actions of agents and the interactions among them. In this paper, we propose a set of services facilitating the development of both non-normative and normative agents for norm-governed MAS. Specifically, we propose to provide agents with norm reasoning services. These services will help agent designers/developers to programme agents that consider norm reasoning without having to implement the needed mechanisms to reason about norms by themselves. Moreover, these services have been integrated into the Magentix2 agent platform.

Keywords: norms, services, agents.

1 Introduction

The main feature of open Multi-Agent Systems (MAS) is that they are populated by heterogeneous agents, which can enter or leave the system dynamically. These heterogeneous agents may have been designed independently, according to different goals, and no assumption about their behaviours can be made [1]. To ensure social order and avoid potential conflicts, norms are used in open MAS as a formal specification of deontic statements aimed at regulating the actions of agents and the interactions among them [17].

A great amount of work has been done to use norms in open MAS [4]. For example, there are works aimed at: allowing the system designers to define and represent the norms that regulate a concrete application [15], controlling norms inside specific agent platforms [5, 11, 12], proposing norm-autonomous agent architectures [2] that are endowed with norm reasoning capabilities. In this paper, we try to facilitate both non-normative and normative agents to participate inside norm-governed MAS. Specifically, we propose to provide agents with *norm reasoning services*. These services range from a simple services that inform about the norms that are in force at a given moment, given norm-autonomous agents the chance to decide which norms are relevant to them and which ones they want to obey; to more elaborated services that allow agents to known which are the the normative goals that they must pursue, allowing non-normative agents to behave

accordingly to norms. Thereby, agent designers may refrain from programming agents with norm reasoning capabilities. These services have been integrated into the Magentix2[1] agent platform (AP). The Magentix2 AP allows the management of open MAS in a secure and optimized way. Its main objective is to bring agent technology to real domains: business, industry, e-commerce, among others.

This paper is structured as follows: Section 2 briefly describes Magentix2; Section 3 describes the norm reasoning services; Section 4 contains a brief discussion and future work.

2 The Magentix2 Agent Platform

Magentix2 is an agent platform for open MAS in which heterogeneous agents interact and organize themselves into Virtual Organizations (VOs) [10]. Magentix2 provides support for VOs at three levels:

- *Organization level*. Magentix2 provides access to the organizational infrastructure through the *Organization Management System* (OMS) [7], which is in charge of the management of VOs, taking control of their underlying structure, the roles played by agents, and the norms that govern the VO.
- *Interaction level*. Magentix2 provides support to: *agent communication*, supporting asynchronous reliable message exchanges and facilitating the interoperability between heterogeneous entities; *agent conversations* [9], which are automated Interaction Protocols; *tracing service support* [3], which allows agents in a MAS to share information in an indirect way by means of trace events; and, finally, Magentix2 incorporates a *security module* [18] that provides features regarding security, privacy, openness and interoperability.
- *Agent level*. Magentix2 provides native support for executing Jason agents and conversational agents that carry out simultaneous conversations.

Norms define what is considered as permitted, forbidden or obligatory in an abstract way. However, norm compliance must be controlled considering the actions and messages exchanged among agents at the interaction level. Magentix2 fills the gap between the organizational level, at which norms are registered by the OMS; and the interaction level, at which actions and communications, through a *norm-enforcing architecture* [5]. In this paper, we aim at filling the gap between the organizational level, at which norms are defined; and the agent level, at which norms must be considered before taking action. Specifically, we define Norm Reasoning Services (NRSs) that avoid agents from being endowed with norm reasoning capabilities. Next, the tracing service and the storage of norms, provided by the OMS, are described.

2.1 Tracing Service

To facilitate indirect communication (i.e., indirect ways of interaction and coordination), Magentix2 provides a Tracing Service [3]. These tracing facilities are provided by a set of components named Trace Manager (TM).

[1] http://magentix2.gti-ia.upv.es/

A trace event or *event* is a piece of data representing an action, message exchange or situation that has taken place during the execution of an agent or any other component of the MAS. *Generic* events, which represent application independent information, are *instrumented* within the code of the platform. *Application* events are domain dependent information. An *event* is defined as a tuple $\langle Type, Time, Origin, Data\rangle$, where: $Type$ is a constant that represents the nature of the information represented by the event; $Time$ is a numeric value that indicates the global time at which the event is generated; $Origin$ is a constant that identifies the tracing entity that generates the event; and $Data = \psi_1 \wedge ... \wedge \psi_n$ is a conjunction of possibly negated first-order grounded atomic formulae that contains extra attached data required for interpreting the event.

Any entity is provided with boxes for receiving or delivering events (E_{In} and E_{out}). The tracing service is based on the publish/subscribe software pattern, which allows subscribers to filter events attending to some attributes (content-based filtering), so that agents only receive the information in which they are interested and only requested information is transmitted. Entities that want to receive certain trace events request the subscription to these events by sending to the TM a *subscription* event that contains the template of those events they are interested in. A *template* is a tuple $\langle Type, Origin, Data\rangle$ that contains the filtering specified criteria for events, where: $Type$ is a constant that represents the nature of the information represented by the event; $Origin$ is a constant that identifies the entity that generates the event; and $Data = \psi_1 \wedge ... \wedge \psi_n$ is a conjunction of possibly negated first-order atomic formulae that may contain free variables. Let us consider the standard notion of substitution as a finite and possibly empty set of pairs X/y where X is a variable and y is a term. The application of a substitution on a template $\sigma(\langle Type, Origin, Data\rangle)$ is $\langle Type, Origin, \sigma(Data)\rangle$ since $Type$ and $Origin$ take constant values.

According to the definitions of events and templates the *matching* and $unifiaction$ relationship between events and templates are defined as follows:

Definition 1 (Matching Function)**.** *Given an event* $e = \langle Type, Time, Origin, Data\rangle$ *and a template* $t = \langle Type', Origin', Data'\rangle$*, their matching is a boolean function defined as follows:*

$$matching(e,t) = \begin{cases} true & \text{if } (Type = Type') \wedge (\forall \psi_i \in Data' : \psi_i \in Data) \\ & ((Origin = Origin') \vee (Origin' \text{ is undefined})) \\ false & \text{otherwise} \end{cases}$$

Definition 2 (Unification Function)**.** *Given an event* e *and a template* t*, their* $unification$ *is a boolean function defined as follows:*

$$unification(e,t) = \begin{cases} true & \text{if exists a substitution of variables } \sigma \text{ such that} \\ & matching(e, \sigma(t)) \text{ is true} \\ false & \text{otherwise} \end{cases}$$

2.2 Organization Management System (OMS)

The Organization Management System (OMS) [7] is responsible for the management of VOs and their constituent entities. The OMS provides a set of services: **structural services**, which comprise services for adding/deleting norms (*registerNorm* and *deregisterNorm* services allow entities to modify the norms that are in *force* or applicable within a VO), and for adding/deleting roles and groups; **informative services**, that provide information of the current state of the organization; and **dynamic services**, which allow agents to enact/leave roles inside VOs (*acquireRole* and *leaveRole* services). Moreover, agents can be forced to leave a specific role (*expulse* service). When the OMS provides any of these services successfully, then it generates an event for informing about the changes produced in the VO.

According to the normative definitions provided in [15], in Magentix2 a distinction among *norms* and *instances* is made. A *norm* is defined as a tuple $\langle id, D, T, A, E, C, S, R\rangle$, where: id is the norm identifier; $D \in \{\mathcal{F}, \mathcal{O}\}$ is the deontic modality of the norm, $\mathcal{F}$ represents prohibition and $\mathcal{O}$ represents obligation; T is the target of the norm, the role to which the norm is addressed; A is the norm activation condition, it defines under which circumstances the norm is active and must be instantiated; E is the norm expiration condition that determines when the norm expires and no longer affects agents; C is the norm condition that represents the action or state of affairs that is forbidden or obliged; S and R describe the sanctions and rewards that will be carried out in case of norm violation or fulfilment, respectively. As previously argued, the norm-enforcing architecture builds on the event tracing approach to monitoring. Thus, the conditions A, E, C, S and R are expressed in terms of event templates.

Over the course of this paper, we will use an example to illustrate and motivate the need of the NRSs that we propose. Specifically, this example consists on an *auction house* that has been implemented as a VO in Magentix2. Heterogeneous agents can enter or leave the *auction house*. To control the system and avoid the potential excesses of malicious agents, this *auction house* is regulated by a set of norms that define which are the rights and responsibilities of each role in terms of obligations, prohibitions and permissions. Let us suppose the existence of norm $n1$ that forbids to bid for an item once the auction corresponding to this item has been closed:

$$\langle n_1, \mathcal{F}, buyer, \langle auctionEnd, -, item(I)\rangle, \langle auctionStart, -, item(I)\rangle, \langle bid, -, item(I)\rangle, -, -\rangle$$

According to norm n_1 once the $\langle auctionEnd, -, item(I)\rangle$ event is sent, any agent that enacts the *buyer* role is forbidden to bid for the item I. This prohibition expires when the item I is auctioned again (i.e., when the $\langle auctionStart, -, item(I)\rangle$ event is sent).

