PhD THESIS

TRAMMAS: Enhancing Communication in Multiagent Systems

Author: Luis Antonio Búrdalo Rapa

Advisors: Dr. Andrés Terrasa Barrena
Dr. Vicente Julián Inglada

Departamento de Sistemas Informáticos y Computación,
Universidad Politècnica de València,
Valencia, Spain

January 2016

A mi familia, tanto a los que me han aguantado todo este tiempo, como a los que aún no saben leer ;)
Y especialmente a ti, papá. Cómo me jode que no hayas podido tener esto entre tus manos...

Agradecimientos

Desde que empecé, allá por el 2004, he ido leyendo los agradecimientos en las tesis de mis compañeros, al tiempo que fantaseaba con lo que pondría en los de la mía. No me explico cómo, después de tantos años pensando, no se me ocurre nada ahora que por fin me toca a mí. Me vienen tantos momentos y personas a la mente que me siento como un tiburón frente a un banco de peces: no sé ni por qué o quién empezar.

Gracias sobre todo a mi familia, por estar ahí siempre y ser una de esas patas que a veces me falta para mantenerme en pie y seguir caminando. Es algo que se suele dar por hecho, pero en realidad no todo el mundo tiene la suerte que he tenido yo con ellos.

Gracias a Andrés y Vicente, mis dos directores, por su paciencia y su tesón, por saber encarrilarme cuando tiendo al caos, por saber darme caña cuando ha llegado el momento para poder ponerle el punto final al trabajo de estos años y, sobre todo, por ser mucho más que unos directores de tesis. Creo que hace unos meses ninguno de los tres pensaba que este trabajo se llegase a acabar. Gracias por no tirar la toalla. ¡Cañas y pintas por doquier!

Gracias también a Vicente Botti por dejarme jugar con sus trenes allá por el 2002, por haber contado conmigo todo este tiempo y por amenazar con robarme las guitarras si no acababa. Gracias a Ana García-Fornes y a Agustín Espinosa, sin cuyo trabajo una parte importante de esta tesis no se habría llevado a cabo.

Gracias infinitas a todos los compañeros y amigos del GTI-YA (*GTI-IA Y Adyacentes*), por hacer que las horas de trabajo, los volcados de pila en hexadecimal, los malabarismos de LaTeX, los *accept* y *reject*, las noches, festivos y vacaciones al pie del cañón pasasen siempre entre cafés, pandorinos, cervezas, testeos de red, charlas de autobús, verbenas, conciertos, películas y muchas, muchas risas:

Gracias a Sole y Fanny, a las que conocí por casualidad durante la carrera, por las que conocí en primera instancia el trabajo que se llevaba a cabo en el grupo de investigación, a las que no cambiaba por nada y de las que no me he separado desde entonces. Gracias a Stella y Elena, por estar siempre ahí 24/7, por conocerme tanto y quererme aún más, por dejarme ser vuestra amiga, por dejarme formar parte de vuestras vidas y ser siempre una parte tan importante y bonita de la mía. Gracias a Carlos (*Doc 3C*) y a Damián, por madrugar conmigo todos los sábados durante muchos años con la excusa de dejarnos los dedos en las cuerdas y hacer que los madrugones en fin de semana apenas supusiese un esfuerzo. Gracias a Javi por compartir mis primeros pasos en un mundo musical en el que me encontré un poco más a mí mismo.

Gracias también a Pepe, Mari Carmen, Martí, Mafuente, Nancy, Natalia, Flora, Óscar y Patti, con quienes tuve la suerte de compartir laboratorio a lo largo de mis años en la universidad. Y gracias a tantos otros con quienes no compartí laboratorio, pero tuve la suerte de compartir trabajo, viajes, mesa a la hora de comer, alegrías, penas y el día a día todos estos años: Carlos G., Fabián, Juancho, Juan Ángel, Gus, Juanmi, Jose, Lluís, Jaume, Víctor, Sergio, Alejandro, María, Marlene, Jorge, Laura, Inma, Aída, Eva, Luis, Emilio, Miguel, Eliseo, Miguel Ángel, Adriana, Antonio, Óscar...

En realidad, con la mayoría de estas personas he compartido mucho más de lo que se puede abarcar en un par de páginas. Lo que más me molestó de verdad al dejar de trabajar como investigador en la universidad, fue dejar de veros casi a diario.

No puedo pasar por alto a un montón de personas con las que no compartí mi día a día laboral, pero sí muchas otras cosas de las que me he alimentado estos años. Muchos seguramente no sabéis cuánto me habéis ayudado, cuánto me seguís ayudando y cuánto os sigo necesitando de una u otra manera. Gracias a Pepe F. y Alicia, a Laura, a Lidón, a Fabiola, a Ana, a Valentina (ninaninanina), a César, a Daniela, a Carol, a Raúl, a Paco, a Mac, a Pai, a Giovanna, a Susana, a Vanessa, a Edu, a la gente de la Sanford Alligator Band y de Blue Dots y a muchos otros que me repiquetean en la cabeza ahora mismo... Si fuésemos los músicos del Titanic, me quedaría tocando con vosotros hasta el final sin dudarlo.

Gracias a todos por este viaje. Habría que ir pensando en embarcarse en el siguiente... ¿Quién se viene?

Resumen

A lo largo de los últimos años, los sistemas multiagente han demostrado ser un paradigma potente y versátil, con un gran potencial a la hora de resolver problemas complejos en entornos dinámicos y distribuidos, gracias a su comportamiento flexible y adaptativo. Este potencial no es debido únicamente a las características individuales de los agentes (como son su autonomía, y su capacidades de reacción y de razonamiento), sino que también se debe a su capacidad de comunicación y cooperación a la hora de conseguir sus objetivos. De hecho, por encima de la capacidad individual de los agentes, es este comportamiento social el que dota de potencial a los sistemas multiagente.

El comportamiento social de los sistemas multiagente suele desarrollarse empleando abstracciones, protocolos y lenguajes de alto nivel, los cuales, a su vez, se basan normalmente en la capacidad para comunicarse e interactuar de manera indirecta de los agentes (o como mínimo, se benefician en gran medida de dicha capacidad). Sin embargo, en el proceso de desarrollo software, estos conceptos de alto nivel son soportados habitualmente de manera débil, mediante mecanismos como la mensajería tradicional, la difusión masiva, o el uso de pizarras, o mediante soluciones totalmente ad hoc. Esta carencia de un soporte genérico y apropiado para la comunicación indirecta en los sistemas multiagente reales compromete su potencial.

Esta tesis doctoral propone el uso del trazado de eventos como un soporte flexible, efectivo y eficiente para la comunicación indirecta en sistemas multiagente. La principal contribución de esta tesis es TRAMMAS, un modelo genérico y abstracto para dar soporte al trazado de eventos en sistemas multiagente. El modelo permite a cualquier entidad del sistema compartir su información en forma de eventos de traza, de tal manera que cualquier otra entidad que requiera esta información sea

capaz de recibirla. Junto con el modelo, la tesis también presenta una arquitectura abstracta, que redefine el modelo como un conjunto de funcionalidades que pueden ser fácilmente incorporadas a una plataforma multiagente real. Esta arquitectura sigue un enfoque orientado a servicios, de modo que las funcionalidades de traza son ofrecidas por parte de la plataforma de manera similar a los servicios tradicionales. De esta forma, el trazado de eventos puede ser considerado como una fuente adicional de información para las entidades del sistema multiagente y, como tal, puede integrarse en el proceso de desarrollo software desde sus primeras etapas.

Resum

Al llarg dels últims anys, els sistemes multiagent han demostrat ser un paradigma potent i versàtil, amb un gran potencial a l'hora de resoldre problemes complexes a entorns dinàmics i distribuïts, gràcies al seu comportament flexible i adaptatiu. Aquest potencial no és només degut a les característiques individuals dels agents (com són la seua autonomia, i les capacitats de reacció i raonament), sinó també a la seua capacitat de comunicació i cooperació a l'hora d'aconseguir els seus objectius. De fet, per damunt de la capacitat individual dels agents, es aquest comportament social el que dóna potencial als sistemes multiagent.

El comportament social dels sistemes multiagent solen desenvolupar-se utilitzant abstraccions, protocols i llenguatges d'alt nivell, els quals, al seu torn, es basen normalment a la capacitat dels agents de comunicar-se i interactuar de manera indirecta (o com a mínim, es beneficien en gran mesura d'aquesta capacitat). Tanmateix, al procés de desenvolupament software, aquests conceptes d'alt nivell son suportats habitualment d'una manera dèbil, mitjançant mecanismes com la missatgeria tradicional, la difusió massiva o l'ús de pissarres, o mitjançant solucions totalment ad hoc. Aquesta carència d'un suport genèric i apropiat per a la comunicació indirecta als sistemes multiagent reals compromet el seu potencial.

Aquesta tesi doctoral proposa l'ús del traçat d'esdeveniments com un suport flexible, efectiu i eficient per a la comunicació indirecta a sistemes multiagent. La principal contribució d'aquesta tesi és TRAMMAS, un model genèric i abstracte per a donar suport al traçat d'esdeveniments a sistemes multiagent. El model permet a qualsevol entitat del sistema compartir la seua informació amb la forma d'esdeveniments de traça, de tal forma que qualsevol altra entitat que necessite aquesta informació siga capaç de rebre-la. Junt amb el model, la tesi també presenta una arquitectura

abstracta, que redefineix el model com un conjunt de funcionalitats que poden ser fàcilment incorporades a una plataforma multiagent real. Aquesta arquitectura segueix un enfoc orientat a serveis, de manera que les funcionalitats de traça són oferides per part de la plataforma de manera similar als serveis tradicionals. D'aquesta manera, el traçat d'esdeveniments pot ser considerat com una font addicional d'informació per a les entitats del sistema multiagent, i com a tal, pot integrar-se al procés de desenvolupament software des de les seues primeres etapes.

Summary

Over the last years, multiagent systems have been proven to be a powerful and versatile paradigm, with a big potential when it comes to solving complex problems in dynamic and distributed environments, due to their flexible and adaptive behavior. This potential does not only come from the individual features of agents (such as autonomy, reactivity or reasoning power), but also to their capability to communicate, cooperate and coordinate in order to fulfill their goals. In fact, it is this social behavior what makes multiagent systems so powerful, much more than the individual capabilities of agents.

The social behavior of multiagent systems is usually developed by means of high level abstractions, protocols and languages, which normally rely on (or at least, benefit from) agents being able to communicate and interact indirectly. However, in the development process, such high level concepts habitually become weakly supported, with mechanisms such as traditional messaging, massive broadcasting, blackboard systems or ad hoc solutions. This lack of an appropriate way to support indirect communication in actual multiagent systems compromises their potential.

This PhD thesis proposes the use of event tracing as a flexible, effective and efficient support for indirect interaction and communication in multiagent systems. The main contribution of this thesis is TRAMMAS, a generic, abstract model for event tracing support in multiagent systems. The model allows all entities in the system to share their information as trace events, so that any other entity which require this information is able to receive it. Along with the model, the thesis also presents an abstract architecture, which redefines the model in terms of a set of tracing facilities that can be then easily incorporated to an actual multiagent platform. This architecture follows a service-oriented approach, so that the tracing facilities are

provided in the same way than other traditional services offered by the platform. In this way, event tracing can be considered as an additional information provider for entities in the multiagent system, and as such, it can be integrated from the earliest stages of the development process.

Contents

Part I

Introduction and Objectives

CHAPTER

1

Introduction

1.1 Motivation

Multiagent systems are a software engineering paradigm for designing and developing complex software systems, where autonomous software entities (*agents*) solve problems that are beyond the capacities or knowledge of each of them as individuals, by interacting with one another in terms of high level protocols and languages [128, 102, 90]. In the last years, multiagent systems have been proven to be a powerful and versatile paradigm with a big potential when it comes to solving complex problems in distributed environments.

For their development and execution, multiagent systems require a specific framework called *multiagent platform*, which is in charge of providing all the basic infrastructure required to create and manage the system [115]. These facilities habitually include run time management, agents' lifecycle management, and also some abstractions which allow for the development of multiagent systems without having to take into account the internals of the multiagent platform, or any detail on its interaction with the operating system.

Unlike it occurs in systems with just one singular agent, where goal achievement usually depends on the agent's reasoning power, success in multiagent systems comes to a large extent from agents' ability to perceive their *environment* [124] and other agents in the system [90], as well as from their capabilities to react according to that perceived information. This reaction can result in internal or external responses. Internally, agents can decide to change their goals, or their current strategy in order to achieve them. Externally, they can choose to modify their environmental conditions (if possible) or to interact with other agents in the system, in order to exchange (obtain and/or transmit) information with them. In turn, this information exchange becomes a new perception input for agents, which may then trigger new internal or external actions.

These environmental and social abilities make it possible for agents to associate and coordinate in complex ways, such as forming teams or taking part in negotiations or

auction protocols. Also, these capabilities make it possible for agents to incorporate abstract concepts like norms, agreements, reputation or trust to their internal logic, in order to achieve their goals in an effective and efficient way. However, despite being so fundamental, many aspects regarding the relation of agents with their environment and their sociability are, most of the times, poorly supported by multiagent platforms and by agent programming languages [26]. As a consequence, the actual capabilities of multiagent systems become limited in terms of functionality, versatility and scalability.

Regarding the environment, it is a common developing practice to consider the environment as a mere set of resources, external to agents and to their communication infrastructure, instead of considering it one of the multiagent system components, a *first-class abstraction* [124] with whom agents interact. As a consequence, non-agent entities are rarely considered in the system design or architecture and, even when they are considered, their interaction with agents is usually addressed in an *ad hoc* way. This poor integration is also discussed in [97] and [26], which claim that the environment should be better supported by agent oriented programming languages and platforms, as well as playing a more relevant role in the development of multiagent systems from the first design steps.

As with the environment, many social and organisational aspects in agent communication, as well as the knowledge needed to support this social behavior (*social knowledge* [90]), are weakly supported by agent platforms and agent oriented programming languages [26]. Multiagent system developers usually design their systems by thinking in terms of high level interaction protocols and social abstractions, while most of the times they have traditional agent messaging as the only interaction mechanism available. As a result, these concepts are usually implemented by using basic, limited approaches such as massive broadcasting [39, 78, 83] or blackboard systems [55].

This lack of appropiate tools for the support of agents' environmental and social knowledge reduces the scalability and effectiveness of multiagent systems and thus, it

hinders their actual potential [26]. An additional mechanism, which agents could use in order to extract information from the running system and the environment, could be useful in order to overcome these limitations. In combination with traditional messages, this mechanism could be used as the necessary knowledge provider agents require in order to be able to properly interact with other agents and with the environment using high level social abstractions.

Information extraction from systems at run time has been used for decades in different areas of Computer Science, such as event-driven architectures [94] or real-time systems [44, 118]. This extraction can be carried out either during the development process, as a debug/validation mechanism, or after that, as a monitoring mechanism to extract information from systems at run time. When a system behavior needs to be observed, some run time information needs to be generated and retrieved. One of the ways in which this information may be obtained is *event tracing*. A *trace event* is a data object representing an action which is executed by either a running process or by the operating system. A brief survey of some approaches using trace analysis can be found in [116], which also presents a generic and extensible framework for the automatic extraction of temporal properties of real-time systems at run time.