When the activation condition of a norm holds; i.e., the activation event is detected, then it becomes active and several norm *instances* (or instances for short) are created, according to the possible groundings of the activation condition. Given a norm $\langle id, D, T, A, E, C, S, R\rangle$ and a perceived event e, an instance is the tuple $\langle id', D', T', E', C', S', R'\rangle$, where: there is a substitution σ such that

$matching(e, \sigma(A))$ is true (i.e., the norm is active); $C' = \sigma(C)$, $E' = \sigma(E)$, $S' = \sigma(S)$, and $R' = \sigma(R)$; and $id' = id, D' = D$ and $T' = T$.

In our example, let us suppose that the event $\langle auctionEnd, -, item(car)\rangle$ is sent. Thus, norm n_1 will be instantiated as follows:

$$\langle n_1, \mathcal{F}, buyer, \langle auctionEnd, -, item(car)\rangle, \langle auctionStart, -, item(car)\rangle, \langle bid, -, item(car)\rangle, -, -\rangle$$

Definition 3 (Instantiation Function)**.** *Given an event $e = \langle Type, Time, Origin, Data\rangle$ and a norm $n = \langle id, D, T, A, E, C, S, R\rangle$, instantiation is a function that instantiates norm n as follows:*

$$instantiation(e, n) = \langle id', D', T', E', C', S', R'\rangle$$

where: there is a substitution σ such that $matching(e, \sigma(A))$ is true; $C' = \sigma(C)$, $E' = \sigma(E)$, $S' = \sigma(S)$, and $R' = \sigma(R)$; $id' = id, D' = D$ and $T' = T$.

The operational semantics of norms and instances (i.e., how they are created, deleted, fulfilled and violated) is described in [5].

3 Norm Reasoning Services

Norm Reasoning Services (NRSs) have been designed with the aim of allowing both non-normative and normative agents to interact within norm-governed VOs. To provide their functionality, NRSs require the existence of a Normative Monitor (NM) that keeps track of VOs. Both the NM and the NRSs are explained below.

3.1 Normative Monitor (NM)

The NM is responsible for monitoring VOs and providing NRSs with the information that they require. Specifically, it maintains three lists that contain the set of norms (N), instances (I) and the roles that are enacted by agents (RE) at a given moment. Moreover, it records information that can be used to judge past actions. Specifically, it maintains two log files: the log named $LogI$ contains information about the activation and expiration of instances, and the $LogRE$ contains information about which roles agents are playing (or played) at a given moment [2]. To maintain these lists and logs, the NM subscribes to the events sent by the OMS that related to the creation and deletion of norms (i.e., $registerNorm$ and $dregisterNorm$ events) and the enactment of roles (i.e., $acquireRole$, $leaveRole$ and $expel$ events). Algorithm 1 illustrates the pseudocode of the control loop performed by the NM. Each time the NM receives an event (e), it handles the event according to the event type. The NM carries out a process that can be divided into three differentiated tasks: norm management, instance management and role enactment management.

[2] The lists and log files may be implemented as blackboards, a database that can be accessed by the NRSs, or simply as files that are shared with the NRSs.

Algorithm 1 Normative Monitor Control Loop

Require: Norm list N
Require: Instance list I
Require: Instance log $LogI$
Require: Role Enactment list RE
Require: Role Enactment log $LogRE$
1: Add $\langle subscription, NM, \langle registerNorm, OMS, - \rangle\rangle$ to E_{Out}
//where NM stands for Norm Monitor
2: Add $\langle subscription, NM, \langle deregisterNorm, OMS, - \rangle\rangle$ to E_{Out}
3: Add $\langle subscription, NM, \langle acquireRole, OMS, - \rangle\rangle$ to E_{Out}
4: Add $\langle subscription, NM, \langle leaveRole, OMS, - \rangle\rangle$ to E_{Out}
5: Add $\langle subscription, NM, \langle expel, OMS, - \rangle\rangle$ to E_{Out}
6: **while** E_{In} is not empty **do**
7: Retrieve e from E_{In} // $e = \langle Type, Time, Origin, Data \rangle$
//...
// Norm Management
//...
// Instance Management
//...
// Role Enactment Management
72: **end while**

3.1.1 Norm Management

Algorithm 2 contains the portion of pseudocode corresponding to the norm management process. Any time the NM receives an event informing about the creation of a new norm, then it adds this norm into its norm list and subscribes to the event that activates the norm. When a norm is deregistered, then the NM removes it from its norm list. Moreover, it removes all instances that have been created out of this norm. For each one of these deleted instances, the NM registers the expiration of the instance in the corresponding log and unsubscribes from the expiration event.

Algorithm 2 Norm Management

8: **if** $Type = registerNorm$ **then** // $Data = \langle id, D, T, A, E, C, S, R \rangle$
9: Add $Data$ to N
10: Add $\langle subscription, NM, A \rangle$ to E_{Out}
11: **end if**
12: **if** $Type = deregisterNorm$ **and** $Data$ **in** N **then** // $Data = \langle id, D, T, A, E, C, S, R \rangle$
13: Remove $Data$ from N
14: Add $\langle unsubscription, NM, A \rangle$ to E_{Out}
15: **for all** i **in** I **do** // $i = \langle id', D', T', E', C', S', R' \rangle$
16: **if** $id' = id$ **then**
17: Remove i from I
18: **for all** (i', t_{In}, t_{Out}) **in** $LogI$ **do**
19: **if** $i' = i$ **and** $t_{Out} = null$ **then**
20: Remove (i', t_{In}, t_{Out}) from $LogI$
21: Add $(i, t_{In}, Time)$ to $LogI$
22: **end if**
23: **end for**
24: Add $\langle unsubscription, NM, E' \rangle$ to E_{Out}
25: **end if**
26: **end for**
27: **end if**

3.1.2 Instance Management

According to Algorithm 3, when the activation event of a norm is received, then the NM instantiates the norm and adds it to the instance list. At this moment, the NM registers the creation of the instance in the corresponding log and subscribes to the expiration event. Similarly, when the NM receives the expiration event of any instance, then it removes it from the instance list, unsubscribes from the expiration event and registers the expiration of the instance in the log file.

Algorithm 3 Instance Management

```
28: for all n in N do // n = ⟨id, D, T, A, E, C, S, R⟩
29:   if unification(e, A) then // the norm is active
30:     i = instantiation(e, n) // i = ⟨id′, D′, T′, E′, C′, S′, R′⟩ is an instance
31:     if i not in I then
32:       Add i to I
33:       Add (i, Time, null) to ILog
34:       Add ⟨subscription, NM, E′⟩ to E_Out
35:     end if
36:   end if
37: end for
38: for all i in I do // i = ⟨id′, D′, T′, E′, C′, S′, R′⟩
39:   if unification(e, E′) then
40:     Remove i from I
41:     for all (i′, t_In, t_Out) in LogI do
42:       if i′ = i and t_Out = null then
43:         Remove (i′, t_In, t_Out) from LogI
44:         Add (i, t_In, Time) to LogI
45:       end if
46:     end for
47:     Add ⟨unsubscription, NM, E′⟩ to E_Out
48:   end if
49: end for
```

3.1.3 Role Enactment Management

Algorithm 4 illustrates the pseudocode corresponding to the role enactment management process. Specifically, if the OMS informs that an agent ($AgentID$) has acquired a new role ($RoleID$), then the NM updates the role enactment list and the log file. When the OMS informs that an agent is not longer playing a role, then both the role enactment list and log are updated.

3.2 Norm Reasoning Services

Magentix2 allows heterogeneous agents to interact via FIPA-ACL messages. Similarly, NRSs are provided with mail boxes for receiving or sending FIPA-ACL messages (M_{In} and M_{Out}). For the purpose of this paper we will define a message as a tuple $\langle Type, Sender, Receiver, Content\rangle$; where $Type$ contains the message performative, $Sender$ contains the ID of the entity that has delivered the message, $Receiver$ contains the identifier of the entity to which the message is addressed, and $Content$ contains the content of the message. Agents access NRSs by sending a request message to the corresponding service. The result of the service is sent back to the agent through an inform message[3].

[3]Note that this is a simplification of the FIPA Request Interaction Protocol.

Algorithm 4 Role Enactment Management

```
50: if Type = acquireRole then // Data is a pair (AgentID, RoleID)
51:   Add Data to RE
52:   Add (Data, Time, null) to LogRE
53: end if
54: if Type = leaveRole or Type = expel then // Data is a pair (AgentID, RoleID)
55:   Remove Data from RE
56:   for all ((agentID, roleID), t_In, t_Out) in LogRE do
57:     if agentID = AgentID and roleID = RoleID and t_Out = null then
58:       Remove ((agentID, roleID), t_In, t_Out) from LogRE
59:       Add ((agentID, roleID), t_In, Time) to LogRE
60:     end if
61:   end for
62: end if
```

3.2.1 Norm Information Service (NIS)

Suppose that an *auditor* agent analyses the performance of the *auction house* and that it aims at ascertaining the influence of the norms on the transactions that take place in the *auction house*. Thus, it needs to gather evidences from both the actions and the norms. Specifically, it needs to know not only which norms are in force, but also the instances that are active at a given moment.

The Norm Information Service (NIS) is in charge of proving information about the norms and the instances that have been created out of these norms. Algorithm 5 contains the pseudocode corresponding to this functionality.

Algorithm 5 Norm Information Service Control Loop

```
1: while M_In is not empty do
2:   Retrieve m from M_In // m = ⟨Type, Sender, Receiver, Content⟩
3:   if Type = request and Receiver = NIS and Content = norm then
4:     Add ⟨inform, NIS, Sender, norm(N)⟩ to M_out
5:   end if
6:   if Type = request and Receiver = NIS and Content = instance then
7:     Add ⟨inform, NIS, Sender, instance(I)⟩ to M_out
8:   end if
9: end while
```

3.2.2 Relevance Information Service (RIS)

Suppose that a *buyer* agent (b_1) enters the *auction house*. Agent b_1 wants to know which its expected behaviour is and which the expected behaviours of its interaction partners are.

The Relevance Information Service (RIS) is in charge of proving information about the instances that are relevant to a target agent, which is specified in the service request. Algorithm 6 contains the pseudocode corresponding to this functionality. When the service is requested, then the RIS obtains the set of roles that are currently played by the target agent (*RoleList*). After this, the RIS searches the instance list for the instances that are addressed to the roles currently played by the target agent (*InstanceList*).

In our example, b_1 asks RIS about its relevant instances by sending the following message:

$$\langle request, b_1, RIS, relevance(b_1)\rangle$$

Algorithm 6 Relevance Information Service Control Loop

```
1: while M_In is not empty do
2:   Retrieve m from M_In // m = ⟨Type, Sender, Receiver, Content⟩
3:   if Type = request and Receiver = RIS and Content = relevance(AgentID) then
4:     InstanceList = {}
5:     RoleList = {}
6:     for all (agentID, roleID) in RE do
7:       if AgentID = agentID then
8:         Add roleID to RoleList
9:       end if
10:    end for
11:    for all i in I do // i = ⟨id', D', T', E', C', S', R'⟩
12:      if T' in RoleList then
13:        Add i to InstanceList
14:      end if
15:    end for
16:    Add ⟨inform, RIS, Sender, relevance(InstanceList)⟩ to M_out
17:  end if
18: end while
```

Then the RIS checks the instances that are relevant to b_1. Since b_1 is affected by the instance that has been created out of n_1 (as detailed in Section 2.2), this instance is relevant to b_1, so that the RIS sends the following message:

$$\langle inform, RIS, b_1, relevance(\{\langle n_1, \mathcal{F}, buyer, \langle auctionEnd, -, item(car)\rangle, \langle auctionStart, -, item(car)\rangle, \langle bid, -, item(car)\rangle, -, -\rangle\})\rangle$$

3.2.3 Norm Advice Service (NAS)

Suppose that b_1 does not understand the language used for representing instances and, as a consequence, it does not have capabilities for reasoning about them. However, it would like to respect the norms to avoid sanctions and maintain a good reputation.

The Norm Advice Service (NAS) determines which goals ($NormativeGoals$) must be pursued according to norms. Agents that request this service provide their intrinsic desires, which are a set of literals formed by event templates (or the negation of event templates). This set represents the actions or states of affairs pursued (or avoided) by the agent. With this information, the NAS calculates which goals must be pursued according to norms and to what extent these goals are advisable for the agent. Algorithm 7 contains the pseudocode corresponding to the NAS. Once the service receives a request, then the NAS calculates the set of instances that are relevant to the petitioner agent. For each relevant instance, the NAS computes how much the instance is advisable for the agent. This advisability degree is calculated by a function as follows:

Definition 4 (Advisability Function)**.** *Given an instance* (i) *and a set of goals* ($Goals$) *the advisability of this instance is calculated as follows:*

$$advisability(i, Goals) = \frac{f_{Interest}(i, Goals) + f_{Expectations}(i, Goals)}{2}$$

The advisability function is defined as the average among the values calculated by two functions:

- $f_{Interest}$. This function considers the influence of norm compliance on the agent's goals:

$$f_{Interest}(\langle id', D', T', E', C', S', R' \rangle, Goals) = \begin{cases} 1 & \text{if } D' = \mathcal{O} \text{ and } C' \in Goals \\ 1 & \text{if } D' = \mathcal{F} \text{ and } \neg C' \in Goals \\ 0 & \text{otherwise} \end{cases}$$

- $f_{Expectations}$. This function considers the influence of the external enforcement on the agent's goals:

$$f_{Interest}(\langle id', D', T', E', C', S', R' \rangle, Goals) = \begin{cases} 1 & \text{if } R' \in Goals \text{ and } \neg S' \in Goals \\ 0.5 & \text{if } R' \in Goals \text{ or } \neg S' \in Goals \\ 0 & \text{otherwise} \end{cases}$$

The value calculated by the advisability function is used to annotate the goal that is added to the $NormativeGoals$ set. Each normative goal is annotated with a real number that represents to what extent the agent is interested in complying with the instance that has inferred the normative goal. If the instance is an obligation, a new goal to pursue the obliged condition is added. On the contrary, if the instance is a prohibition, a new goal for avoiding the forbidden condition is added.

Algorithm 7 Norm Advice Service Control Loop

```
1: while M_In is not empty do
2:   Retrieve m from M_In // m = ⟨Type, Sender, Receiver, Content⟩
3:   if Type = request and Receiver = NAS and Content = compliance(Goals) then
4:     NormativeGoals = {}
5:     RoleList = {}
6:     for all (agentID, roleID) in RE do
7:       if Sender = agentID then
8:         Add roleID to RoleList
9:       end if
10:    end for
11:    for all i in I do // i = ⟨id', D', T', E', C', S', R'⟩
12:      if T' in RoleList then
13:        if D' = F then
14:          NormativeGoals = NormativeGoals ∪ {(¬C', advisability(i, Goals))}
15:        end if
16:        if D' = O then
17:          NormativeGoals = NormativeGoals ∪ {(C', advisability(i, Goals))}
18:        end if
19:      end if
20:    end for
21:    Add ⟨inform, NAS, Sender, compliance(NormativeGoals)⟩ to M_out
22:  end if
23: end while
```

Let us assume that b_1 is only interested in buying cars. Therefore, its goal set only contains one proposition $\langle bid, -, item(car) \rangle$. Since it wants to obey norms, its asks NAS about its normative goals by sending the following message:

$$\langle request, b_1, NAS, \{\langle bid, -, item(car) \rangle\} \rangle$$

The NAS checks the instances that are relevant to b_1 and calculates the advisability of complying with them. Only the instance that has been created out of n_1 is relevant to b_1. The advisability of complying with this instance is 0 and the NAS sends the following message:

$$\langle inform, NAS, b_1, compliance(\{(\neg\langle bid, -, item(car)\rangle, 0)\})\rangle$$

Since $\neg\langle bid, -, item(car)\rangle$ contradicts the main goal of agent b_1 and the advisability of complying with this norm is 0, then agent b_1 decides to violate the norm and it makes a bid.

In this paper we propose to determine the agent advisability of complying with a given instance by simply considering the effect of this instance on the agent goals. However, if the norm reasoning services are provided with domain information, then a more complex decision making procedure could be used. For example, the decision making mechanism proposed in [6] could be used if the norm reasoning services are informed about the importance of each norm and the situations that are predicted to occur when the norms are violated.

It should be noticed that the NAS does not ensures that the set of normative goals is consistent: i.e., it is possible that a proposition and its negation belong to the set of normative goals. It is the responsibility of agents to decide which of the normative goals will be pursued, resolving conflicts between normative goals and their intrinsic goals.

3.2.4 Norm Judgement Service (NJS)

Finally, suppose that a *seller* agent, identified by s_1, receives the bid made by b_1. s_1 may be unable to judge whether this bid is legal or not with respect to the normative system. Specifically, it may want to know whether b_1 has violated any norm and its bid must be ignored, or whether b_1 has acted legally and b_1 wins the auction.