Retrieving trace events from a running system requires the system to be instrumented at hardware or software level. This instrumentation implies modifying the system which is being studied and thus, it is necessary to take the degree of intrusiveness into account, in order to minimize the interference caused by the tracing process. The POSIX 1003.1-2001 standard [73] includes the definition of standard tracing services, so that tracing facilities are supported in a consistent, native way, from the operating system itself. As a result, once incorporated to a real-system, they become available for any running process, with minimal interference and overhead. Authors in [116] adapted the POSIX tracing standard to be suitable for small, embeded, real time systems and incorporated these facilities to the real time operating system RT-Linux [20], showing very low memory and computation time overhead in experimental results.

The author's research experience and results in the use of trace events to retrieve information from running real-time systems, showed that event tracing is an effective way to extract information from critical, embeded systems, in an efficient way, minimizing the computational intrusiveness and without interfering in their regular functionality. Some of this experience, the one published in [37], is included in this thesis as Chapter 2. As a result of this experience, this thesis proposes that an event tracing mechanism for multiagent systems, similar to the one proposed by POSIX for processes, could be used by running agents in order to retrieve information from their environment and from other running agents in the system. In a similar way to the POSIX proposal, this event tracing mechanism can be incorporated to the multiagent platform, so that event tracing facilities are offered by the platform itself minimizing the interference caused by the tracing process in the multiagent system.

Event tracing has been, and still is, widely used in the field of multiagent systems as a mechanism to extract information. In fact, it is so common to use event tracing to monitor multiagent systems, that many of the most popular multiagent platforms have developed their own tracing facilities, and have made them available for other developers or users. This is the case of JADE [21], JACK [4], Zeus [45] or Jason [24, 25], for example. Also, there are many tools developed by third party authors to trace different multiagent platforms, like JADEX [101], Java Sniffer [120] or ACLAnalyser [28, 29, 111] for JADE, MAMSY [109] for Magentix, the tool suite developed in [95] for Zeus.

In most of this work, however, there is a common factor: The information extracted from the system is mainly intended to be received and processed by a human observer, which uses this information for debugging or validation purposes, or in order to study and characterize the multiagent system. This thesis proposes that an event tracing mechanism for multiagent systems, similar to the one proposed by the POSIX standard, could be used, not only as a debug/validation tool, but also as a more appropiate environmental and social information provider for agents than *ad hoc* approaches. Such mechanism could be used as an indirect way of

communication, which agents could use together with traditional messages. As a result, more sophisticated ways of agent coordination would be available and high level social abstractions would also become more reliable, even from the first stages of multiagent systems design and development.

1.2 Objectives

This thesis proposes the use of a generic, event-trace based support for indirect communication in multiagent systems, which may be used by any entity in the system as a better social knowledge provider than traditional agent-to-agent messages. The term *entity* in this work comprehends agents in the system and also non-agent entities which may play a role in the multiagent system, like data bases or sensors. The final goal of this thesis is to prove that, by means of event tracing, entities in the system are able to improve the way in which they communicate, associate, coordinate and make use of some high level social abstractions, such as teams or virtual markets/organizations, in order to enhance their social potential.

This goal can be subdivided in the following, below detailed, sub-objectives:

- Review of the current state of the art about indirect communication in multiagent systems in order to identify advantages and lacks in previous approaches.
- Specification of a list of requirements that a generic event tracing support would have to meet in order to serve as an appropiate indirect communication mechanism for entities in a multiagent system.
- Development of an abstract model to incorporate a standarized trace-based support for indirect communication which covers the previously mentioned list of requirements.

- Development of a generic architectural design which allows the concepts in the trace model to be incorporated to a real multiagent platform, so that all entities in the system, both agent and non-agent entities, can benefit from event tracing as an indirect communication mechanism.
- Incorporation of the proposed model and architectural design into at least one existing multiagent platform.
- Validation of the developed model and architectural design by solving a case of study and analysing the obtained results.

1.3 Structure of the Thesis

Considering the motivation and objectives of this thesis, the rest of the document is structured as follows:

- **Part I. Introduction and Objectives:** In this part, the motivation and objectives of this thesis, as well as the structure of the document are presented.
- **Part II. Selected Papers:** This part presents a selection of the most representative articles supporting this thesis which were published in conferences and journals.
- **Part III. Discussion:** This part presents a final review and discussion of published work and results, as well as future directions for further research.

1.4 Publications List

In this section, all the international publications related to this thesis are listed. They have been classified according to their type (journals or international conferences) as

well as whether they are listed in JCR or in CORE, respectively. Those publications which have been included in this document are marked with **(*)**.

- Journals listed in JCR:
  - **(*)** L. A. Búrdalo, A. Terrasa, A. Espinosa and A. García-Fornes. *Analyzing the Effect of Gain Time on Soft Task Scheduling Policies in Real-Time Systems*. **IEEE Transactions on Software Engineering** Vol. 38 N. 6 pp. 1305-1318 . (2012) **Impact Factor: 2.588** · DOI: 10.1109/TSE.2011.95
    http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6025357
  - **(*)** L. A. Búrdalo, A. Terrasa, V. Julián and A. García-Fornes. *TRAMMAS: A tracing model for multiagent systems*. **Engineering Applications of Artificial Intelligence** Vol. 24 N. 7 pp. 1110-1119. (2011) **Impact Factor: 1.665** · DOI: 10.1016/j.engappai.2011.06.010
    http://www.sciencedirect.com/science/article/pii/S0952197611001102
  - **(*)** J. M. Alberola, L. A. Búrdalo, V. Julián, A. Terrasa and A. García-Fornes. *An Adaptive Framework for Monitoring Agent Organizations*. **Information Systems Frontiers** Volume 16 Issue 2, April 2014 Pages 239-256. Kluwer Academic Publishers Hingham, MA, USA. **Impact Factor: 1.077** · DOI: 10.1007/s10796-013-9478-x
    http://link.springer.com/article/10.1007%2Fs10796-013-9478-x
- Other international journals:
  - **(*)** L. A. Búrdalo, A. Terrasa, A. García-Fornes and A. Espinosa. *Supporting social knowledge in multiagent systems through event tracing*. **Journal of Physical Agents** Vol. 3 N. 3 pp. 19-24. (2009)
    http://www.jopha.net/index.php/jopha/article/viewArticle/55
- International conferences listed in CORE:

  - L. A. Búrdalo, A. Terrasa, A. García-Fornes and A. Espinosa. *Towards Providing Social Knowledge by Event Tracing in Multiagent Systems*. Hybrid Artificial Intelligence Systems, 4th International Conference, HAIS 2009, Salamanca, Spain, June. Lecture Notes in Computer Science, Volume 5572 484-491, 2009. DOI: 10.1007/978-3-642-02319-4_58. Print ISBN: 978-3-642-02318-7. Online ISBN: 978-3-642-02319-4. **CORE ERA2010 Rank: C**. *http://link.springer.com/chapter/10.1007%2F978-3-642-02319-4_58*

- Other international conferences:

  - L. A. Búrdalo, A. Terrasa, V. Julián and A. García-Fornes. *A Tracing System Architecture for Self-adaptive Multiagent Systems*. Advances in Practical Applications of Agents and Multiagent Systems, 8th International Conference on Practical Applications of Agents and Multiagent Systems, PAAMS 2010, Salamanca, Spain, 26-28 April 2010. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-12384-9_25. Print ISBN: 978-3-642-12383-2. Online ISBN: 978-3-642-12384-9. *http://link.springer.com/chapter/10.1007%2F978-3-642-12384-9_25*
  - (*) L. A. Búrdalo, A. Terrasa, V. Julián, C. Zato, S. Rodríguez, J. Bajo and J. M. Corchado. *Improving the Tracing System in PANGEA Using the TRAMMAS Model*. Advances in Artificial Intelligence – IBERAMIA 2012. Lecture Notes in Computer Science, At pp. 422-431, Volume: Volumen 7637. Springer Berlin Heidelberg. Print ISBN: 978-3-642-34653-8. Online ISBN: 978-3-642-34654-5. *http://link.springer.com/chapter/10.1007%2F978-3-642-34654-5_43*
  - J. M. Alberola, L. A. Búrdalo, V. Julián, A. Terrasa and A. García-Fornes. *Dynamic Monitoring for Adapting Agent Organizations*. Proceedings of the Third International Workshop on Infrastructures and Tools for Multiagent Systems – AAMAS 2012 . At page 121. Editorial Universitat Politècnica de València ISBN: 978-84-8363-850-7.

https://riunet.upv.es/handle/10251/16889

The publications selected to be included in the Part II of this document are the most relevant and closely related to this research work, regarding the objectives set in Section 1.2.

Chapter 2 (previously published in [37]) presents results of a study on the influence caused by the over estimation of worst case execution times of tasks in a real-time system. The paper also presents the framework developed by the atuhors to carry on the study. By means of even tracing, this framework allowed for the observation of different task sets executing over a real-time operating system. This observation was carried out at run time with a minimal interference in the regular execution of the tasks sets and published results contributed to show the effectiveness and efficiency of event tracing as a mechanism to extract information from critical systems, which made this mechanism a good candidate to support indirect communication in multiagent systems.

Chapter 3 (previously published in [34]) presents the list of functional, efficiency and security requirements a generic event tracing support has to meet in order to serve as an appropiate indirect communication mechanism for entities in a multiagent system. This list was developed attending to both the authors' previous experience in event tracing, and current, state-of-the-art research in the fields of Event Tracing and Indirect Communication in Multiagent Systems.

Chapter 4 (previously published in [36]) introduces TRAMMAS, a generic, abstract model for providing multiagent systems with an event tracing support. Together with the abstract model, a generic, service-oriented architecture is also presented. This architecture can be integrated within a generic multiagent platform to allow tracing entities in the TRAMMAS model to exchange trace events.

Chapter 5 (previously published in [38]) details the incorporation of the TRAMMAS model and architecture to the multiagent platform *PANGEA* [129]. The paper also includes a study on how the use of event tracing as an indirect communication

mechanism can reduce the amount of information transmitted and received by entities in the multiagent system when compared to the use of regular agent messages.

Chapter 6 (previously published in [13]) details the incorporation of an event tracing support based on the presented TRAMMAS model and architecture to the multiagent platform *Magentix2* [60, 115]. In this work, trace events are used as a way to dynamically detect situtations when a multiagent system has to change its objectives or behaviour in order to adapt to internal or environmental changes. The paper also includes a study on the benefits of using event tracing against traditional agent messages in order to detect local and global changes.

1.5 Research Projects

The research work presented in this PhD Thesis was carried out in the context of the following research projects:

- **Agreement Technologies**
  - Funder: Consolider Ingenio *CSD2007-00022*
  - Lead Applicant: Carles Sierra
  - Years: 2007 - 2012
- **Advances on Agreement Technologies for Computational Entities (ATforCE)**
  - Funder: Generalitat Valenciana, Consellería d'Educació, Direcció General de Política Científica *PROMETEO 2008/051*
  - Lead Applicant: Vicente Botti Navarro
  - Years: 2008 - 2011
- **MAGENTIX II: Una Plataforma para Sistemas Multiagente Abiertos**

  - Funder: Ministerio de Ciencia e Innovación *TIN2008-04446/TIN*
  - Lead Applicant: Ana M[a] García-Fornes
  - Years: 2009 - 2011

- **Organizaciones Virtuales Adaptativas: Arquitecturas y Métodos de Desarrollo**
  - Funder: Ministerio de Ciencia e Innovación *TIN2009-13839-C03-01*
  - Lead Applicant: Vicente J. Julián Inglada
  - Years: 2010 - 2012

- **PlanInteraction: Interacción Multi-Agente para Planificación**
  - Funder: *MICINN TIN2011-27652-C03-00*
  - Lead Applicant: Eva Onaindía de la Rivaherrera
  - Years: 2012 - 2014

- **iHAS: Sociedades Humano-Agente: Diseño, Formación y Coordinación**
  - Funder: Ministerio de Economía y Competitividad *TIN2012-36586-C03-01*
  - Lead Applicant: Vicente J. Julián Inglada
  - Years: 2013 - 2015

Part II

Selected Papers

CHAPTER

2

Analyzing the Effect of Gain Time on Soft Task Scheduling Policies in Real-Time Systems

AUTHORS:
LUIS BÚRDALO, ANDRÉS TERRASA, AGUSTÍN ESPINOSA AND ANA GARCÍA-FORNES
{lburdalo,aterrasa,aespinos,agarcia}@dsic.upv.es
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA
CNO/ DE VERA SN
46022 VALENCIA, SPAIN

Abstract

In hard real-time systems, *gain time* is defined as the difference between the worst-case execution time of a hard task and its actual processor consumption at run time. This paper presents the results of an empirical study about how the presence of a significant amount of gain time in a hard real-time system questions the advantages of using the most representative scheduling algorithms or policies for aperiodic or soft tasks in fixed-priority preemptive systems. The work presented here refines and complements many other studies in this research area, in which such policies have been introduced and compared. This work has been performed by using the authors' testing framework for soft scheduling policies, which produces actual, synthetic, randomly-generated applications, executes them in an instrumented real-time operating system, and finally processes this information to obtain several statistical outcomes. The results show that, in general, the presence of a significant amount of gain time reduces the performance benefit of the scheduling policies under study when compared to serving the soft tasks in *background*, which is considered the theoretical worst case. In some cases, this performance benefit is so small that the use of a specific scheduling policy for soft tasks is questionable.

2.1 Introduction

In the field of hard real-time systems, the main goal is to achieve that none of the so-called *hard tasks* in the system ever fails to meet its temporal requirements, usually defined in terms of deadlines. The current practice for achieving this goal is to adopt a certain scheduling paradigm in the development of the real-time system. The paradigm imposes both a particular task model at design time and a corresponding scheduling policy at run time, and then provides the system designer with a formal, *off-line* feasibility analysis by which it is possible to prove whether all hard tasks will be able to meet their deadlines before the system starts running. One of the

most sound and widespread paradigms is fixed-priority preemptive scheduling. In this paradigm, the task model requires each hard task to have some known temporal attributes (release times, computation times, deadlines, etc.) and a fixed priority. At run time, the system always selects the ready task with the highest priority for execution in a preemptive manner.

Hard real-time systems may also include some other tasks without hard or strict deadlines, which are normally referred to as *soft tasks*. The scheduling paradigm typically considers that the execution of a soft task produces some utility value to the system if the task can be completed before some point in time (related to the task's arrival time), after which this value progressively decreases; in contrast, the utility value of a hard task instantly drops to zero after reaching its deadline. Soft tasks are by definition not included in the off-line guarantee analysis, resulting in two main consequences. First, the system is not a priori committed to run them in a given time. And second, soft tasks are less restricted by the task model; in particular, their worst-case execution times or their exact arrival patterns do not need to be determined at design time. Thus, the general way to deal with soft tasks in hard real-time systems is to try to run them as soon as possible when they arrive to the system (thereby maximizing their utility), without compromising the deadlines of hard tasks. In systems following the fixed-priority preemptive paradigm, the trivial solution for this is to assign soft tasks a lower priority than any hard task, which relegates them to running in the background. In order to improve the poor quality of service obtained by this *background policy*, many authors have proposed specific scheduling algorithms or policies for soft tasks. These policies are normally run-time algorithms that work in a compatible way with the fixed-priority preemptive scheme by which hard tasks are dispatched.

The off-line feasibility (or *schedulability*) analysis is based on comparing the temporal requirements of each hard task against its theoretical worst-case running scenario. In order to do so, one of the input parameters of the analysis is the worst-case execution time (WCET) of each hard task. This is probably the most

difficult issue in the system design, since obtaining an accurate value of a task's WCET can be very complex, or even impossible, depending on the characteristics of both the task's code and the hardware on which the code is to be executed. An extensive study of different techniques and tools, as well as existing trends and open issues in the field of timing analysis in real-time systems can be found in [126].