The Norm Judgement Service (NJS) allows agents to determine if an event that may have happened at some moment in the past is legal or not according to the normative system. Therefore, the NJS judges the performance of this event with respect to the context (i.e., active instances and roles, and their interplay) when the event was performed. Algorithm 8 contains the source code corresponding to this functionality. When the NJS receives a request then it determines: which roles were played by the agent that performed the event ($Origin$), at the time the event was performed ($Time$). Then, the NJS determines which norms were relevant to that agent at that time. Agents can play several roles simultaneously and these roles may be affected by conflicting norms. For this reason, the NJS counts the number of prohibitions that were violated by the event ($FulfilmentCount$), and the number of obligations that were fulfilled by the event ($ViolationCount$). It is up to the agents how to use this information: e.g., to select the most suitable interaction partners, to exclude non-compliant agents, etc.

In our example, s_1 asks NJS about the bid made by b_1 by sending the following message:

$$\langle request, b_1, NJS, \langle bid, time, b_1, item(car)\rangle\rangle$$

The NJS answers with the following message:

$$\langle inform, NJS, b_1, normJudgement(1,0)\rangle$$

Then s_1 is able to know that the bid is illegal, and thus, s_1 ignores.

Algorithm 8 Norm Judgement Service Control Loop

```
1: while M_In is not empty do
2:   Retrieve m from M_In // m = ⟨Type, Sender, Receiver, Content⟩
3:   if Type = request and Receiver = NJS and Content = judgement(e) then // e =
     ⟨Type, Time, Origin, Data⟩
4:     RoleList = {}
5:     for all ((agentID, roleID), t_In, t_Out) in RELog do
6:       if Origin = agentID and t_In ≤ Time and (t_Out = null or t_Out > Time) then
7:         Add roleID to RoleList
8:       end if
9:     end for
10:    ViolationCount = 0
11:    FulfilmentCount = 0
12:    for all i in ILog do // (⟨id', D', T', E', C', S', R'⟩, t_In, t_Out)
13:      if T' in RoleList and t_In ≤ Time and (t_Out = null or t_Out > Time) then
14:        if unification(e, C') then
15:          if D' = O then
16:            FulfilmentCount = FulfilmentCount + 1
17:          end if
18:          if D' = F then
19:            ViolationCount = ViolationCount + 1
20:          end if
21:        end if
22:      end if
23:    end for
24:    Add ⟨inform, NJS, Sender, normJudgement(ViolationCount, FulfilmentCount)⟩ to
       M_out
25:  end if
26: end while
```

4 Discussion

The idea of providing agents with normative information is not new. In [8], Felicíssimo et al. propose a solution for continuously supporting agents with updated norm information. In the proposal of Felicíssimo et al. the scope of norms is defined using contexts. Therefore, agents are provided with information about the norms that are in force in their current context. Similarly, in [13] Okuyama et al. propose the definition of *normative objects* that allow agents to be informed about the norms that regulate their context. However, these two solutions do not provide agents with information about norm dynamics (i.e., the activation and expiration of norms), norm compliance and norm judgement. Therefore, these functionalities must be implemented by agent programmers at agent level.

Other solutions that provide agents with normative information are based on the use of normative artifacts. Artifacts are resources and tools that agents can create and use to perform their individual and social activities [14]. For example, the ORA4MAS [11] proposal defines artifacts as first-class entities to instrument multi-agent organisations to support agent activities within them. In the ORA4MAS the

enforcement of norms has been implemented by means of artifacts, which detect norm violations; and by means of agents, which are informed about norm violations and carry out the evaluation and judgement of these situations. Therefore, the agent designers are responsible for programming agents endowed with capabilities for performing these tasks. In [19], Tinnemeier et al. propose a language for programming normative artifacts that are responsible for detecting when norms are active, detecting violations of the norms and applying sanctions and rewards. However, the language developed by Tinnemeier et al. is not suitable for programming normative artifacts that provide agents with information about norm activations, violations and norm enforcement. Finally, in [16] Piunti et al. propose that normative artifacts provide a series of observable properties that can be inspected by agents to know the actual normative state of the organisation. Therefore, agents are able to know which norms are active at a given moment. Autonomous agents can use this information to reason about whether to follow or not the norms that are active. Our proposal also allows agents to know the current normative state. Moreover, our proposal provides agents with norm compliance services that help agents to make decisions on whether or not to follow the norms that are active. Finally, our proposal also manages information about the past normative states, which allows agents to judge the behaviour exhibited by them and by other agents. To our knowledge, this is the first proposal that provides this kind of normative information.

As future work, we plan to improve the norm judgement service to deal with norm conflicts. Currently, the norm judgement service only considers how many norms are violated and fulfilled by an event. In the future, we plan to annotate norms with their salience. The salience of norms determines the hierarchy of norms. With this information the judgement process will be able to determine not only the number of norms that are violated and fulfilled by an event, but also to determine if an event may be considered as an offence in case of norm conflict.

5 Acknowledgments

This paper was partially funded by the FPU grant AP-2007-01256 awarded to N. Criado and by the Spanish government under grants CONSOLIDER-INGENIO 2010 CSD2007-00022, TIN2009-13839-C03-01. This research has also been partially funded by Valencian Prometeo project 2008/051.

References

[1] A. Artikis and J. Pitt. A formal model of open agent societies. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 192–193, 2001.

[2] J. Broersen, M. Dastani, J. Hulstijn, Z. Huang, and L. van der Torre. The boid architecture: conflicts between beliefs, obligations, intentions and desires. In *Proceedings of the fifth international conference on Autonomous agents*, pages 9–16. ACM, 2001.

[3] L. Burdalo, A. Garcia-Fornes, V. Julian, and A. Terrasa. TRAMMAS: A tracing model for multiagent systems. *Engineering Applications of Artificial Intelligence*, 24(7):1110–1119, 2011.

[4] N. Criado, E. Argente, and V. Botti. Open Issues for Normative Multi-Agent Systems. *AI Communications*, 24(3):233–264, 2011.

[5] N. Criado, E. Argente, P. Noriega, and V. Botti. A Distributed Architecture for Enforcing Norms in Open MAS. In *Advanced Agent Technology*, volume 7068, pages 457–471. Springer, 2012.

[6] N. Criado, E. Argente, P. Noriega, and V. Botti. Determining the Willingness to Comply With Norms (Extended Abstract). In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

[7] N. Criado, V. Julián, V. Botti, and E. Argente. A norm-based organization management system. *Coordination, Organizations, Institutions and Norms in Agent Systems V*, pages 19–35, 2010.

[8] C. Felicíssimo, C. Chopinaud, J. Briot, A. Seghrouchni, and C. Lucena. Contextualizing normative open multi-agent systems. In *Proc. of the ACM symposium on Applied computing*, pages 52–59. ACM, 2008.

[9] R. L. Fogus, J. M. Alberola, J. M. Such, A. Espinosa, and A. Garca-Fornes. Towards Dynamic Agent Interaction Support in Open Multiagent Systems. In *Proc. of the International Conference of the Catalan Association for Artificial Intelligence (CCIA)*, volume 220, pages 89–98. IOS Press, 2010.

[10] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200, 2001.

[11] J. Hubner, O. Boissier, R. Kitio, and A. Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.

[12] S. Modgil, N. Faci, F. Meneguzzi, N. Oren, S. Miles, and M. Luck. A framework for monitoring agent-based normative systems. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 153–160, 2009.

[13] F. Okuyama, R. Bordini, and A. da Rocha Costa. A distributed normative infrastructure for situated multi-agent organisations. In *Proc. of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1501–1504, 2008.

[14] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.

[15] N. Oren, S. Panagiotidi, J. Vázquez-Salceda, S. Modgil, M. Luck, and S. Miles. Towards a formalisation of electronic contracting environments. *Coordination, Organizations, Institutions and Norms in Agent Systems IV*, pages 156–171, 2009.

[16] M. Piunti, A. Ricci, O. Boissier, and J. Hubner. Embodying organisations in multi-agent work environments. In *Proc. of IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT)*, volume 2, pages 511–518, 2009.

[17] R. Rubino and G. Sartor. Preface. *Artificial Intelligence and Law*, 16:1–5, 2008.

[18] J. M. Such, A. Espinosa, A. García-Fornes, and V. Botti. Partial Identities as a Foundation for Trust and Reputation. *Engineering Applications of Artificial Intelligence*, 24(7):1128–1136, 2011.

[19] N. Tinnemeier, M. Dastani, J. Meyer, and L. Torre. Programming normative artifacts with declarative obligations and prohibitions. In *Proc. of IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT)*, volume 2, pages 145–152, 2009.

Social Multi-agent Simulation Framework

Michal Wrzeszcz[1] and Jacek Kitowski[1]

[1] *AGH University of Science and Technology, Faculty of Electrical Engineering, Automatics, Computer Science and Electronics, Department of Computer Science, Krakow, Poland,* {wrzeszcz,kito}@agh.edu.pl

Abstract

Human societies appear in many types of simulations. Although, simulation of people is a very complex problem, models of agents that well reproduce behaviour of a single individual already exist, often however, neglecting social context. People behaviour, when they are alone, is different than their behaviour when they are in group. In our work we study social networks as the instrument that may provide social context. The main problem with use of social networks is that relations in social networks should be changed automatically during the simulation as a result of agents' actions (especially in consequence of exchange of messages) what is not offered by the existing models of social networks. This fact encouraged us to extend the social network model to allow creation of simulations of human societies. An experimental proof of concepts is also presented.

Keywords: social networks, behaviour modelling, simulation of human societies, multi-agent systems, social context.

1 Introduction

Simulations of human societies may be used to create virtual worlds that allow to predict collective behaviour of crowd e.g. they can be used to check if a demonstration can turn into a riot. Furthermore, these virtual worlds may be used to show people how they should behave in particular situations e.g. policemen can take part in virtual demonstration to check which actions may cause increase of crowd's aggression. Simulations of human societies are also used to entertain people. There are a lot of computer games (especially role-playing games) where individuals controlled by artificial intelligence live in cities and villages. These individuals interact with the player. Their attitude to the player often depends on the information about player's actions that they poses.

In all applications, the simulations of human societies are useful only if they are highly realistic so the main challenge is to make behaviour of simulated individuals as similar as possible to human behaviour. In this article we present a framework that may be used to build and simulate an artificial society on the basis of the extended model of the social network. However, the presented framework does not provide any complete simulation. It focuses on provision of mechanisms that may

be used to create the social context. Agents' models are not included - to create them the user may adapt currently known techniques (see section 2.1).

The rest of the paper is organized as follows. In Section 2, we discuss important elements of simulations of human societies. Section 3 describes how social networks can be applied during these simulations. Next, in section 4 we discuss an information flow in a social network to justify the need of extension of the social network model. Then, in section 5, the proposed model is mathematically described. Section 6 includes description of implementation of this model and of the whole framework. In section 7, we show a sample simulation that makes use of the framework and proves that proposed social network model can be easily rebuilt during the simulation. Section 8 contains discussion about relationship between classical social network and communication channels. We conclude this paper in section 9 and describe future work directions of our research.

2 Simulation of human societies

2.1 Virtual human

Virtual human society contains virtual humans that co-operate, negotiate, make friends, communicate etc. Simulation of these individuals requires definitions of four main ingredients [23]: high-level behaviour, perception, animation and graphics. In this article we will focus only at first and second aspect.

To model perception of individuals we can adopt software agents interacting with the environment [19]. The agents are independent beings, hence they can be used to model virtual humans. Therefore, in the paper we will treat virtual human society as multi-agent system. Several agent architectures were created to enable behaviour modelling, e.g., FSMs (Finite State Machines), BDI (Belief, Desire, Intention) and PECS models (Physical conditions, Emotional state, Cognitive capabilities and Social status) [21]. Research showed that agents that base on PECS models and ontology can be configured to simulate specific scenarios [12]. Besides making decisions, the agents can move. Several mobility models have been proposed, e.g., random walk models [17], encounter based models [22], community based models [16], time-variant models [6] and day movement model [5]. The pattern based mobility model proposed by Nazir, Prendinger and Seneviratne [18] well reproduces day-to-day human activities of common people. These facts show that agents that well reproduce behaviour of a single individual can be created using currently known techniques so we will not consider any particular behaviour model during our research, instead we provide the social context that can be used by many models.

2.2 Social context and social simulations

When modelling human behaviour, we have to remember that behaviour of each human depends on social context. People behaviour when they are alone is usually different from their behaviour when they are in a group. Although the most obvious way of influencing agent behaviour is direct communication, there are also other possibilities, e.g., use of nonverbal communication elements (observation of other

agents) - posture (position of the arms and legs etc.) can indicate somebody's feelings [23]. However, regardless the way of influencing agent behaviour, a key issue is the decision which agents influence each other and how strong this effect is. In the EUSAS project (European Urban Simulation for Asymmetric Scenarios) [11, 12, 13], that uses virtual worlds to predict human behaviour, the social influence of agents is modelled according to Latan's formula of strength, immediacy and number of other agents [14]. It means that observation of agents that belong to the same group can change the agent's internal state. The strength of this effect depends on social position of observed agent. Other model was introduced by James March [15]. It uses an organizational code that represents the organization's approximation of beliefs about that reality. Each period, every individual agent alters any given belief to conform to that of the organizational code with some probability, reflecting the rate of socialization of individuals in the organization. The organizational code also alters any given belief based on the dominant belief of the set of agents - the superior group, defined as those agents whose individual beliefs correspond better with reality than does the code's. The model was later extended by Kane and Prietula [7]. They let individuals learning from (being influenced by) one another rather than the organizational code by introducing a parameter that represented the probability that a given individual will learn from the other individuals rather than the code in a given period. However, analysing this type of models, we have to remember that agents in multi-agent systems easily develop erroneous beliefs [4]. Another interesting approach is use of the tag-based computational model [2]. An agent cooperates with another agent only if they are similar (i.e., their tags are sufficiently similar).

The described models use global parameters or agents' parameters to decide which agents influence each other and to calculate the strength of this influence. However, we believe that without the representation of the information about the relationship between each pair of people the model is incomplete. A friendship relation between people may strengthen the influence that one person exerts on the other, e.g., the influence of our good friend may be higher than influence of the leader of the group that has higher social position. It is the reason why we think that a good model should contain a social network.

3 Social networks and agent simulations

A well-known concept connected with the multi-agent systems is the reputation system. However, each agent needs some information about others to form opinions about them. Direct interaction is the most reliable source of information needed to form an opinion about another person. Unfortunately, this information is not always available. In this case, the social network may be used to provide social reputation that depends on witness reputation, neighbourhood reputation and system reputation. This mechanism is described in [20]. The reputation can be used to provide social context. At each step of the simulation the agent should do the following actions:

- receive messages from agents with which it can communicate,

- use these messages to modify its plans/behavior - if it has a good opinion about the sender, the influence of the message will be higher,
- use these messages to modify its opinions about other agents when messages contain information about them,
- do action.

Although we know how to use the information represented by the social network to provide social context (using reputation system), creation of a mechanism that manages information flow in the social network is still a challenge.

4 Information flow in social networks

The previous studies of the information flow in a social network have tried to answer the question how people select the next person to whom to forward the message [3]. In [18] authors consider a way of searching of a person with some specific knowledge desired by an agent. However, even if the sender and the receiver of the message are known, it is necessary to check if direct communication between these nodes of the social network can occur at a particular step of the simulation - i.e., to check if any communication channel[1] exists.

Communication channels have appeared in articles about social networks before, e.g., in [18] the authors use them to define Send and Receive Protocol. However, they do not define a method of building and destroying them. We reckon that it is very important issue and the model of the social network should support managing communication channels. Communication channels appeared also in the literature about Multi-layered Social Networks. Multi-layered Social Network is defined as a tuple $< V, E, L >$ where: V is a non-empty set of nodes, E is a set of edges where each edge belongs to exactly one layer and L is a set of layers [1]. Each layer corresponds to one type of relationship between users [9]. A network can be built where each type of relationship (each layer) is connected with one type of communication channel [8]. However, using Multi-layered Social Network we cannot check if any communication channel still exists or can be recreated at a particular step of the simulation - we can only use relation which was built when this channel existed.

Another model that can be used to represent the ability of communication is Multidimensional Social Network model. It allows to observe changes of the network and contains three distinct dimensions: layer dimension that describes all relationships between the users of a system, time-window dimension that allows temporal analysis and group dimension which describes subsets of users [10]. From our point of view, the time-window dimension is the most interesting one because it allows to observe the evolution of the network. The time-window dimension approach limits social network to those nodes and relationships that have existed in a period defined by the time-window size. Unfortunately the time-window approach

[1] In this article, we define communication channel as a link between people that allows sending of messages. Similar communication channels form a type (e.g. telephone communication channels types).

has a serious disadvantage. If we liked to model communication channels that way, it would bind communication channel with the relation. If a relation between particular nodes exists, the communication channel between them also exists (in this model). It means that if we liked to model a situation when the communication channel between particular nodes did not exist in a particular time-window, we would have to delete a relation between these nodes. Afterwards, when possibility of communication was restored, we should recreate the relation. This type of modelling causes a problem. We are not able to change the relation between nodes when these nodes cannot communicate. In the real life we can change opinion about someone in the situation when we are not able to contact him, e.g., we can change our opinion about someone when we receive information about him from our friend.