Since an underestimated value of a hard task's WCET can make an apparently safe system crash at run time, the traditional approach has been to overestimate the WCETs of hard tasks. On the other hand, even if an accurate estimation of a task's WCET can be assumed, the worst-case behavior is actually very rare, since tasks do not always take the exact worst-case path within their code, and thus, it is often the case for tasks to consume only a fraction of this maximum time, as pointed out in [99]. Taking these two facts into account, it can be concluded that the usual case for hard tasks is to consume less processor time than their WCETs at run time, and often very significantly less. In real-time systems, the difference between a task's WCET and its actual processor consumption at run time is referred to as *gain time*.

When related to WCET overestimation, gain time has traditionally been considered as a design problem for hard tasks, but also as a benefit for soft tasks. The problem with WCET overestimation of hard tasks is that it restricts the ability of the system to be schedulable on a particular processor. This means that the system may be wrongly rejected by the off-line analysis, thereby forcing the system designer either to redesign the hard task set or to run the system in a faster (and more expensive) hardware than actually needed. However, from the viewpoint of soft tasks, gain time is considered an advantage, since it increases the expected amount of processor time available to their execution. In fact, some scheduling algorithms for soft tasks are designed to make an effective use of gain time, in order to further improve the running opportunities of soft tasks.

On the contrary, this paper presents the results of an empirical study about how the presence of a significant amount of gain time in a real-time system considerably *reduces* the advantages of using some of the most representative scheduling

policies for soft tasks in fixed-priority preemptive systems. This work refines and complements many other previous studies in this research area, in which these scheduling policies have been introduced and compared, usually in a theoretical way or by means of simulations. For this reason, the study has considered some of the usual assumptions in the previous work regarding the specification of the experiment load and the evaluation of the scheduling algorithms. In particular, the most important assumptions are the following: soft tasks are assumed to have no deadlines, soft tasks are dispatched in FIFO order, and the performance of the algorithms is measured by means of the average response time of soft tasks.

The study presented here has been carried out by using the authors' testing framework for soft scheduling policies. The framework first generates synthetic test programs and then runs each program on an instrumented operating system (a modified version of Open Real-Time Linux) that implements the scheduling policies under study. As a result of each execution, the framework automatically produces a complete set of statistical data about the performance of the scheduling policy. The framework has been carefully designed in order to make the results of the different scheduling policies comparable, which basically involves two main aspects. First, the policies themselves have been implemented in order to run the applications on equal terms; in particular, all policies have a compatible interface of system calls, which allow application tasks to have exactly the same code regardless of the policy running the application. And second, the results of each execution are processed in order to make all experiments comparable with each other. Furthermore, the combination of experiments with different *factors* (such as amount of hard tasks, hard task utilization, soft task utilization, etc.) can be used to determine to what extent each of these factors *individually* affects the behavior of the different scheduling policies.

The main results of this paper show that, in general, the fact that hard tasks consume less execution time than their estimated WCETs (which in turn produces the availability of gain time) negatively affects the performance *benefit* of using any of the policies under study with respect to scheduling soft tasks in background. This is also

true even for those policies that are specifically designed to efficiently reclaim and use gain time. In nearly all cases, this performance benefit is significantly reduced as the amount of gain time increases in the system. Under some conditions, this performance benefit is so small, or even negative, that the use of a specific scheduling policy for soft tasks becomes questionable. The final purpose of this work is for it to be used as a guide to determine which scheduling policies for soft tasks are more appropriate depending on the running conditions of the system and, specifically, the amount of gain time that is available at run time.

This paper is structured as follows: First, Section 2.2 describes the scheduling policies that take part in this work and some of the results obtained in previous comparative studies. Section 2.3 introduces the framework used to generate and run the experiments designed for this study, which are described in Section 2.4. The results of the experiments are presented in Section 2.5. Finally, Section 6.6 discusses the conclusions of the study.

2.2 Previous Work

2.2.1 Scheduling Policies

This study includes five of the most representative scheduling policies for aperiodic or soft tasks in fixed-priority preemptive real-time systems: *Deferrable Server* [86] (DS), *Sporadic Server* [17] (SS), *Extended Priority Exchange* [112] (EPE), *Dynamic Approximate Slack Stealing* [16, 50] (DASS), and *Dual Priorities* [48, 50] (DP). The execution of soft tasks in background, or *Background scheduling* (BG), is also included in the study as a lower bound in the performance of soft task scheduling.

Server-based scheduling policies are founded on the idea of reserving some execution bandwidth for soft tasks by means of adding a special task called “server” to the hard task set which in turn runs the soft tasks. The priority and temporal parameters of the server (period and computation time, also called *budget* or *capacity*) are adjusted

to off-line guarantee the entire task set. Both the Deferrable Server and the Sporadic Server work in a very similar way at run time. The main difference between them is the run-time strategy they use to replenish their budgets as soft tasks use them. This, in turn, limits the particular off-line equations that can be applied to analyze their schedulability. These conditions are more pessimistic for DS than for SS.

The Extended Priority Exchange algorithm uses a more complicated run-time strategy than the two previous algorithms. This strategy is based on the fact that there may be some available capacity for running soft tasks at each priority level as well as producing dynamic priority exchanges among tasks in order to preserve and use this capacity in an advantageous way for soft tasks. The initial capacity available at each priority level is computed off line in order to guarantee the schedulability of hard tasks, but it can be increased at run time if hard tasks consume less than their WCET.

Slack-based algorithms are based on delaying the execution of hard tasks in order to run soft tasks as soon as possible without missing any hard deadline. The family of slack scheduling algorithms includes some exact [85, 51, 103, 104] and approximate [50] versions. Among these algorithms, the Dynamic Approximate Slack Stealing algorithm is the only one that is feasible in practice, since the others present an excessive temporal or spatial overhead. The DASS is based on a fine-grained run-time supervision of the application tasks' execution, in order to keep track of the available *slack time* at each (hard task's) priority level. Then, soft tasks can safely run before hard tasks while there is slack time available in the system (that is, at all active hard priority levels) without compromising any hard deadline.

The Dual Priorities algorithm is based on assigning two priorities to each hard task, an *upper band* and a *lower band*, while soft tasks run in a *middle band*. The middle and lower bands have to be below the upper band of any hard task. At run time, every hard task starts its periodic activations in its lower band until a *promotion time* is reached; then, it runs the rest of the activation in its upper band. The system may assign any priority ranges to the middle and lower bands, as long as the hard task

set is schedulable in the upper band. Compared to DASS, the main benefit of the DP algorithm is that it needs very little run-time supervision by the system, since promotion times can be calculated off-line.

Because of the purpose of this study, it is important to note that for both the DASS and DP algorithms, some extensions have been developed in order to *reclaim* gain time as it becomes available at run time, and then to use this gain time to run soft tasks. These extensions, originally defined in [85, 50] (for slack-based algorithms) and [50, 48] (for DP), are referred to as *Propagated Gain Time* and *Self Gain Time* in this paper:

- *Propagated Gain Time*. Both DASS and DP admit an extension by which the available gain time of any hard task i (g_i) is computed every time it ends an activation (g_i is calculated by subtracting the actual computation time spent by the task in the activation from the task's WCET). By definition, this time may be used to run soft tasks at task i's or any *lower* priority level (hence the name *propagated*).

  The implementation of the propagated gain time extension is different for each algorithm. In particular, DASS with this extension adds g_i to the slack time available at task i's and lower priority levels, thereby increasing the amount of time that all (active) hard tasks may be safely delayed to run soft tasks. On the other hand, DP with this extension may delay for g_i time units the promotion times of hard tasks with priorities lower than i, thereby increasing the opportunities for running soft tasks (in their middle band).

- *Self Gain Time*. This extension is exclusive for the DP algorithm. In DP, the promotion time of each hard task is computed off-line in such a way that the task can safely run (for its entire WCET) *after* reaching this promotion time. Thus, if at the beginning of an activation, the task is allowed to run for some time in its lower band, then it is safe to delay its promotion for that amount of time in the current activation, potentially increasing the amount of time soft

tasks may be run in their middle-band priority.

2.2.2 Previous Comparative Studies

This section first presents the main conclusions of the simulation studies made by other authors. It must be noted that results from different simulation studies are difficult to compare because not all of them consider the same policies and they do not present comparable testing strategies. However, it is commonly accepted that the performance of the soft task scheduling policies is measured by means of the average response time of soft tasks, which are usually considered to have no deadlines and are served in FIFO order. The final part of the section concisely presents some general results derived from the authors' empirical testing framework, where all policies have been tested on equal terms.

In general, server-based policies improve the results obtained by scheduling soft tasks in background when the system's total utilization (including hard and soft tasks) is not too high; however, as the utilization of hard tasks grows, these policies tend to perform like background scheduling. When comparing the DS and SS policies, different studies do not come to the same conclusions. Studies in [65, 86, 17, 114] conclude that SS is better than DS because it allows for larger capacities and gets higher utilization values, while [66] shows larger response times for SS than for DS and. Finally, [22] concludes that both policies have similar response times and can get similar utilization values. Compared to servers, [112] shows that EPE obtains better results than DS when the hard utilization is high.

Slack-based policies are taken into account in several studies. When compared with the server-based and EPE policies [66, 62, 50, 49, 48], the main conclusion is that slack-based algorithms outperform all of them. However, some of these studies also state that the main drawback of slack-based algorithms is that most of them are not practicable due to their high overhead. In particular, the *Dynamic Slack Stealing* (DSS) policy is commonly used as a reference to compare other policies since it

has been proved to be optimal (see [51]) in the sense that it minimizes the response times of soft tasks without missing any hard deadlines (however, Tia et al. [119] showed some situations in which it is better not to use spare capacity immediately and therefore, optimality cannot be achieved). When compared with this optimal or *exact* version, the DASS exhibits a close performance with much less overhead. In particular, the study in [50] shows that the DASS presents a performance that is very near to DSS until the total load in the system (hard+soft+overhead) gets to 90%, at which point the system performance starts to degrade. The results in [62] show that DASS is very near to DSS in all cases, although in this study overhead is considered to be negligible.

The *Dual Priorities* scheduling policy (DP) is compared to other scheduling policies in [49, 48, 62, 64]. The results are very similar in all the studies: DP gets lower response times than BG, EPE and server-based policies. In fact, DP performs in a similar way to DASS when the system utilization is less than 90%, although response times are better for slack-stealing-based policies. [64] also shows that DP performs better than BG if and only if soft load is served in FIFO order.

Nearly all these studies do not consider the run-time overhead produced by the specific scheduling policies. Some studies do consider overhead, but in terms of theoretical worst-case costs (in orders of magnitude). The only studies in which run-time overhead is included are [66], which uses the number of context switches in the different simulations to approximate the total overhead produced by the scheduling policies, and [50], which includes extra CPU cycles in the simulations to approximate the cost of calculating the available amount of slack in the system. This study also includes the implementation of BG, DSS and DP in a real operating system and some overhead results, which show that DP presents a moderately higher amount of overhead than BG, while DSS incurs in such a great overhead penalty that makes it unfeasible in practice in systems with large numbers of hard tasks.

In contrast with these simulation studies, a general empirical study running real applications in a Real Time Operating System (RTOS) was presented in [33].

This study presented three general results: (1) in general, all the algorithms perform better than BG, even considering overhead; (2) all policies improve their performances compared to background as the *hard task utilization* grows; and (3), these performance improvements of all policies with respect to BG tend to disappear as the *number of hard tasks* increases, and the same happens as the *soft task utilization* grows. In addition, the study also presented some conclusions for each policy. The two server-based policies (DS and SS) perform much worse than DASS or DP, in spite of producing less overhead. In particular, differences in performance range from 15% in systems with low utilization up to 40% or more in heavy loaded systems. When compared, SS always performs better than DS, although the difference between the two policies is not very significant unless the system presents a high hard task utilization. This confirms the theoretical disadvantage of DS respect to SS about having a more restrictive feasibility test, which leads to lower server capacity and poorer performance. However, SS is more difficult to implement and produces more overhead than DS. Due to the overhead, the performance of the DASS policy is worse than expected (in the simulation studies) and most of the times it is outperformed by DP. This confirms the conclusions of [48]. However, DASS gets slightly better results than DP in heavy loaded systems, especially when the total utilization gets close to 100%.

Some of these scheduling policies have been subject to more recent studies in the field of multi-processor systems. For example, SS has been adapted and optimized to be effectively used in multi-processor systems [59]; DS has been shown to improve the performance of soft tasks when compared to BG in asymmetric multi-processor systems [40]; and both an optimal slack-based policy and DP have been used in order to globally allocate soft tasks among processors [18, 19], with DP outperforming the slack policy in heavy loaded systems.

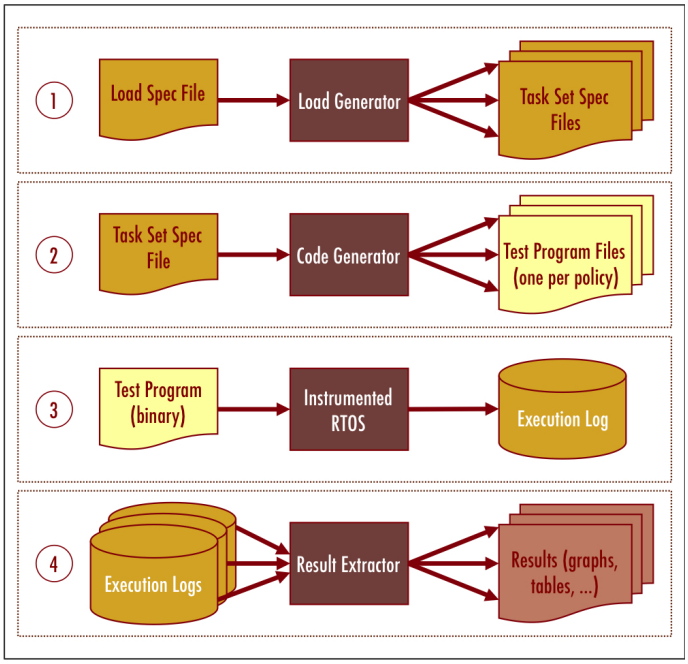


Figure 2.1: The Testing Framework

2.3 The Testing Framework

This section summarizes the framework used to generate and run the experiments described later in the paper. As depicted in Figure 2.1, the framework is basically composed by four modules: Load Generator, Code Generator, the instrumented RTOS, and Result Extractor. These modules are now described, placing special emphasis on the main design ideas that support the validity of the results presented in the paper.

2.3.1 The Load Generator Module

The framework can be configured to generate tests for many different scenarios, depending on the particular goals of the experiment. This configuration is mainly

carried out in the input file of the Load Generator module, or *load specification file*. According to the specification included in this file, the module generates the experiment's set of *task set specification* (TSS) files as an output. The load specification file contains the desired values of the parameters that the load generator will combine in order to create the task sets, as well as the number of task sets to generate for each parameter combination (or number of *replicas*). The main parameters that can be specified in the file include the number of hard and soft tasks, hard tasks utilization, soft tasks utilization, the priority levels for soft tasks, maximum hyperperiod, and hard tasks gain time. The Load Generator considers all possible combinations of the input parameters and then, for *each* combination, it generates as many task sets (TSS files) as the number of specified replicas. As a result, each TSS file contains a complete specification of a task set, including the number of tasks, their types (hard or soft) and their attributes (execution times, budgets, deadlines, periods, priority bands, etc.).