5 Proposed social network model

Analysis of information flow encouraged us to extend Multi-layered Social Networks model. This extended model is defined as a tuple $< V, E, L, C, T >$ where:

- V - a non-empty set of nodes;
- E - a set of relations; each relation is a tuple $< x, y, l >$, where x,y are different nodes and l is a layer $(x, y \in V, l \in L, x \neq y)$;
- L - a set of layers; a layer is a set of relations of the same type (e.g. trust layer, family ties layer); maximum two relations between particular nodes (x to y and y to x) belong to each layer (up to $|V| \cdot (|V| - 1)$ relations belong to each layer): $< x, y, l > \in E \wedge < x', y', l' > \in E \wedge x = x' \wedge y = y' \Rightarrow l \neq l'$;
- C - a set of communication channels; each communication channel is a tuple $< x, y, t, n >$, where x,y are different nodes, t is communication channel type and n is a number of the channel $(x, y \in V, t \in T, n \in N, x \neq y)$;
- T - a set of types of communication channels (e.g. face to face communication channel type, telephone communication channel type, e-mail communication channel type); communication channels are directed (e.g. x can know telephone number of y, while y does not know telephone number of x); two nodes can be connected by more than one channel of each type, e.g., one person can know two different telephone numbers to a friend: $< x, y, t, n > \in C \wedge < x', y', t', n' > \in C \wedge x = x' \wedge y = y' \wedge t = t' \Rightarrow n \neq n'$.

The most important novelty is a set of communication channels, which could be organized in different channels types. They represent the ability to exchange information between nodes (x can send message to y when any communication channel from x to y exists). Using many types of communication channels is reasonable because there may exist many ways of communication between nodes and agent should be able to choose which channel it wants to use (e.g. use of e-mail is cheaper than use of a telephone but the message will be delivered faster using telephone).

Each step of the simulation contains the following steps:

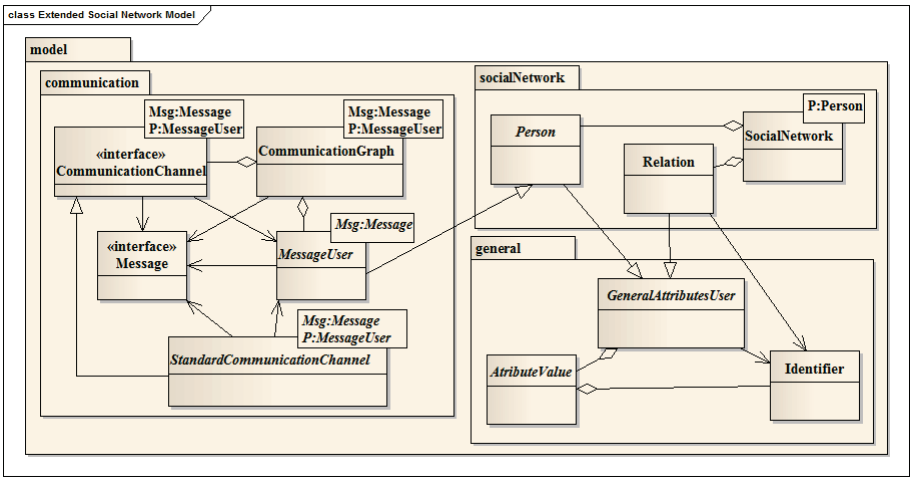


Figure 1: Implementation of extended social network model.

- automatic rebuilding of communication network (creation of new channels if new communication ability appears, delete of channels which are out-of-date),
- agents' actions which may result in relations' change (see section 3 of this paper).

Ability of automatic managing of communication channels is very important because it creates a base for information exchange and social context provision. More information about automation that rebuilds the communication network can be found in section 6.1 while section 7 contains an example of automaton use.

6 Implementation of simulation framework

The framework was implemented using the Java language, the MASON framework and the JUNG framework. It contains templates of classes to allow the user to create their own implementations of agents, communication channels etc.

The classes that implement the social network model, described in section 5, are shown in Figure 1. Package "socialNetwork" contains a representation of standard social network model. Subclasses of class "Person" are nodes of the network while instances of "Relation" class are edges. Each instance of "Relation" belongs to a layer described by instance of "Identyfier" class. Package "general" includes classes that may be used to create attributes for nodes and edges.

Package "communication" implements the proposed new abstraction layer. The communication graph contains subclasses of "CommunicationChannel" graph. The user may create any number of "CommunicationChannel" subclasses . However, each class must give ability to identify the type of channel and mechanism for propagating messages. For most of the users, it is a good idea to use the standard communication channels which provide a mechanism of messages propagation that calculates time and cost of sending of information on the basis of message length.

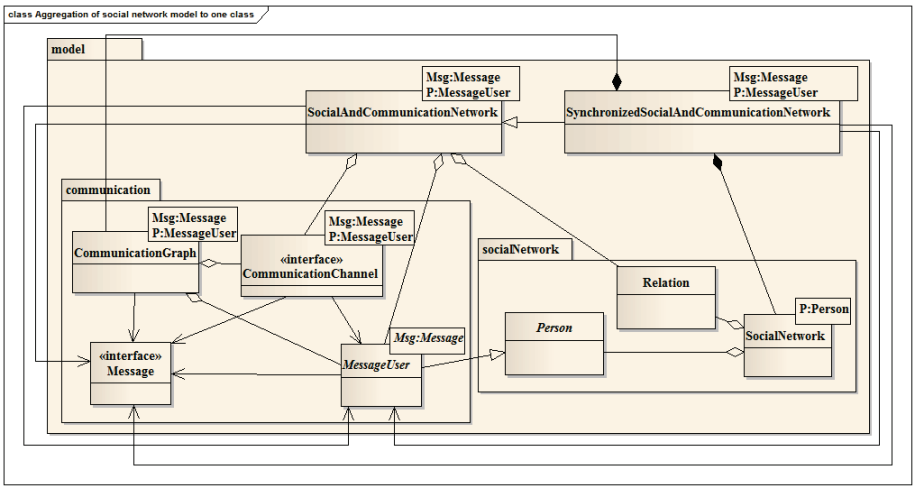


Figure 2: Aggregation of social network model to one class.

The model was aggregated to one class called "SocialAndCommunicationNetwork" (see Figure 2). This class represents social relations and communication channels as one multi-graph. Moreover, "SynchronizedSocialAndCommunicationNetwork" class was created. It contains three graphs - aggregated model from "SocialAndCommunicationNetwork" class, social network graph and communication graph. It may be used instead of "SocialAndCommunicationNetwork" class to provide better visualization (visualization is discussed in section 6.2).

6.1 Simulation and environment model

The classes that provide simulation tools and simulation environment are shown in Figure 3. An "AbstractCrowd" class is a template of the simulation. It contains agents and all information needed for visualization. It also allows to inject automatons that manage simulation. Automaton is an object which provides a function that is executed once at the beginning of each step of the simulation. The most important role of automatons is managing of communication channels. Each simulation should contain at least one automaton. The framework does not contain any universal automaton because different types of communication channels require different management. The users may also implement and use custom managers for their own purposes.

"Environment" class contains extended social network model implementation. Although the user may add social network to simulation manually (by creating a subclass of "AbstractCrowd"), we propose to use "Environment" subclasses and define all relations within them. It allows to visualize social relations and communication channels using tools provided with framework. "StandardCrowd" class adds to "AbstractCrowd" an easy way of simulation creation (using crowd creators) while "StandardCrowdWithEnvironment" binds the simulation with the Environment.

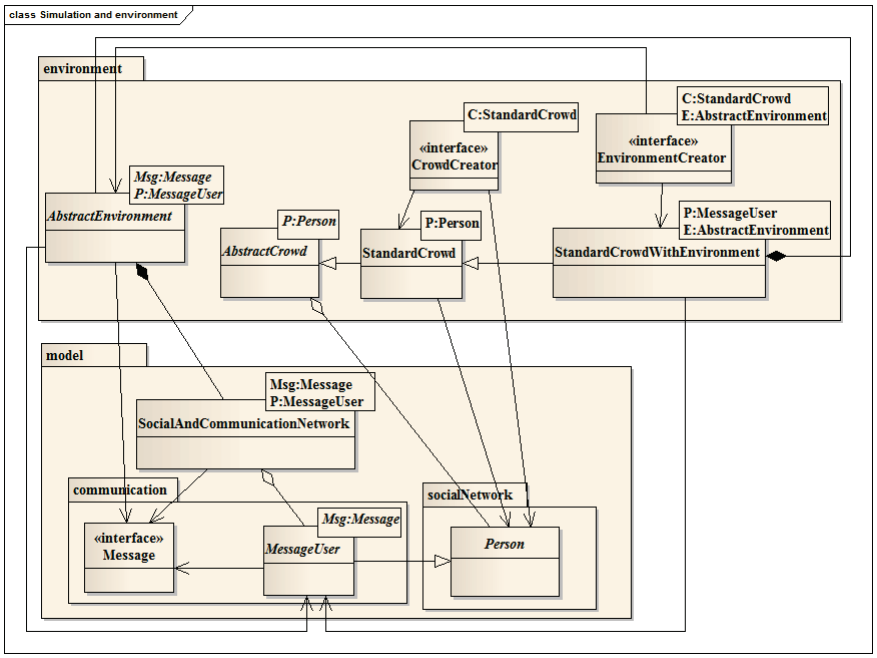


Figure 3: Implementation of simulation and environment.