Each task set is randomly generated within the limits of its corresponding combination of input parameters, with two main restrictions: the set of hard tasks has to be schedulable, and the budgets assigned to soft tasks in server-based policies are maximized (while keeping the hard tasks schedulable). In particular, for each task set to generate, the Load Generator follows this procedure:

1. The period of each task (T_i) is randomly generated, according to three input parameters: the maximum hyperperiod of the task set, the type of random distribution to use (uniform or exponential) and the range of this distribution (maximum and minimum values). The set of generated periods is accepted if the resulting hyperperiod is not greater than the maximum specified value; otherwise, the process starts again.

2. According to the input specification, tasks are separated into two groups: hard and soft. Then, in each group, the individual utilization of each task (U_i) is computed by using the Uunifast algorithm [23] (which has been shown to

produce unbiased random task sets) according to two input parameters per group: the number of tasks and the utilization.

3. The WCET of each task (C_i) is calculated as follows: $C_i = T_i U_i$. The input specification may establish a minimum WCET; if so, the task set is discarded if any C_i is below this value (and then the procedure returns to the first step).

4. The deadline for each task (D_i) is generated between C_i and T_i by using a uniform distribution.

5. The group of hard tasks is reordered by deadline (lowest deadline first). Then, hard tasks are assigned priorities following a deadline monotonic policy (the lower the deadline, the higher the priority). At run time, the actual priority of each task will depend on the particular scheduling policy, but in any case, the relative priority among hard tasks is maintained according to this initial assignment.

6. For each hard task i, its computation time ($Comp_i$) is calculated as a function of its WCET and the gain time percentage (G) specified in the input file:

$$Comp_i = \frac{(100 - G) * C_i}{100} \tag{2.1}$$

7. The budgets for the servers in DS and SS policies are calculated as the maximum values that keep the hard task set schedulable according to the exact schedulability test (based on the maximum response time of each task) required by each server policy. If the hard task set is not schedulable for any of the two tests, then the task set is discarded and the procedure returns to the first step.

8. The initial parameters of some policies, such as the promotion times of hard tasks in the dual-related policies or the initial aperiodic time available for hard tasks in the EPE algorithm, are calculated.

9. In order to better approach the usual run-time behavior of tasks, and also to ensure that different policies running the same task set will face exactly the

same load, the framework also generates a list of *activations* per task. In the list, each activation corresponds to a task's release and it stores the run-time parameters that may vary in different releases of the same task. In particular, for each activation k of any task i, two values are calculated:

(a) The actual computation time of the release ($Comp_i[k]$). This value is computed by using a normal distribution with a mean equal to $Comp_i$ and a standard deviation value defined by the input specification.

(b) The time of the next release ($T_i[k]$). If task i is periodic, this value is always set to T_i. Otherwise, if task i is aperiodic, this value is randomly generated by using a uniform distribution with a range centered on T_i plus-minus some percentage value defined by the input specification.

After this procedure successfully generates a schedulable task set, all its relevant temporal parameters are written in the corresponding TSS file.

2.3.2 The Code Generator Module

The Code Generator module is responsible for producing actual *test programs* for each TSS file (task set), specifically, one test program per soft scheduling policy to be tested. Test programs are C source files that contain synthetic code generated for a particular TSS file and scheduling policy, in a compatible way to the system interface of the instrumented RTOS on which tests will be run, which is compatible with the POSIX standard. Each task set requires the generation of a different test program per policy because in POSIX, the selection of the scheduling parameters (priority and policy) for each task is performed by the application at run time.

The generation of test programs has been designed to make the running conditions of each test program both as close to its specification (TSS file) as possible, and as similar for all policies as possible, so that results are not biased for any particular policy. In order to achieve these goals, the test programs in the group corresponding

to the *same* TSS file (one program per policy) are generated to meet the following four requirements:

- In every program in the group, execution starts at a *critical instant*, in which all tasks are simultaneously released. This is the worst scenario for soft tasks, but it is also a straightforward way of making all scheduling policies start on equal terms.
- In every program in the group, any given task i is generated in order to follow the sequence of release times ($T_i[k]$) specified in the TSS file. This ensures that the release pattern of each task will be the same for every policy.
- In every program in the group, any given task i is generated to have exactly the same code. This is possible because the policies under study have been designed with a compatible set of system calls, and the selection of the particular policy is performed at run time.

  For each task i, the code is generated in order to spend a computation time equal to $Comp_i[k]$ for each release k, as specified in the TSS file. In order to do so, the framework incorporates a calibration mechanism that adjusts the generation of code to the expected computation time for a particular processor. In practice, this calibration mechanism achieves actual execution times to be between 90% and 100% of the expected ones.
- In every program in the group, hard tasks are released for two consecutive hyperperiods, while soft tasks are released during the first hyperperiod only. This is further discussed below.

In the simulation studies mentioned in Section 2.2, each experiment is carried out by executing the simulated workload for a given period of time, which is normally the hyperperiod of the experiment's hard task set. The rationale for this is that the release pattern of the hard task set is repeated after each hyperperiod, and the same is true for the running conditions for the soft workload. However, with this approach,

it may happen that some soft workload is pending (i.e., awaiting execution) at the end of the first hyperperiod, especially in heavily loaded systems. Furthermore, for any given task set, the amount of pending soft workload at the first hyperperiod may be notably different from one scheduling policy to another. The problem with this is that the measurements corresponding to pending soft workload are not included in the experiment's results.

In the framework described in this paper, the solution for this is to generate the soft workload during the first hyperperiod, in such a way that the specified soft utilization is met, but to run the experiment until there is no pending soft workload, even if this happens after reaching the first hyperperiod. In this way, for any given task set, the running conditions are equal to all the scheduling policies, and the results of each policy always include the entire soft workload. However, if any two policies running the same task set finish at different times, the results related to the system *overhead* are not directly comparable. Taking both restrictions into account, the strategy adopted by the framework is to generate the soft workload during the first hyperperiod of each experiment, but to run the experiment until exactly its second hyperperiod, in order to guarantee that both the performance results are complete for all policies and that their overhead results are comparable. This solution is valid as long as the experiment design (in particular, the sum of hard and soft utilizations, plus the overhead) allows the entire soft workload to be completed before the second hyperperiod.

2.3.3 The Instrumented RTOS

The instrumented Real-Time Operating System (RTOS) on which test programs are run is a modified version of Open Real-Time Linux [127], which is a small, hard real-time executive running under Linux. From the application's perspective, Open RT-Linux provides a programming interface that is compatible with the POSIX standard [73]. Internally, this RTOS has a simple run-time behavior, in which the system deals with each hardware interruption or system call invocation by means

of a specific function and then, in all cases, the same scheduling function is called; this function is the one that is responsible for selecting the new task to be run, and then dispatching it, which provokes a context switch if the selected task is different from the running one. The original scheduling policy that Open RT-Linux applies by default for all tasks is fixed-priority preemptive (that is, POSIX's "SCHED_FIFO"), which has been used by the framework as the background (BG) policy. Because of its simple design and small size, Open RT-Linux produces very low and predictable overhead at run time. According to the experiments presented in this paper, carried out on a Pentium III 700Mhz computer, the average cost of the aforementioned scheduling function in RT-Linux ranges from 4.5μs (4 tasks) to 12μs (16 tasks), while the maximum cost ranges from 12μs (4 tasks) to 30μs (16 tasks).

The framework has extended Open RT-Linux in two main ways: the implementation of the soft scheduling policies under study and the incorporation of a tracing mechanism by which it is possible to collect run-time information as applications are executed.

The soft scheduling policies under study have been implemented and provided as new scheduling choices for soft tasks at run time. To do this, some of these policies had to be redesigned in order to be provided with POSIX-like interfaces (as presented in [32]). The framework provides a specialized version of the system scheduler module for each policy, in order to avoid the implementation of any given policy to affect any other (in terms of overhead).

The tracing mechanism introduced to the RTOS includes a general, POSIX-like tracing system, and a particular instrumentation of the RTOS code which traces some system events (e.g., scheduling decisions, context switches, costs of the scheduler's functions, etc.) in order to analyze the behavior of the different scheduling policies. As a result, the execution of any real-time program on the instrumented RTOS can automatically produce a log file containing the events traced inside the RTOS during this execution. Both the instrumentation and the collection of events have been designed to make the overhead related to event tracing predictable and equivalent

for all policies (the average cost of tracing each event is under 500 ns in a 700Mhz Pentium III processor, as shown in [117]). In particular, the set of system events and their instrumentation points are the same for all policies; at run time, events are traced to memory during the program execution, being dumped to the log file only after the application tasks have stopped running.

2.3.4 The Result Extractor Module

The Result Extractor module is responsible for extracting useful information from the logs generated by the instrumented RTOS when it runs the test programs. In particular, the extractor module works in three steps, which are presented below.

In the first step, the extractor module opens the log corresponding to an individual test program execution and then traverses it in order to calculate some predefined *metrics*. A metric is defined as a property of a program's execution. Examples of metrics are some temporal properties of application tasks (response times, execution times), costs of RTOS functions, number of context switches, etc. The output of this step is a basic statistical analysis of each metric (average, maximum, minimum and standard deviation). Among all the metrics, some of them are selected to be the *relevant results* of the tests (for example, the average response time of soft tasks), and then they are further processed by the module.

In the second step, the module combines the relevant results of all the executions corresponding to the *same* task set (one per policy). For each result, the module calculates the result *ratio* for each policy as the division of the policy's result value and the corresponding value obtained by the reference policy, which is BG. This ratio thus represents the result *difference* of using this scheduling policy with respect to using the BG policy for this particular task set.

In the third step, after the relevant result ratios have been computed for each task set, the extractor module combines the result ratios corresponding to *different* task sets in the experiment. The corresponding result ratios of different task sets are

directly comparable with each other because ratios express the relative differences of results between a particular policy and the BG policy in each task set. For each relevant result, this third and final step obtains two types of outcomes: a *global value* per policy, expressing the average ratio for all the task sets in the experiment, and the *variation* of the result ratio for each policy as a function of the different input parameters of the experiment (or *factors*), such as the soft load utilization or the amount of gain time, for example. This variation is computed by grouping the result ratios of all tests according to the different values of this factor, and then calculating the average ratio value for each group. For example, if an experiment comprised task sets with four different values of gain time (0%, 25%, 50% and 75%), one possible outcome would be the variation of the soft task response time of each policy as a function of gain time. For each policy, the module would obtain four values: it would first classify all the experiment results according to the gain time (in four groups), and then, for each group, it would compute the average of the soft task response time ratios obtained by this policy in all the experiments in the group. These four values would show the variation (or evolution) of the result ratio for this policy as the factor varies.

2.4 Experiment Design

The main goal of this study is to determine to what extent the presence of gain time in real-time systems with hard and soft tasks influences the performance of the most representative scheduling algorithms for soft tasks in fixed-priority preemptive real-time systems. In order to better describe the experiments and their results, the following two concepts related to the hard task set in the system are defined: the *nominal* hard utilization, or nominal hard load, is the theoretical utilization of the hard task set, derived from the WCET values established at the schedulability analysis. On the other hand, the *real* hard utilization is the actual utilization of the hard task set at run time, derived from the real execution time consumed by hard tasks as the system

runs. Thus, the difference between the nominal and the real utilization in a given task set will determine the amount of gain time that will be available for soft tasks at run time. Please note that according to the framework described above, the experiment sets as input parameters both the percentages of nominal hard load and gain time, and then the real hard utilization is derived from them.

The main decision about the experiment design was to determine the number of experiments to be carried out and to select the amount of nominal and real hard utilization in each experiment. Some preliminary experiments with the same framework proved that differences in performance between systems with and without gain time increased along with the nominal hard utilization for all the scheduling policies. For this reason, a total of four experiments were designed, with each of them fixing a particular value of *nominal* hard utilization: 20%, 40%, 60%, and 80%. Then, for each experiment, the framework generated four series of task sets with different percentage values of gain time (0%, 25%, 50%, and 75%), thereby producing different values of *real* hard utilization. Because of size limitations, this paper presents the results of the two experiments that rendered the most significant results: the ones with 40% and 80% of nominal hard utilization[1]. The rest of this section presents the parameter configuration of both experiments in full detail (summarized in Table 2.1).

The next design decision was to determine which input specification parameters to fix and which others to vary in order to generate the task sets for each experiment. In particular, periods of hard tasks were generated by following a uniform distribution between 50 and 2000 milliseconds, with a maximum hyperperiod of 10 seconds. This distribution, along with its limits, were chosen in order to produce task sets that are comparable with previous studies in the literature, such as [50, 66, 62] (some of these studies concluded that choosing a uniform or exponential distribution did not

[1]Gain time had little effect on the performances of the scheduling algorithms in the experiment of 20% of nominal hard load, since the absolute amount of gain time was very small in this case, while the effect in the experiment of 60% of nominal hard load was intermediate between the results of the two experiments presented in the paper.

Table 2.1: Parameter summary of the experiments

Experiment 1: 40% *Nominal* hard load	
# hard tasks	4, 8, 12, 16
Policies	BG, DS, SS, EPE, DASS, DASS-GAIN, DP, DP-GSELF, DP-GPROP, DP-GBOTH
% Gain time	Soft task utilization
0	10% to 60% (in steps of 10%)
25	10% to 70% (in steps of 10%)
50	10% to 80% (in steps of 10%)
75	10% to 90% (in steps of 10%)
# replicas	50

Experiment 2: 80% *Nominal* hard load	
# hard tasks	4, 8, 12, 16
Policies	BG, DS, SS, EPE, DASS, DASS-GAIN, DP, DP-GSELF, DP-GPROP, DP-GBOTH
% Gain time	Soft task utilization
0	10% to 20% (in steps of 10%)
25	10% to 40% (in steps of 10%)
50	10% to 60% (in steps of 10%)
75	10% to 80% (in steps of 10%)
# replicas	50

affect the results). On the other hand, the rest of the parameters in the specification were varied within certain limits in each experiment, in order to be able to study the influence of these parameters on the performances of the policies under study. In particular, for each experiment, the framework generated task sets for all the combinations of the following varying parameters: number of hard tasks (4, 8, 12, and 16), percentage of gain time (0%, 25%, 50%, and 75%) and soft load utilization (from 10% up to achieving 100% of total *real* utilization, in increments of 10%). For each possible combination of all input parameters, 50 replicas (different task sets) were generated. In the generation of each task set, the run-time variability parameters of hard and soft tasks were set in the following manner: the computation

time of each (hard or soft) task was varied in an interval of $[-10\%, 10\%]$ of the task's real computation time (using a normal distribution), while the arrival pattern of each soft task was varied within an interval of $[-10\%, 10\%]$ of the task's period (using a uniform distribution). Please note that, as explained in Section 2.3, the framework always generates task sets in such a way that (1) hard task are schedulable according to the feasibility analysis, and (2) the budgets for the server (soft) tasks in server-based policies are as large as possible.

By combining all these different specification parameters, the total number of task sets for the experiments of 40% and 80% of nominal hard utilization were 6000 and 4000, respectively. For each task set, the framework generated a series of test programs, one for each of the ten scheduling policies considered in this study: Background (BG), Deferrable Server (DS), Sporadic Server (SS), Extended Priority Exchange (EPE), Dynamic Approximate Slack Scheduling (DASS), DASS with the propagated gain time extension (DASS-GAIN), Dual Priorities (DP), DP with the propagated gain time extension (DP-GPROP), DP with the self gain time extension (DP-GSELF) and DP with both types of gain time extensions (DP-GBOTH). As a result, the total number of test programs generated, compiled and executed for the two experiments were 60000 and 40000, respectively. All programs were run on a Pentium III 700Mhz computer, with 384Mb of RAM.

In every task set, the soft load was modeled (and generated) as a single aperiodic task configured to have the best running opportunities according to each scheduling policy under study: in BG, the soft task is always scheduled at the lowest priority. In SS and DS, the soft task is scheduled at the highest priority as long as there is some budget left, and it is otherwise relegated to running in the background. In EPE, the DASS-related and the DP-related policies, the soft task is always scheduled in a middle-band priority, while hard tasks start their activations in their lower-band priorities (where the soft task can preempt them) and then they may change to their upper-band priorities if certain running conditions are reached (in particular, the available capacity is exhausted in EPE, or the available slack is exhausted in DASS,

or the tasks' promotion times are reached in DP). According to some studies, this configuration may not be optimal in the case of DS and SS (in [22] it is shown that, by assigning the soft task the highest priority, the budget adjustment in server-based policies may not be optimal if task deadlines are lower than their periods). However, the selection of the optimal server parameters (budget, period and priority) is still an open research issue. For this reason, the experiments were set to schedule servers at the maximum priority, which is the most common approach in the literature.

In order to be able to compare the results with previous studies, the performances of the scheduling polices were measured by means of the average response time of soft tasks. In addition, the experiments also measured the overhead of the scheduling algorithms, in order to globally quantify the extra cost incurred by each algorithm and to relate it to its performance, if possible. In particular, the experiments measured two overhead indicators in each test: the number of context switches and the total scheduling time spent by the RTOS (the cumulated cost of the scheduling function inside the RTOS for the duration of each test). In this context, it has to be noted that the potential effect of the overhead on the performance of a scheduling policy depends on the proportion between the overhead values and the computation times of the application tasks (the effect increases as task computation times get closer to the scheduling costs). Taking this into account, the experiments were configured in order to generate task sets with task computation times in a reasonable range when compared to the average scheduling overhead of the reference policy (BG) measured on the same testing hardware. In particular, the experiments were configured for this *base overhead* to be around 10% of the total execution time, which is considered to be enough to influence the policy performance, but still within a reasonable range when compared to the overhead in real systems.

2.5 Results

The experiments considered three *relevant results* for each scheduling policy in each test: the soft task response time, the number of context switches, and the total scheduling cost, with the first one measuring the performance and the last two measuring the overhead of the scheduling algorithms. As explained in Section 2.3.4, these relevant results are expressed in relative terms (ratio values) with respect to the respective results obtained by BG, which is the reference policy. Thus, a ratio value of 1.0 expresses a result that is equal to the one obtained by BG, a ratio value of 0.9 expresses a result that is 10% lower than the result obtained by BG, and so on.

The following subsections analyze the results obtained in the two experiments: 40% and 80% of nominal hard utilization, named *Experiment 1* and *Experiment 2*.

2.5.1 Experiment 1: 40% of Nominal Hard Utilization

The global performance results of the experiment for each scheduling policy are presented in Table 2.2. This table shows the performance difference between each policy and the BG policy by means of a set of percentile values of the soft task response time ratios for all the 6000 task sets in the experiment (hence including all combinations of the varying parameters in the experiment: number of hard tasks, gain time, and soft load).

According to the values in the table, both DS and SS get better results than BG in a very low number of task sets only (the 5^{th} percentile values are 0.88 in DS and 0.85 in SS, meaning that in 5% of the task sets, these algorithms get 12% and 15% of improvement in the soft task response time over BG). However, all values from the 25^{th} percentile on are higher than 1.0 for both policies, which means that in at least 75% of the cases, they present slightly *negative* benefits when compared to BG, due to their extra overhead. DASS-related and DP-related policies perform better than BG in all cases, but only significantly better in a reduced number of task sets

Table 2.2: Soft task response time ratios in Experiment 1 (percentile values)

	Percentiles				
Policy	5%	25%	50%	75%	95%
DS	0,8850	1,0007	1,0026	1,0058	1,0135
SS	0,8590	1,0011	1,0030	1,0061	1,0126
EPE	0,6721	0,9035	0,9692	0,9957	1,0073
DASS	0,5149	0,7568	0,8876	0,9594	0,9967
DASS-GAIN	0,5129	0,7403	0,8661	0,9384	0,9867
DP	0,5575	0,7837	0,8947	0,9551	0,9920
DP-GSELF	0,5416	0,7822	0,8928	0,9545	0,9919
DP-GPROP	0,5570	0,7845	0,8931	0,9552	0,9924
DP-GBOTH	0,5429	0,7822	0,8924	0,9540	0,9919

(their improvement over BG is around 25% in their 25^{th} percentile, but this figure is reduced to 5% of improvement in their 75^{th} percentile). In addition, the ratio values show almost no difference among these six policies, which implies that the gain time reclaiming extensions of DASS or DP do not improve the results obtained by these two algorithms in the experiment. Finally, the ratios of EPE show intermediate results between DS/SS and DASS/DP-related policies. The improvement of EPE over BG is 10% or higher in 25% of cases, it is negligible in at least 50% of the cases, and it is negative in at least 5% of cases, again due to its extra overhead.

The performance difference between each specific policy and BG can be further analyzed by considering the effect of gain time. In Figure 2.2, the average ratio of the soft task response time of each policy is represented as a function of the available gain time. For each policy, the graph presents the average of the ratio values obtained by the policy in all the task sets with a particular amount of gain time. As can be observed, the ratio values for *all* policies become closer to 1 as the amount of gain time grows, meaning that the performance difference between each policy and BG is reduced as the amount of available gain time increases. This effect is more acute in DASS-related and DP-related policies, since they obtain much better results (between 30% to 35% of benefit) in systems with no gain time than in systems with 75% of gain time (where the benefit is only around 5%).

Finally, Figures 2.3, 2.4, 2.5 and 2.6 show the combined effect of gain time with the increase of the amount of soft load in the system. This is done by showing four graphs, each one depicting the average values of the soft task response time ratios as a function of the amount of soft load, in task sets with a specific amount of gain time (graphs correspond to tests where hard tasks had 0%, 25%, 50%, and 75% of gain time, respectively). Please note that the number of points for every policy in each graph is different because task sets were produced in such a way that the soft task utilization was generated in increments of 10% until a 100% of total (hard plus soft) *real* utilization was reached. There are three relevant aspects to be pointed out about these graphs. First, in the four graphs, the results classify policies in three groups (the two server-based policies, EPE, and the DASS-related and DP-related policies) with little difference among the policies in each group. This is consistent with the conclusions derived from both Table 2.2 and Figure 2.2. Second, considering each graph separately, the performance benefits of all policies with respect to BG are reduced as the amount of soft load increases in the system (this confirms the conclusions of [33], independently of the amount of gain time available in the system). And third, considering the four graphs together, the performance benefits of *all* policies with respect to BG are reduced, some of them severely, as the amount of gain time increases in the system. In the last graph (75% of gain time), there is practically no benefit in using any of the policies, especially for a high amount of soft load.

Regarding the overhead results of the experiment, Table 2.3 displays the global values, in terms of the *increment* percentage with respect to BG of two average ratios: the total number of context switches (second column) and the total scheduling cost of each execution (third column). For each increment value, the number in parenthesis expresses its standard deviation. The context switch values in the table show that, except for the EPE algorithm (with 8% of increment), there is a small general penalty in the number of context switches for using specific policies for soft tasks rather than using BG (less than 3% in all these policies). The DASS algorithm even presents a negative value, meaning that this policy actually produces fewer context switches (on

Table 2.3: Average overhead ratios (and std. dev.) in Experiment 1 (values in % of increment with respect to BG)

Policy	# C. Switch	Total sched. Cost
DS	+0,90 (3,24)	+14,36 (13,25)
SS	+1,42 (4,42)	+18,58 (15,23)
EPE	+8,35 (6,04)	+53,36 (25,76)
DASS	-0,79 (3,06)	+43,15 (23,42)
DASS-GAIN	+2,74 (4,99)	+43,99 (24,35)
DP	+2,37 (3,95)	+24,82 (17,04)
DP-GSELF	+2,34 (3,92)	+22,06 (16,45)
DP-GPROP	+2,38 (3,97)	+25,96 (17,41)
DP-GBOTH	+2,34 (3,94)	+24,98 (17,19)

average) than BG. On the other hand, considering the total scheduling cost of each test, it is clear that there is a significant penalty for using specific policies with respect to using BG, especially in some of the policies. In particular, the extra overhead is considerably higher than BG in DASS-related policies and EPE.

2.5.2 Experiment 2: 80% of Nominal Hard Utilization

The global performance results of the experiment are shown in Table 2.4. This table shows the performance difference between each policy and the BG policy by means of a set of percentile values of the soft task response time ratios for all the 4000 task sets in the experiment (including all combinations of the varying parameters).

Comparing the data in this table with the global results of Experiment 1 (in Table 2.2), the policies in this second experiment present the following performances: the two server-based policies again present the worst results, only slightly better than in the previous experiment (both policies now perform better than BG in at least 25% of the cases). EPE now performs considerably better than BG in a large number of cases (20% better in 50% of the cases, and 40% better in half of them); in fact, EPE now show ratios that are similar to DASS, or even slightly better (from the 75^{th} percentile on). In this experiment, there is a great difference between the two versions of DASS.

Table 2.4: Soft task response time ratios in Experiment 2 (percentile values)

	Percentiles				
Policy	5%	25%	50%	75%	95%
DS	0,5672	0,9815	1,0028	1,0079	1,0196
SS	0,3549	0,9088	1,0007	1,0064	1,0162
EPE	0,2022	0,5872	0,8135	0,9318	1,0016
DASS	0,0761	0,5321	0,8137	0,9495	1,0018
DASS-GAIN	0,0632	0,2427	0,5147	0,7538	0,9068
DP	0,2076	0,4291	0,6334	0,8032	0,9543
DP-GSELF	0,1539	0,3674	0,6000	0,7958	0,9543
DP-GPROP	0,2083	0,4285	0,6333	0,8022	0,9535
DP-GBOTH	0,1539	0,3644	0,5983	0,7930	0,9532

For every percentile rank shown in the table, the ratio value of DASS is notably higher than the value of DASS-GAIN, meaning a better performance for the latter. Globally, DASS-GAIN obtains the best performance results in this experiment, while the performance of DASS is worse than all the dual-based policies, and sometimes worse than EPE. Finally, the performances of the four DP-related policies are quite homogeneous, with DP-GSELF and DP-GBOTH only moderately improving the results of the other two, and all of them being intermediate between DASS-GAIN and DASS. If compared with the previous experiment, the performances of all DP-related algorithms are now much better in all percentile ranks.

These global results are now refined by introducing the effect of gain time on the soft task response ratios, as shown in Figure 2.7. This graph again shows that gain time poses a negative effect on the performance benefit of all policies with respect to BG. In fact, comparing this graph with the one for the previous experiment, the effect is now more severe for all policies. This general trend presents an exception, the EPE policy in systems with no gain time, which is further discussed below. When looking at specific policies, some relevant aspects may be pointed out: SS now outperforms DS, especially in systems with no gain time. EPE gets better performance than DASS in systems with large amounts of gain time (50% or higher). The effect of gain time on DASS is critical, where the performance benefit ranges from around 85%

in systems without gain time to less than 10% in systems with a 75% of gain time); this effect is also evident in DASS-GAIN, but this latter policy only degrades up to an average benefit of 25% due to its ability to effectively use gain time for executing soft tasks. Finally, DP-GSELF and DP-GBOTH outperform both DP and DP-GPROP in systems with a low amount of gain time, but this difference tends to disappear as gain time augments.

The special case of the EPE policy is now discussed. According to the graph, EPE performs worse in systems without gain time than in systems where there is some gain time available (up to 50% of gain time). The reason for this can be related to (1) the extra overhead of this algorithm and (2) the inefficient way in which the algorithm computes off-line the initial aperiodic time available for hard tasks (which is more evident in this second experiment, where task sets have a high nominal hard utilization). The algorithm is designed to increase these aperiodic time values at run time by reclaiming gain time; however, in systems with no available gain time, EPE cannot compensate its poor initial configuration, and thus its performance gets closer to BG. Excluding this particular case, the effect of gain time on the EPE algorithm time shows the same trend than on any other policy.

Finally, Figures 2.8, 2.9, 2.10 and 2.11 show the influence of the soft task utilization in the performance ratios of all policies in each of the four possible gain time values of the experiment. The conclusions that can be drawn from these graphs are similar to the ones in the previous experiment and are consistent with the general performance results of this experiment presented above: first, in systems with any particular value of gain time, the average performance benefits of all policies versus BG decrease as the amount of soft utilization grows. Second, for any given value of soft utilization, all policies exhibit less performance benefit with respect to BG as the amount of gain time increases in the system (except for the case of EPE in systems with no gain time, as discussed above). And third, all policies perform better now than in the previous experiment, for any given value of gain time. In this second experiment, there is a clear advantage of using some specific policies for soft tasks with respect to using

BG, even in systems with a high percentage of gain time. Among such policies, the best results are rendered by DASS-GAIN and the four DP-related policies.

Table 2.5: Average overhead ratios (and std. dev.) in Experiment 2 (in % of increment with respect to BG)

Policy	# C. Switch	Total sched. cost
DS	+4,48 (7,44)	+14,23 (12,98)
SS	+6,01 (9,75)	+18,05 (14,69)
EPE	+11,86 (8,02)	+51,04 (26,69)
DASS	+1,20 (5,85)	+43,09 (23,41)
DASS-GAIN	+4,14 (8,10)	+44,74 (24,79)
DP	+2,49 (5,40)	+23,32 (16,57)
DP-GSELF	+2,53 (5,53)	+21,84 (16,84)
DP-GPROP	+2,48 (5,41)	+23,52 (16,31)
DP-GBOTH	+2,52 (5,56)	+23,88 (17,51)

Table 2.5 presents the global overhead results for this experiment. When compared with the previous experiment, the values for the total scheduling costs are similar for each policy, while the values for the number of context switches are now higher in all policies except the DP-related ones. In particular, server-based policies have increased their context switch penalties with respect to BG around 4% (due to a higher number of times in which their budgets run out), the DASS-GAIN exhibits an increment of around 2% (due to the presence of a greater amount of absolute gain time, which allows more tasks to be run within the intervals of reclaimed gain time), and the EPE algorithm has incremented its penalty from around 8% to almost 12% (due to both more gain time available and a mechanism of reclaiming and using this time less efficiently than other algorithms, such as DASS-GAIN).

When analyzing the global overhead ratios as a function of the experiment parameters, the one with the greatest influence was, as expected, the number of hard tasks (since all the scheduling policies are based on certain computations to be performed over the entire list of hard tasks). In order to better show this, this second experiment was extended to incorporate task sets with more hard tasks (up to 32). The results are presented in Figure 2.12. The graph shows that the number of

hard tasks in the system produces a linear increment of the total overhead ratio with respect to BG in all policies, but this effect is more pronounced in some policies than in others. In particular, three different groups of policies can be observed: (1) the overhead values of DS and SS with respect to BG are barely affected; (2) for DASS, DASS-GAIN and EPE, the effect of this parameter is very significant (in EPE, the total scheduling cost is 26% higher than BG with 4 tasks, but it gets up to 87% higher with 32 tasks); and (3) the DP-related policies present an intermediate effect, in which the ratio worsens by around 7% every time the number of hard tasks doubles.