6.2 Visualisation

Athough aspects connected with visualisation of individuals are not considered in this article, it is worth to mention that visualization of social network and communication channels can help users during the analysis of results. Figure 4 shows classes that provide this type of visualization. The framework allows to show only a simulation ("CrowdVisualization" class), only graphs ("GraphsVisualization" class) or simulation and graphs together ("CrowdAndGraphsVisualization" class).

7 Automatic rebuilding of reputation graph using created framework

7.1 Simulation scenario

To demonstrate automatic rebuilding of the social network (to reflect changes in mutual reputation), we created a simulation in which people live in two cities. The social network describes agents' reputation (each agent can have its own opinion about another agent - an agent can be respected by one agent and not respected by another).

In the simulation we distinguish four agents: A1.1 and A1.2 live in the first city while A2.1 and A2.2 live is the second city. They know each other. Assume, that agent A1.1 possesses an information that discredits agent A1.2 and agent A1.2 possesses an information that discredits agent A1.1 (each information is supported

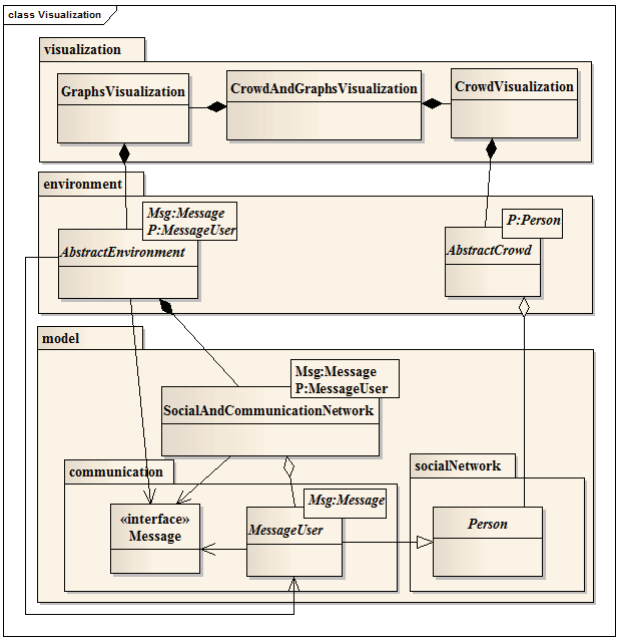


Figure 4: Classes connected with the visualization.

by evidence so receiver of this information will change opinion about the subject of information - reputation of the subject will be reduced to zero). We want to show that reputation of agents A1.1 and A1.2 depends on their behaviour.

7.2 Implementation of simulation and classes that provide automatic rebuilding of reputation graph

Figure 5 shows the classes that we implemented to create this simulation with classes together with their class parents. To enable the information flow we created two types of communication channels: face to face communication channels and telephone communication channels. Moreover, we created face to face communication channels manager which is injected to the simulation class and creates channels when the ability of communication between agents appears (i.e., in the case of face to face communication when the distance between them is short). When the distance between agents is growing, the manager automatically deletes channel between them. This class is very important because it provides automatic rebuilding of the communication network. Namely, nodes which may communicate at particular step of the simulation are known, this information may be used for creation or changing relations between nodes.

We had also to create classes that represent agent ("Citizen" class), to create social network that describe reputation ("demo.reputation" package) and a class that creates simulation ("CrowdAndEnvironmentCreator") e.g. sets initial positions of agents. Since the classes that represent and manage communication

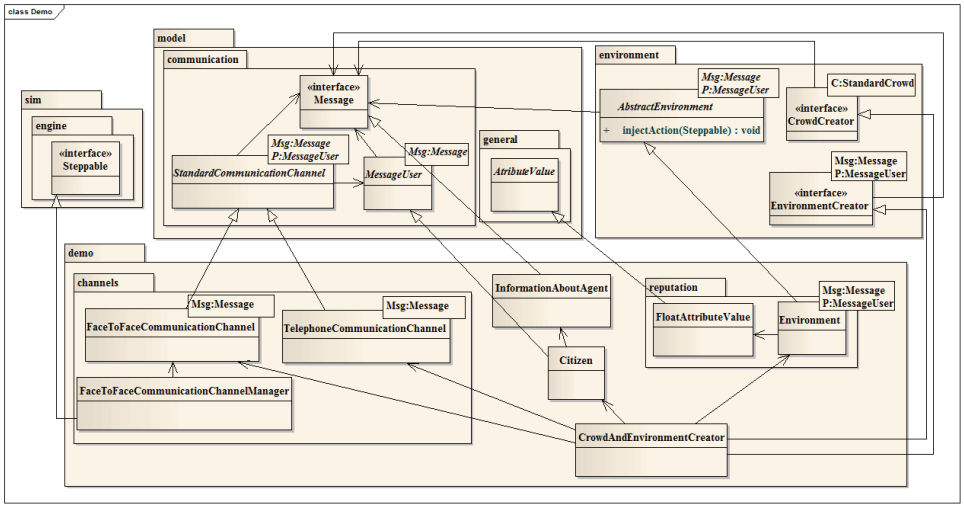


Figure 5: Classes that creates simulation classes with which they cooperate.

channels can be shared between many simulations, the user who wants to create a simulation has to create agents only, simulation creator and environment. The abstract environment contains a social network so during the environment creation the user only has to define a relation that will be used by the network.

Therefore, to start the simulation the user has to create an instance of a "StandardCrowdWithEnvironment" class (with the appropriate template arguments). Instances of "CrowdAndEnvironmentCreator" and "Environment" classes should be used as arguments of the constructor. Afterwards, the user can use the created object as standard MASON simulation or use one of visualization classes (e.g. "CrowdAndGraphsVisualization") to start simulation with GUI.

7.3 Results of the simulation

We have created several instances of the simulation with different initial conditions (use of the instance of "CrowdAndEnvironmentCreator" class with different parameters). During each simulation the agents (instances of "Citizen" class) from the first city travelled to the second city.

- In the first simulation agent A1.1 arrived to the second city faster than agent A1.2. A1.1 met there agents A2.1 and A2.2 and forwarded information (using face to face channel) about agent A1.2 so reputation of A1.2 decreased. When agent A1.2 arrived to the city, agents A2.1 and A2.2 did not want to talk with him so reputation of agent A1.1 did not change.

- In the second simulation agent A1.2 arrived to the city before agent A1.1 so the reputation of agent A1.1 decreased and reputation of agent A1.2 did not change.

- In the next simulations we added a telephone communication channel between

A1.1 and A2.1. At the beginning of each simulation A1.1 used this channel to send a message to A2.1. A2.1 and A2.2 live in the same city so A2.1 forwarded this message to A2.2 before agent A1.1 or A1.2 arrived to the city. It allowed agent A1.1 to keep its reputation regardless of who first came to the second city.

These results show that social network was rebuilt dynamically during the simulation on the basis of agents actions. The discussed cases show that we were able to create social context using our framework. Moreover, the amount of work needed to do it is not much bigger than amount of work needed to create the simulation without social context because most of classes that provide it are included in the framework or may be shared between many simulations.

8 Relationship between classical social network and communication channels

Social networks may be characterized by several metrics, e.g., centrality or page rank [24]. To better understand the extended social network model, we can check if change of a metric of the social network influences communication channels. During our research we have checked influence of density (a ratio of the number of edges over the number of possible edges) represented by density ratio and number of nodes in the network on dynamic creation of communication channels during the simulation. The size of the environment was constant, start positions of agents were initialized randomly. Afterwards, we iterated over all possible pairs of agents and used the following expression to decide if the relation should be created: $rand() \cdot (dist \div maxDist)^2 < densityRatio$, where (for the example from the previous section):

- rand() - random number generator,
- dist - Euclidean distance between agents,
- maxDist - maximal distance which allows agents to communicate.

A relation was created when the value of expression was true. Table 1 shows density ratio-density correspondence (the table presents average values of densities for runs with particular density ratios).

Density ratio	2	4	8	16	32	64	128	256	512	1024
Density	0,02	0,05	0,07	0,11	0,21	0,34	0,53	0,71	0,86	0,99

Table 1: Relationship between density ratio and density.

We assumed that each agent may communicate only with friends (relation between agents must exist) and may have open only three communication channels at each simulation step. The communication channels are created and deleted dynamically by automaton. The automaton tries to create the communication

channel between particular agents when the channel between them does not exist and the distance between them is less than or equal to maximal distance which allows agents to communicate. The operation succeeds when each of these two agents is connected with other agents by less than three communication channels, otherwise it fails. When the distance between agents rises above the maximum distance that enables communication, the automaton deletes the communication channel between them. Tables 2 and 3 present numbers of channel creation operations which succeeded and which failed. These results are averaged over 10 runs.

	Nodes number							
Density ratio	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**
2	0	0,7	7,4	15,1	62,7	211,2	789,9	3309,2
4	0,4	1,5	10,5	19,2	99,3	361,4	1340	5473
8	0	1,3	5,3	46,8	155,7	587,4	2203,1	8934,5
16	0	3,3	20,3	62,1	231,2	890,9	3656,4	13768,3
32	0,6	9,4	21,1	94,4	345,8	1440,3	5874,7	20691
64	3,7	7,7	34,4	132,5	554,7	2132	8185,2	28709,8
128	3,1	14,4	55,6	185,8	898,7	3193,9	11163,9	38092,2
256	3,7	18,6	70,6	268,3	1104,3	4041,5	14616,2	47232,7
512	4,6	18,5	86,5	357,6	1321	5016,7	17236,1	54373,1
1024	6	20,8	98,1	356,7	1415,2	5403,8	18479,2	58158,2

Table 2: Numbers of channel creation operations which succeeded.

	Nodes number							
Density ratio	**2**	**4**	**8**	**16**	**32**	**64**	**128**	**256**
2	0	0	0	0	0	0,4	4,4	80,9
4	0	0	0	0	0	0,7	14,2	338,7
8	0	0	0	0	0	3,9	67,9	1282,9
16	0	0	0	0,4	4,7	27,8	353,3	4497
32	0	0	0	0	0,5	104,2	1394,1	15352,6
64	0	0	0	0	16,1	270,9	3752	33426,5
128	0	0	0,4	1,1	120,2	830,3	9009,2	67064
256	0	0	0	15,1	168,4	2287,5	16590,4	121118,9
512	0	0	0	22	321,1	3244,8	25995,9	169237,7
1024	0	0	6	19,4	352,7	3512,2	30223,9	193689,7

Table 3: Numbers of channel creation operations which failed.

The number of channel creation operations which succeeded grows with growth of nodes number and density ratio. The number of operations which failed is much smaller than the number of operations that succeeded for small number of nodes and small density ratios (in most cases it is equal zero). However, when the density and number of nodes are high, the number of operations which failed starts very

fast growing. Moreover, when these numbers are very high, most operations fail. It shows that detailed studies of relation between social network parameters and communication channels are necessary.

9 Conclusions and future work

The tests have shown that the proposed social network model allows for dynamical creation of a social network that may be used to provide a social context of the simulation. Furthermore, ability of injecting automates that manage communication channels makes the framework easy to use (once created, the channel types and managers may be reused in many simulations).

Regarding the future work, there are three possibilities. The first can focus on studies of relation between social network parameters and communication channels (see section 8). The second can concentrate on groups of agents added to our social network model. This extension addresses the case when an agent hears a voice that comes from a group of people but is not able to evaluate which agent produced it. The information that this sound provides should be evaluated in the context of relation between the agent and this group. The third possibility is exploration of capability of parallelization of the social network representation. When the network is large, it is impossible to manage it using only one thread. We believe that analysis of dynamics of communication channels can allow us to improve network decomposition algorithms.

Acknowledgments. This work is partially supported by the EDA project A-0938-RT-GC EUSAS and European Union project UDA-POKL.04.01.01-00-367/08-00.

References

[1] P. Brodka, T. Filipowski, and P. Kazienko. An introduction to community detection in multi-layered social network. In *WSKS 2011, The 4th World Summit on the Knowledge Society*, 2011.

[2] Y. Chen and M. Prietula. To deceive or not to deceive? mimicry, deception and regimes in tag-based models. In *Intra-Organizational Networks (ION) Conference*, 2005.

[3] P. S. Dodds, R. Muhamad, and D. J. Watts. An experimental study of search in global social networks. *Science*, 301:827–829, 2003.

[4] J. Doran. Social simulation, agents and artificial societies. In *Third International Conference on Multi-Agent Systems*, pages 4–5, 1998.

[5] F. Ekman, A. Kernen, J. Karvo, and J. Ott. Working day movement model. In M. Kim, C. Mascolo, and M. Musolesi, editors, *MobilityModels*, pages 33–40. ACM, 2008.

[6] W. Jen Hsu, T. Spyropoulos, K. Psounis, and A. Helmy. Modeling time-variant user mobility in wireless mobile networks. In *INFOCOM*, pages 758–766. IEEE, 2007.

[7] G. Kane and M. Prietula. Influence and structure: Extending a model of organizational learning. In *Twelfth Annual Organizational Winter Science Conference*, 2006.

[8] P. Kazienko, P. Brodka, and K. Musial. Individual neighbourhood exploration in complex multi-layered social network. In *Web Intelligence/IAT Workshops*, pages 5–8. IEEE, 2010.

[9] P. Kazienko, P. Brodka, K. Musial, and J. Gaworecki. Multi-layered social network creation based on bibliographic data. In A. K. Elmagarmid and D. Agrawal, editors, *SocialCom/PASSAT*, pages 407–412. IEEE Computer Society, 2010.

[10] P. Kazienko, K. Musial, E. Kukla, T. Kajdanowicz, and P. Brodka. Multidimensional social network: Model and analysis. In P. Jedrzejowicz, N. T. Nguyen, and K. Hoang, editors, *ICCCI (1)*, volume 6922 of *Lecture Notes in Computer Science*, pages 378–387. Springer, 2011.

[11] B. Kryza, D. Krol, M. Wrzeszcz, L. Dutka, and J. Kitowski. Interactive cloud data farming environment for military mission planning support. In *Cracow Grid Workshop '11*, in press.

[12] M. Kvassay, L. Hluchy, B. Kryza, J. Kitowski, M. Seleng, S. Dlugolinsky, and M. Laclavk. Combining object-oriented and ontology-based approaches in human behaviour modelling. In *Applied Machine Intelligence and Informatics (SAMI), 2011 IEEE 9th International Symposium on*, pages 177–182, 2011.

[13] M. Laclavk, S. Dlugolinsky, M. Seleng, M. Kvassay, B. Schneider, H. Bracker, M. Wrzeszcz, J. Kitowski, and L. Hluchy. Agent-based simulation platform evaluation in the context of human behavior modeling. In *Agent-based simulation platform evaluation in the context of human behavior modeling, the second international workshop on Infrastructures and Tools for Multiagent Systems*, pages 1–15, 2011.

[14] B. Latane. Dynamic social impact. *Philosophy and Methodology of the Social Sciences*, 23:287–310, 1996.

[15] J. G. March. Exploration and exploitation in organizational learning. *Organization Science*, 2(1):71 – 87, 1991.

[16] M. Musolesi and C. Mascolo. A community based mobility model for ad hoc network research. In *REALMAN '06 Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, 2006.

[17] P. Nain, D. F. Towsley, B. Liu, and Z. Liu. Properties of random direction models. In *INFOCOM*, pages 1897–1907. IEEE, 2005.

[18] F. Nazir, H. Prendinger, and A. Seneviratne. Participatory mobile social network simulation environment. In *ICC*, pages 1–6. IEEE, 2010.

[19] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[20] J. Sabater and C. Sierra. Reputation and social network analysis in multi-agent systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 475–482. ACM, 2002.

[21] B. Schmidt. Modelling of human behaviour: The pecs reference model. In A. Verbraeck and W. Krug, editors, *14th European Simulation Symposium*, 2002.

[22] F. Tan, Y. Borghol, and S. Ardon. Emo: A statistical encounter-based mobility model for simulating delay tolerant networks. In *WOWMOM*, pages 1–8. IEEE, 2008.

[23] D. Thalmann. Simulating a human society: The challenges. In *Computer Graphics International, CGI02*, pages 25–38, 2002.

[24] I. Varlamis, M. Eirinaki, and M. D. Louta. A study on social network metrics and their application in trust networks. In N. Memon and R. Alhajj, editors, *ASONAM*, pages 168–175. IEEE Computer Society, 2010.