2.6 Conclusions

This paper has presented the results of an empirical study on the most relevant scheduling policies for soft tasks in fixed-priority, preemptive real-time systems. In particular, the goal of the study was to characterize the effect of gain time on the behavior of these scheduling policies. The existence of gain time, which is defined as the difference between the WCET of a hard task and its actual execution time, is typical in many real-time systems, for two main reasons. First, because the WCET overestimation is still a common practice in the design of many real-time systems in order to ensure the safety of the schedulability analysis. And second, because even if WCETs are accurately calculated, the typical case for tasks is to consume only a fraction of their WCETs at run time. Traditionally, gain time has been regarded as a design problem for hard tasks (when related to WCET overestimation), but also as an opportunity for soft tasks, which can use this spare time in order to improve their response times. Indeed, some scheduling policies for soft tasks have included specific extensions to make an effective use of this gain time.

The most general conclusion of the paper is that, other things being equal, the increase in gain time in the system significantly reduces the advantages of using any of the policies under study. More specifically, the relative performance *benefits* of all policies with respect to serving soft tasks in background (BG) are significantly

reduced for all policies as gain time increases. This is consistent with the theoretical definition of these policies, where performance can be directly related to some policy variables that depend on the hard *nominal* load (such as the servers' budgets, the run-time available capacity/slack for EPE/DASS, or the promotion times for DP). Furthermore, the results presented in the paper have shown that this negative influence of gain time may affect policies differently, depending on some system parameters, as it is now summarized.

In systems with low hard nominal utilization, gain time produces a homogeneous negative effect on all policies with respect to BG. Although all policies still perform better than BG except in some particular cases (DS and SS actually perform worse than BG in systems with high percentages of gain time, due to their extra overheads), adopting any of them becomes less worthwhile as the amount of gain time increases, especially for systems with high soft load utilization. Moreover, in the case of DASS or DP, their gain-time extensions have no effect on their respective performance benefits with respect to BG.

In systems with high hard nominal utilization, there is an even more pronounced negative impact of gain time on all policies (compared to BG), but this impact does not affect all policies in the same way. Both server-based policies provide good results when no gain time is available, but they rapidly degrade to BG as gain time augments, since their budgets become artificially small, and they cannot compensate this at run time. Moreover, as soft utilization grows, they end up performing worse than BG due to their extra overhead (especially due to the higher number of context switches, which is also a consequence of server budgets being very small). It has to be noted that SS has been implemented according to its official definition by POSIX; a recent study [113] claims that this definition has some defects which directly affect its performance and proposes some corrections, which have not been incorporated to the standard yet. EPE performs worse when there is no gain time available, because of being unable to compensate both its high overhead and its inefficiency at computing the initial capacity values of hard tasks. However, in systems with gain

time, it presents much better results, and it ends up outperforming both server-based policies and DASS. Gain time has a very strong negative effect on DASS, which makes this policy degrade dramatically as gain time augments. In this case, its gain time extension becomes vital to compensate this degradation, to the extent that DASS-GAIN outperforms all other policies, even considering its extra overhead. Finally, the four DP policies (DP plus its three gain time extensions) present the most stable behavior in the performance results as gain time augments. In this case, the incorporation of gain time extensions does not produce a clear benefit, and all policies tend to perform equally (and equal to DASS-GAIN) with greater values of gain time. Also, since that DP has a straightforward implementation and produces little overhead, this policy is probably the best choice for these systems.

Acknowledgment

This work is partially funded by research projects PROMETEO/2008/051, CSD2007-022 and TIN2008-04446.

Appendix A: Figures

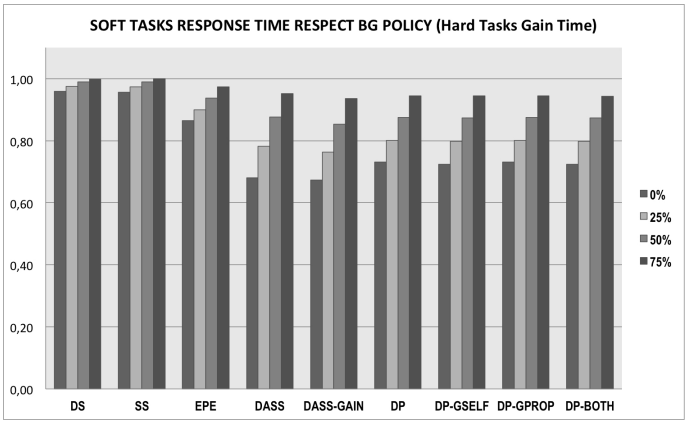


Figure 2.2: Soft task response time ratios as a function of gain time in Experiment 1

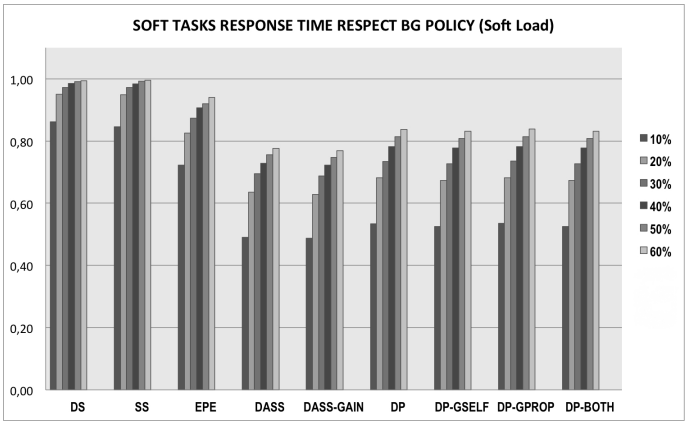


Figure 2.3: Soft task response time ratios as a function of soft load in Experiment 1, 0% of gain time

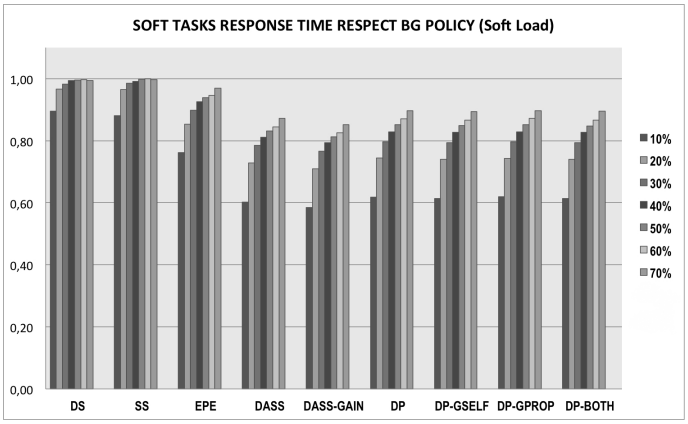


Figure 2.4: Soft task response time ratios as a function of soft load in Experiment 1 with 25% of gain time

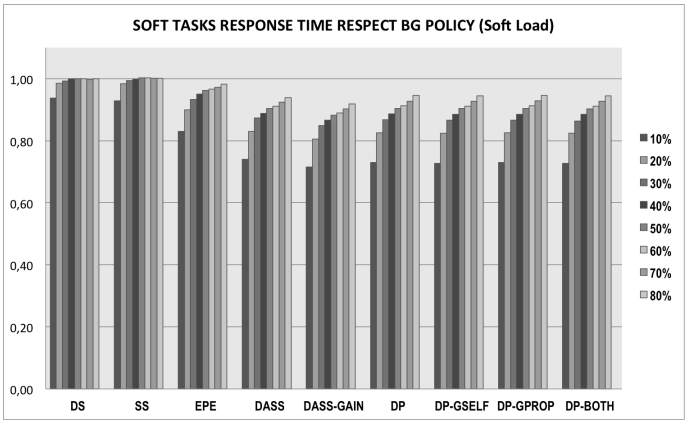


Figure 2.5: Soft task response time ratios as a function of soft load in Experiment 1 with 50% of gain time

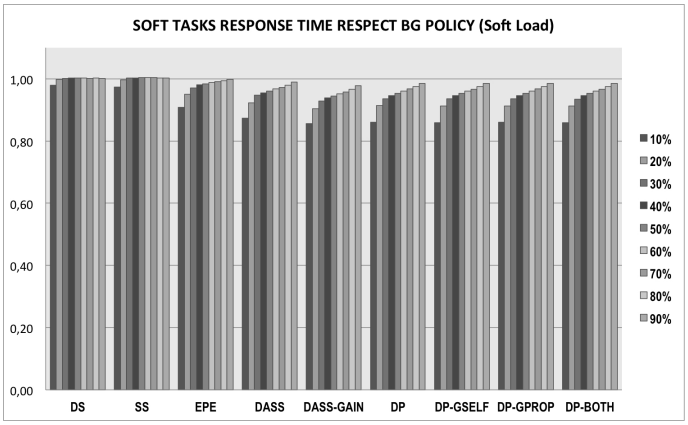


Figure 2.6: Soft task response time ratios as a function of soft load in Experiment 1 with 75% of gain time

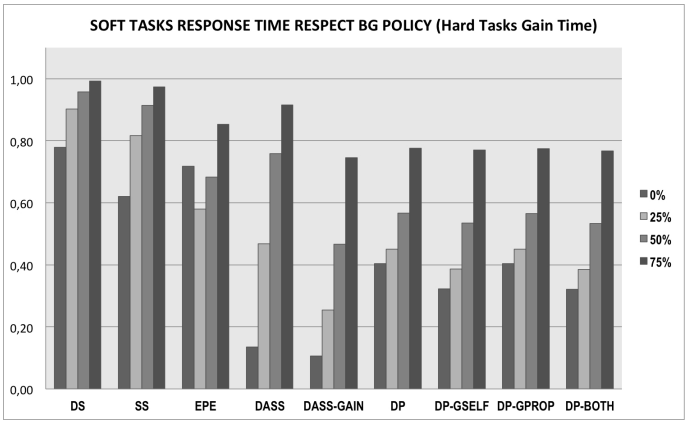


Figure 2.7: Soft task response time ratios as a function of gain time in Experiment 2

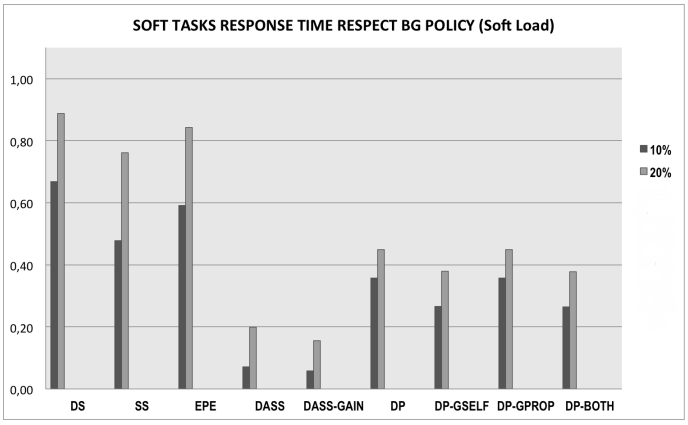


Figure 2.8: Soft task response time ratios as a function of soft load in Experiment 2 with 0% of gain time

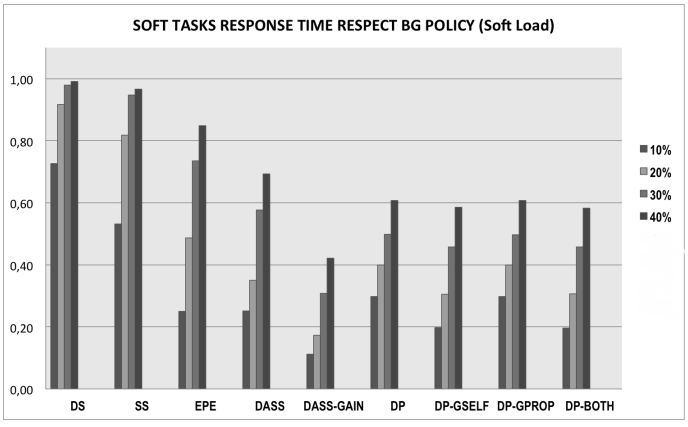


Figure 2.9: Soft task response time ratios as a function of soft load in Experiment 2 with 25% of gain time

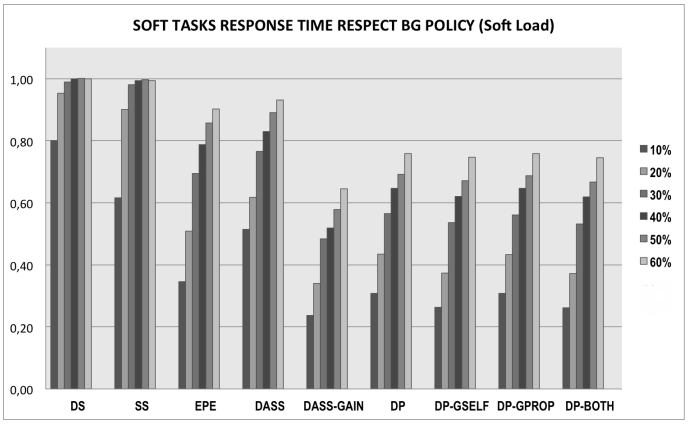


Figure 2.10: Soft task response time ratios as a function of soft load in Experiment 2 with 50% of gain time

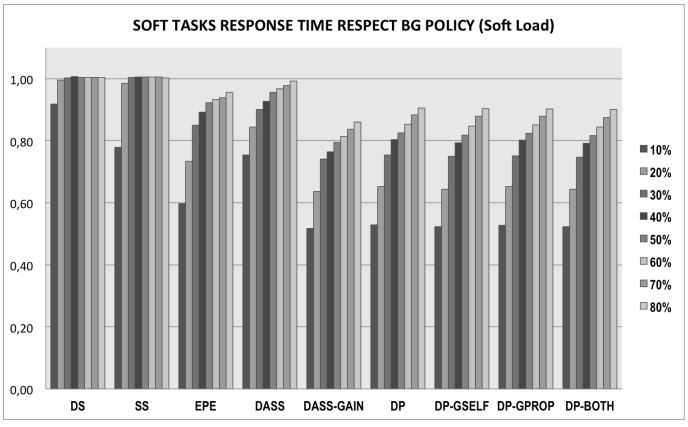


Figure 2.11: Soft task response time ratios as a function of soft load in Experiment 2 with 75% of gain time

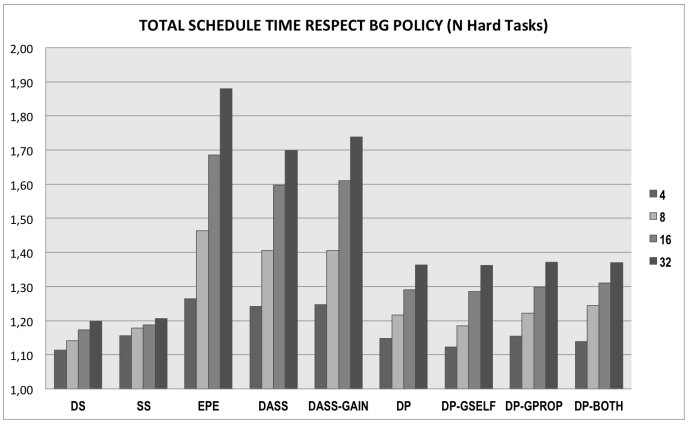


Figure 2.12: Total overhead ratios as a function of the number of hard tasks in Experiment 2

CHAPTER

3

Supporting Social Knowledge in Multiagent Systems through Event Tracing

AUTHORS:
LUIS BÚRDALO, ANDRÉS TERRASA, ANA GARCÍA-FORNES AND AGUSTÍN ESPINOSA
{*lburdalo,aterrasa,agarcia,aespinos*}*@dsic.upv.es*
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA
CNO/ DE VERA SN
46022 VALENCIA, SPAIN

Abstract

Social knowledge is one of the key aspects of MAS in order to face complex problems in dynamical environments. However, it is usually incorporated without specific support on behalf of the platform and that does not let agents take all of the advantage of this social knowledge. At present time, the authors of this paper are working in a general tracing system, which could be used by agents in the system to trace other agents' activity and that could be used as an alternative way for agents to perceive their environment. This paper presents first results of this work, consisting of the requirements which should be taken into account when designing such a tracing system.

3.1 Introduction

These days, the use and importance of multiagent systems (MAS) has increased because their flexible behavior is very useful to deal with complex problems in dynamic and distributed environments. This is not only due to agents individual features (like autonomy, reactivity or reasoning power), but also to their capability to communicate, cooperate and coordinate with other agents in the MAS in order to fulfil their objectives.

The necessary knowledge to support this social behavior is referred to by Mařik et al in [90] as *social knowledge*. This social knowledge plays an important role in increasing the efficiency in highly decentralized MAS. Social abstractions such as teams, norms, social commitments or trust are the key to face complex situations using MAS; however, these social abstractions are mostly incorporated to the MAS at user level; this is, from the multiagent application itself, without specific support from the multiagent platform, by means of messages among agents or blackboard systems. This weak integration of high level social abstractions, also mentioned by Bordini et al in [26], prevents agents in the MAS from exactly knowing what is happening in

their environment, since they depend on other agents actively informing them about what they are doing. This dependance on other agents sets out two major problems. First, it can lead to excesive overhead in some of the agents. And second, it is also difficult to trust the information provided directly by other agents using messages in open MAS.

An alternative solution to provide social knowledge could be an event tracing system, integrated within the multiagent platform, which could be used by agents in the system to perceive their environment without having to actively notify each change to the rest of the agents which could be interested in what they do. Such a tracing system, integrated within the multiagent platform and providing a trustworthy event set which were capable to reflect not only communication among agents, but also agents' perceptions, etc, could be used as a way to provide social knowledge to the MAS. Also, when coming from the multiagent platform, tracing information related to agent's activity is more trustworthy than agent's traditional messages or blackboards, since agents do not have the chance to deliberately communicate false information about their activity. Agents can trust the trace system as much as they can trust the multiagent platform.

Applications which extract information from the system by processing event streams at run time are already considered in the field of event driven architectures [89] and the idea of an standard tracing system available for processes in a system already existed in the field of operating systems (and at present it is contemplated by the POSIX standard[73]). These concepts can be applied to th field of MAS, where event tracing is still considered a facility to help MAS developers in the verification and validation processes.

This paper presents the requirements of such a general, platform-integrated tracing system applied to MAS. These requirements should be taken into account in order to develop a general abstract trace model for MAS, which could be finally incorporated to a real multiagent platform. The rest of the paper presents is structured as folows: Section 4.2 comments existing work by other authors in the field of tracing MAS.

Section 4.3 presents a set of requirements which should be taken into account in order to design a general tracing system which could be used to improve agents sociability. Finally, section 6.6 comments this work's main conclusions and future work which is still to be carried out in order to incorporate such a tracing system to a MAS.

3.2 Event tracing in multiagent systems

One of the most popular tracing facilities for MAS is the Sniffer Agent provided by JADE[21]. This tool keeps tracking of all of the messages sent or received by agents in the system and allows the user/administrator/developer to examine their content. These messages can be stored in a log file to be examined after the application has stopped running, so that the MAS can also be traced off-line. JADE also provides an Introspector Agent, which can be used to examine the life cycle of any agent in the system, its behaviors and the messages it has sent or received.

JADEX[101] provides a Conversation Center, which allows a user to send messages directly to any agent while it is executing and to receive answers to those messages from a user-friendly interface. It also provides a DF Browser to track services offered by any agent in the platform at run time and a BDI Tracer which can be used to visualize the internal processes of an agent while it is executing and show causal dependencies among agents' beliefs, goals and plans. Apart from these facilities, JADEX also incorporates a Remote Agent, which provides access to some of JADE's tracing facilities, like the Agent Introspector and the Sniffer Agent.

The JACK[4] multiagent platform does not provide a sniffer agent, but it supports monitoring communication among agents by means of Agent Interaction Diagrams. It also provides other introspecting tools with different functionalities: a Design Tracing Tool, to view internal details of JACK applications during execution, and a Plan Tracing Tool, to display and trace the execution of plans and the events that handle them. JACK also provides debugging tools that work at a lower level of abstraction in order to debug the multiagent system in a more exhaustive way: Audit

Logging, Generic Debugging/Agent Debugging.

Other examples of tracing facilities provided by platforms is ZEUS' Society Viewer[45] which, apart from showing organisational inter-relationships among agents in the system, it can also show messages exchanged among agents. ZEUS also provides an Agent Viewer, which allows the user/administrator to monitor and change the internal state of the agent, its actions, used resources, etc. JASON[24, 25] provides its Mind Inspector Tool, to examine the internal state of agents across the distributed system when they are running in debugging mode.

Apart from those tools provided by multiagent platforms themselves, there are many tracing facilities provided by third party developers. This is the case of Java Sniffer[120], developed by Rockwell Automation, a stand alone java application based on JADE's Sniffer Agent which is able to connect to a running JADE system in order to track messages among agents, to reason about them and to show them to the user from different points of view. Another third party tool based on JADE's Sniffer Agent is ACLAnalyser[28], which intercepts messages interchanged by agents during the execution of the application and stores them in a relational database. After the execution, this message database can be inspected to detect social pathologies in the MAS. Later work by the same authors [29] combine results obtained with ACLAnalyser with data minning techniques to help in the MAS debugging process.

MAMSY, the management tool presented in [109] lets the system administrator monitorize and manage a MAS running over the Magentix multiagent platform[7]. MAMSY provides graphical tools to interact with the MAS and visualize its internal state at run time, including not only nodes and agents, but also organizational units. It also provides a message tracing tool, similar way to JADE's Sniffer Agent, which lets the system administrator visualize message interchange among agents.

In [95], the authors describe an advanced visualisation tools suite for MAS developed with ZEUS, although the authors also claim these tools could be used with other platforms (more precisely, with CommonKADS). The developed suite allows for inspecting message interchange among agents in a society, displaying graphically

the different tasks in the society and its execution state, examining and modifying the internal state of any of the agents in the system and comparing statistics not only for individual agents, but also for agent societies. It also allows for the graphical display of the different tasks in the society and their execution states, examining and modifying the internal state of any of the agents in the system and comparing statistics not only for individual agents, but also for agent societies.

Tracking messages has also been used in [98], which comments an ampliation of the Prometheus methodology and the related design tool to help the designer to detect protocol violations by tracing conversations among agents in the system and to detect plan selection inconsistencies.

Lam et al present in [81] an iterative method based on tracing multiagent applications to help the user understanding the way those applications internally work. Lam et al also present a Tracer Tool which implements the described Tracing Method. The Tracer Tool can be applied to any agent system implementation, regardless of agent or system architecture, providing it is able to interface with Java's logging API (directly or via a CORBA interface). Results obtained with this method were presented in [82]. Bose et al present in [27] a combination of this Tracer Tool with a Temporal Trace Language (TTL) Checker presented in [3]. This TTL Checker enables the automated verification of complex dynamic properties against execution traces.

As it can be appreciated, tracing facilities in MAS are usually conceived as debugging tools to help in the validation and verification processes. It is also usual to use these tracing tools as a help for those users which have to understand how the MAS works. Thus, generated events are destinated to be understood by a human observer who would probably use them to debug or to validate the MAS and tracing facilities are mostly human-oriented in order to let MAS users work in a more efficient and also comfortable way. Some multiagent platforms provide their own tracing facilities, although there is also important work carried out by third party developers. However, even those tracing facilities which were not designed by platform developer teams are usually designed for a specific multiagent platform. There is not a standard, general

tracing mechanism which let agents and other entities in the system trace each other as they execute like the one provided by POSIX for processes.

3.3 Tracing system requirements

From the viewpoint of the tracing process, a MAS can be considered to be formed by set of tracing entities, or components that are susceptible of generating and/or receiving tracing *events*. The tracing system needs to consider, at least, the following list of components inside the MAS as tracing entities: agents, organizational units (or any type of agent aggregation supported by the multiagent platform) and the multiagent platform itself (and its components).

Unlike existing work on tracing MAS, previously mentioned in Section 4.2, a tracing system which could be used as a knowledge provider must not be human-oriented, but entity-oriented, so that these tracing entities are able to receive events and process them or incorporate them to their reasoning process at run time in order to take advantage from that.

In order to generate trace events, the source code of tracing entities needs to be *instrumented* to include the code which actually produces such events. Attending to where this instrumentation code is placed, trace events can be classified as *platform events* or *application events*.

Platform events are instrumented within the source code of the platform (either in its "core" or in any of its supporting agents). These events represent the generic, application-independent information that the platform designer intends to provide to agents. On the other hand, application events are instrumented within the code of the application agents. These events represent customized run-time information defined by the application designer in order to support specific needs of the application agents.

The rest of the section presents a set of requirements which should be taken into

account when developing such a tracing system. These requirements have been classified in three main groups: functional, efficiency and security requirements.

3.3.1 Functional requirements

Tracing roles. Any tracing entity in the MAS must be able to play two different roles in the tracing process: *event source* (*ES*) and *event receiver* (*ER*). From the viewpoint of tracing entities, these two tracing roles are dynamic and not exclusive, in the sense that each tracing entity can start and stop playing any of them (or both) at any time, according to its own needs. The relation between ES and ER entities is many to many: it must be possible for events generated by an ES entity to be received by many ER entities, as well as it must also be possible for an ER entity to receive events from multiple ES entities simultaneously.

Chronologically ordered event delivery requierement: Events generated in the system must be delivered to ER entities in chronological order or, at least, include information related to the time when they were produced to allow ER entities to process them in chronological order.

Dynamic definition of event types. Trace events can be classified in event types attending to the information which is generated and attached to them when they are generated. In order to let the event processing be more flexible and efficient, it must be possible for tracing entities to dynamically define new event types at run time. This must be applied to both platform and application event types.

Publication of event types. At any time, ER entities must be able to know which ES entities are producing events and of which types. So, as a consequence of event types being dynamic, the tracing system should keep and up-to-date list of such traceable event types (and ES entities) and to make this list available to all tracing entities in the MAS.

On-line and off-line tracing. In order to let entities work with both historical and run time information, both on-line and off-line tracing should be supported. In

on-line tracing, events are delivered to ER entities as they are traced by the tracing system (with a potential delay due to the internal processing of events by the tracing system). In contrast, in off-line tracing, events generated by ES entities are not delivered to running entities, but stored in a log file. Both tracing modes must not be exclusive, meaning that it must be possible for the events generated by any ES entity to be delivered to some ER entities while also being stored in some log files. However, the tracing system does not need to support concurrent access to the events stored in a log file.

3.3.2 Efficiency requirements

In any computing system, tracing can be a very expensive process in terms of computational resources. In the case of MAS, the fact that they are by nature highly decentralized systems, both in number of running entities (agents) and hosts, can make their tracing even more expensive. In this context, the tracing process must be optimized in order to minimize the overhead it produces to the system, since a very sophisticated but excessively costly tracing system can become completely useless in practice. The following list introduces a minimal set of efficiency design guidelines that should be considered when designing a tracing system for MAS, in order to make this system realizable and useful. The first two requirements focus on the potential overload of the tracing system while the last one allows entities to set their own limits in the resources devoted to the tracing process.

Selective event delivery. Each ER entity should be able to express which event types it wants to receive, and the tracing system should only deliver events which belong to such types to the entity. Furthermore, each ER entity should be able to change dynamically which events it wants to receive, since entities may need different tracing information at different times during their execution.

Selective event tracing. The tracing system should not spend resources in tracing events which belong to event types that currently no entity wants to receive.

Resource limit control. Each ER entity should be able to limit the maximum amount of its resources to be allocated to receive events, both in on-line and off-line tracing modes. In on-line tracing, if there is some memory data object where events are delivered to until they are retrieved by the corresponding ER entity, then this entity should be able to define the maximum amount of memory devoted to such data object. In off-line tracing, the ER entity that sets the tracing up to the corresponding log file should be able to define the maximum size of the file.

3.3.3 Security requirements

Tracing in an open MAS has obvious security issues, since many of the events registered by the tracing system may contain sensitive information that can be used by agents to take advantage from, or even to damage, the MAS. This scenario enforces the necessity of applying some security policy over the events that can be delivered to entities, specially if they are application entities. This policy can be materialized in many different ways, but in essence, it has to allow for the definition of security rules in the MAS that limit the availability of events to the *right* ER entities. The following list of requirements express a minimum set of restrictions by which it is possible to incorporate such security rules to the tracing system.

Authorization to ER entities. Each ES entity in the system must be able to decide which ER entities can receive the events that it generates. This can be accomplished by means of an authorization mechanism, provided by the tracing system, which can be used by ES entities to restrict the event types that are available to each ER entity. Such authorization rules must be dynamic, so that ES entities are able to modify the list of authorized ER entities corresponding to each event type at run time.

Supervisor entities. Situations where an entity must be able to access to other ES entity's events in order to fulfil its objectives, even though the ES entity does

not agree with that, are very common in MAS. This can happen, for instance, in normative environments where an agent has to watch the other in order to verify that norms are not being violated and to apply the corresponding sanctions in case they are. The tracing system must also provide mechanisms to let an ER receive events generated by an ES without its authorization under some circumstances.

Delegation of authorizations. If an ER entity is currently authorized to be delivered events corresponding to certain event types, then this entity can delegate this authorization to other ER entities in the system; then, each of them can do so with other entities (potentially forming an *authorization tree*). At any node in the tree, the corresponding entity can add or remove delegations dynamically. If a delegation is removed, all the potential subsequent delegations (subtree) are also removed.

Platform entities authorization. By definition, the tracing system must be granted the authorization for all event types defined in the MAS, both at the platform and application levels. This is required for the tracing system to be able to keep track of any event being generated in the MAS, independently of the privacy rules defined by each ES entity.

3.4 Conclusions and future work

Social knowledge is one of the most important features that make MAS appropiate to deal with complex problems in dynamic and distributed environments. The key to this is the capacity of agents to communicate and coordinate with other agents in the MAS in order to get their objectives. This capacity, though based on high level social concepts such as social commitments, trust, norms or reputation, is usually incorporated to the MAS at user level, using messages or blackboard systems, without support from the multiagent platform. This can produce too much overhead, reducing

the scalability of the MAS. Also, it has to be taken into account that sometimes it is difficult to trust information sent by other agents, specially in open MAS.

A general event tracing system, which agents in the MAS could use to trace other agents in their environment, could be used as a more appropiate and trustworthy social knowledge provider. This paper presents the first step towards defining such a tracing system, which is the identification of its requirements. This paper has identified requirements in different aspects: functionallity, efficiency and security.

Some of the presented requirements set important problems out. Some of these problems are more obvious. For example, the problem of delivering events in chronological order in a distributed MAS. However, others are less evident. For instance, the problem of determining which ES entity is the owner of each trace event, since the instrumented code that produces an event is not always within the source code of the entity which originated it. Just as an example, consider events could as property of those entities which source code has been instrumented to produce them. In this case, all platform events would belong to the multiagent platform, while agents in the MAS would only be owners of application events. It could be more understandable and easier to incorporate considering that events belong to the ES entity which originated them.

Future work will include the design of a general abstract model for MAS which contemplated all of the requirements exposed above and which, after that, could be implemented and incorporated to a multiagent platform.

Acknowledgment

This work is partially supported by projects PROMETEO/2008/051, CSD2007-022 and TIN2008-04446, which is co-funded by the Spanish government and FEDER funds.

CHAPTER

4

TRAMMAS: A Tracing Model for Multiagent Systems

AUTHORS:
LUIS BÚRDALO, ANDRÉS TERRASA, VICENTE JULIÁN AND ANA GARCÍA-FORNES
{lburdalo,aterrasa,vinglada,agarcia}@dsic.upv.es
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA
CNO/ DE VERA SN
46022 VALENCIA, SPAIN

Abstract

Agent's flexibility and autonomy, as well as their capacity to coordinate and cooperate, are some of the features which make multiagent systems useful to work in dynamic and distributed environments. These key features are directly related to the way in which agents communicate and perceive each other, as well as their environment and surrounding conditions. Traditionally, this has been accomplished by means of message exchange or by using blackboard systems. These traditional methods have the advantages of being easy to implement and well supported by multiagent platforms; however, their main disadvantage is that the amount of social knowledge in the system directly depends on every agent actively informing of what it is doing, thinking, perceiving, etc. There are domains, for example those where social knowledge depends on highly distributed pieces of data provided by many different agents, in which such traditional methods can produce a great deal of overhead, hence reducing the scalability, efficiency and flexibility of the multiagent system. This work proposes the use of event tracing in multiagent systems, as an indirect interaction and coordination mechanism to improve the amount and quality of the information that agents can perceive from both their physical and social environment, in order to fulfill their goals more efficiently. In order to do so, this work presents an abstract model of a tracing system and an architectural design of such model, which can be incorporated to a typical multiagent platform.

4.1 Introduction

Due to their flexible and adaptative behavior, multiagent systems are commonly applied to solve complex problems in dynamic and distributed environments. This is not only due to agents' individual features (like autonomy, reactivity or reasoning power), but also to their capability to communicate, cooperate and coordinate with other agents in the multiagent system in order to fulfill their goals. In fact, it is

this social behavior, more than their individual capabilities as agents, what makes multiagent systems so powerful. Social abstractions such as teams, norms, social commitments or trust are the key to face complex situations using multiagent systems.

In [90], Mařik et al. refer to the necessary knowledge to give support to all these social abstractions as *social knowledge*, and they also point that it plays an important role in increasing the efficiency in highly decentralized multiagent systems. Traditionally, these social abstractions are mostly incorporated to the multiagent system at user level; this is, from the multiagent application itself, by means of messages among agents or blackboard systems, without any specific support from the multiagent platform. These traditional methods may be easy to implement; however, each agent's social knowledge depends almost completely on the rest of the agents in the multiagent system actively informing of what they are doing, which has some major problems. First, it can lead to excessive overhead in some agents, specially in situations where agents have to send their information to many other agents because of not being able to determine which of them are really interested in receiving it. Second, it can also be difficult to trust the information provided directly by other agents using messages. This is usually solved by considering that agents are benevolent, but this may not be true in open multiagent systems. And third, it is very difficult to incorporate high level social abstractions, which usually require indirect interaction or coordination, using traditional messages, which are a direct way of communication. This weak integration of high level social abstractions is also mentioned as an important flaw by Bordini et al in [26].

This work proposes the use of event tracing as a way to provide indirect interaction and coordination in multiagent systems, which can be later used to give support to high level social abstractions. In particular, this document introduces TRAMMAS, an abstract event *TRAce Model for MultiAgent Systems* which lets all the entities in the multiagent system share trace information, both at run time or by means of historic information (trace log files).

Applications which extract information from the system at run time are already

considered in the field of event driven architectures [89]. Also, the idea of an standard tracing system available for processes in a system already exists in the field of operating systems, for example, in the case of the POSIX standard[73]. However, event tracing facilities are usually conceived in the field of multiagent systems as debugging tools to help in the system's validation and verification processes. Thus, generated events are destined to be understood by a human observer rather than to be used by agents or other entities in the multiagent system.

The TRAMMAS model is proposed to provide a standardized, trace-based support for indirect communication which may be used by any entity in a multiagent system, not only by agents. In order to do so, the model has adopted the taxonomy published by Omicini el al. in [97], which is based on two main abstractions: agents and artifacts. On the one hand, agents are autonomous, proactive entities that encapsulate control and are in charge of the goals/tasks that altogether define and determine the whole multiagent system behavior. On the other hand, artifacts are those passive, reactive entities in charge of the services and functions that make individual agents work together in a multiagent system. According to this vision, both agents and artifacts are considered as tracing entities by the model. More over, the model also supports aggregations of agents or agents and artifacts. As a consequence, any entity in the multiagent system, as well as the multiagent platform itself are also considered susceptible of generating trace events.

The proposed model is based on the *publish/subscribe* software pattern, which allows subscribers to filter events attending to attributes (content-based filtering). Unlike in some publication/subscription patterns, such as the well known *observer* pattern, the presented trace event model does not require publishers to be aware of which subscribers they have or to which information they are subscribed. The proposed model does not rely on a single, centralized broker, but on a distributed manager, which is in charge of coordinating the event tracing process. This avoids excessive centralization which may lead to bottle necks and poorly scalable systems.

This paper also presents an architectural design, compatible with the TRAMMAS

model, which proposes all the tracing information to be offered as tracing services. As with traditional services, entities in the multiagent system have to request these tracing services when they are interested in receiving tracing information. The architecture is designed to integrate the model within a generic multiagent platform.

The rest of this paper is structured as follows: First of all, Section 4.2 reviews previous work carried out by different authors in the fields of event tracing and indirect interaction in multiagent systems. Then, Section 4.3 reviews generic requirements of a tracing system, which were taken into account when developing the TRAMMAS model, which is described in Section 4.4. Section 4.5 presents an architectural design which allows for the incorporation of the model to a multiagent platform. Section 4.6 presents an example of an agent based GPS system with certain information needs, where TRAMMAS is compared with other techniques which are not based on event tracing. Finally, Section 4.7 comments the conclusions of this work, as well as some future lines of work.

4.2 Related work

There is a need in multiagent systems for indirect ways of interaction and coordination and different research efforts have been carried out in order to satisfy these needs. However, most existing work on event tracing is mainly focused on debugging rather than on the field of agent interaction and communication. This section is divided in two parts. First, Section 4.2.1 will review existing work in the field of event tracing in multiagent systems and second, Section 4.2.2 will review related work focused on indirect interaction and communication.

4.2.1 Tracing in multiagent systems

Event tracing facilities in multiagent systems are usually conceived as debugging tools to help in the validation and verification processes. It is also usual to use these

tracing tools as a help for those users which have to understand how the multiagent system works. Thus, generated events are destined to be understood by a human observer who would probably use them to debug or to validate the multiagent system and tracing facilities are mostly human-oriented in order to let multiagent system users work in a more efficient and also convenient way.

One of the most popular tracing facilities for MAS is the Sniffer Agent provided by JADE[21]. This tool keeps tracking of all of the messages sent or received by agents in the system and allows the user/administrator/developer to examine their content. These messages can be stored in a log file to be examined after the application has stopped running, so that the MAS can also be traced off-line. JADE also provides an Introspector Agent, which can be used to examine the life cycle of any agent in the system, its behaviors and the messages it has sent or received.

Both the Sniffer agent and the Introspector Agent make use of the Event Notification Service (ENS), provided by JADE itself. Agents running over JADE can request the AMS to sniff the activity of other agents or the platform itself in order to be notified each time an event occur. Events managed by the ENS are classified in four main groups: life-cycle related events, message transfer protocol related events, agent messaging events and agent internals related events. The event set provided by the ENS cannot be modified dynamically, in the sense that agents cannot publish new event types for other agents to request them to the ENS. Also, since JADE does not support agent aggregations nor artifacts, only agents are susceptible of generating or receiving trace events.

JADEX[101] provides a Conversation Center, which allows a user to send messages directly to any agent while it is executing and to receive answers to those messages from a user-friendly interface. It also provides a DF Browser to track services offered by any agent in the platform at run time and a BDI Tracer which can be used to visualize the internal processes of an agent while it is executing and show causal dependencies among agents' beliefs, goals and plans. Apart from these facilities, JADEX also incorporates a Remote Agent, which provides access to some of JADE's

tracing facilities, like the Agent Introspector and the Sniffer Agent.

The JACK[4] multiagent platform does not provide a sniffer agent, but it supports monitoring communication among agents by means of Agent Interaction Diagrams. It also provides other introspecting tools with different functionalities: a Design Tracing Tool, to view internal details of JACK applications during execution, and a Plan Tracing Tool, to display and trace the execution of plans and the events that handle them. JACK also provides debugging tools that work at a lower level of abstraction in order to debug the multiagent system in a more exhaustive way: Audit Logging, Generic Debugging/Agent Debugging.

Other examples of tracing facilities provided by platforms is ZEUS' Society Viewer[45] which, apart from showing organizational inter-relationships among agents in the system, it can also show messages exchanged among agents. ZEUS also provides an Agent Viewer, which allows the user/administrator to monitor and change the internal state of the agent, its actions, used resources, etc. JASON[24, 25] provides its Mind Inspector Tool, to examine the internal state of agents across the distributed system when they are running in debugging mode.

Apart from those tools provided by multiagent platforms themselves, there are many tracing facilities provided by third party developers. This is the case of Java Sniffer[120], developed by Rockwell Automation, a stand alone java application based on JADE's Sniffer Agent which is able to connect to a running JADE system in order to track messages among agents, to reason about them and to show them to the user from different points of view. Another third party tool based on JADE's Sniffer Agent is ACLAnalyser[28], which intercepts messages interchanged by agents during the execution of the application and stores them in a relational database. After the execution, this message database can be inspected to detect social pathologies in the MAS. Later work by the same authors ([29]) combine results obtained with ACLAnalyser with data mining techniques to help in the MAS debugging process.

MAMSY, the management tool presented in [109] lets the system administrator monitorize and manage a MAS running over the Magentix multiagent platform[7].

MAMSY provides graphical tools to interact with the MAS and visualize its internal state at run time, including not only nodes and agents, but also organizational units. It also provides a message tracing tool, similar way to JADE's Sniffer Agent, which lets the system administrator visualize message interchange among agents.

In [95], the authors describe an advanced visualization tools suite for MAS developed with ZEUS, although the authors also claim these tools could be used with other platforms (more precisely, with CommonKADS). The developed suite allows for inspecting message interchange among agents in a society, displaying graphically the different tasks in the society and its execution state, examining and modifying the internal state of any of the agents in the system and comparing statistics not only for individual agents, but also for agent societies. It also allows for the graphical display of the different tasks in the society and their execution states, examining and modifying the internal state of any of the agents in the system and comparing statistics not only for individual agents, but also for agent societies.

Tracking messages has also been used in [98], which comments an ampliation of the Prometheus methodology and the related design tool to help the designer to detect protocol violations by tracing conversations among agents in the system and to detect plan selection inconsistencies.

Lam et al present in [81] an iterative method based on tracing multiagent applications to help the user understanding the way those applications internally work. Lam et al also present a Tracer Tool which implements the described Tracing Method. The Tracer Tool can be applied to any agent system implementation, regardless of agent or system architecture, providing it is able to interface with Java's logging API (directly or via a CORBA interface). Results obtained with this method were presented in [82]. Bose et al present in [27] a combination of this Tracer Tool with a Temporal Trace Language (TTL) Checker presented in [3]. This TTL Checker enables the automated verification of complex dynamic properties against execution traces.

As it can be appreciated, although there is also important work carried out by third party developers, many multiagent platforms provide their own tracing facilities.

However, even these tracing facilities which were not designed by platform developer teams are usually designed for a specific multiagent platform. Also, most of this work is focused on human users, rather than agents. There is not a standard, general tracing mechanism which lets agents and other entities in the system trace each other as they execute like the one provided by POSIX for processes.

4.2.2 Indirect interaction and communication

The problem of giving support to additional ways of indirect communication and coordination has already been addressed by other authors using overhearing techniques. Overhearing is normally defined as an 'indirect interaction whereby an agent receives information for which it is not addressee' [83, 78, 55]. This techniques have already been used, among others, in order to maintain social situational and organizational awareness [107], to allow team organization [84, 83], to monitor teams in a non-intrusive way [78] and to develop advising systems [6, 39].

Most of the work in overhearing is modeled and implemented using message broadcasting. This is a very straightforward way to do it, but it does not address the relationship between indirect interaction and the environment. However, using broadcasting to perform overhearing is contradictory, since the definition of overhearing, as well as the overhearer role defined for multi-party dialogues in [91], implies that the sender is not always aware of who is receiving its messages, apart from the specified receivers (this is, who are the overhearers). Using broadcasting makes the difference between the overhearer and the message receiver disappear, since both require to be directly contacted by the sender. Overhearing by broadcasting has also the additional flaw of making it impossible to model sending messages to unknown agents which enter an open system unless these new agents inform of their arrival, which reduces the possibilities of the overhearing model.

Outside the field of overhearing, the environment is claimed in [124] as a first class abstraction, complementary to agents in the system and, at the same time,

independent from them, since it provides the surrounding conditions for agents to exist, as well as an exploitable design abstraction for building multiagent system applications. As pointed out in [100] and [110], the environment should give support to both direct and indirect interaction in multiagent systems and thus, overhearing should be managed by the environment in order to save this gap between the overhearer and the sender.

Authors in [100] propose a model of environment which supports not only direct communication, but also overhearing without broadcasting. The proposed model considers also non-agent entities present in the multiagent system. In a similar way, [110] introduces a model for the environment, which considers not only agents, but also objects (non-agents) and messages. This model lets agents carry out an active perception of their environment, determining after a symbolic data analysis which data are interesting and discarding the rest. However, objects are seen in this model only as entities which generate overhearing information, never as overhearers.

Unlike reviewed work about event tracing in multiagent systems, overhearing techniques are focused in providing information to agents. However, overhearing only considers indirectly receiving regular agent-to-agent messages, which usually do not reflect actions carried out by entities in the system (agents or not), nor by changes in the visible state of these entities.

4.3 Requirements

This section presents a summary of the requirements which were taken into account when developing the presented model for tracing systems. These requirements can be classified in three groups: functional, efficiency and security requirements. A more detailed description of these requirements can be found in [35].

4.3.1 Functional requirements

Any entity in the multiagent system should be able to generate and receive trace events in a non-exclusive way at run time. Events are delivered to entities as they are generated and they must be able to order chronologically any trace event sequence it may receive. In order to let entities work with both historical and run time information, both on-line and off-line tracing must be supported. In on-line tracing, events are delivered to entities as they are generated. In contrast, in off-line tracing, events generated by entities are not delivered to running entities, but stored in a log file, which can be later opened and processed. Both tracing modes must not be exclusive, meaning that it must be possible for the events generated anywhere in the system to be delivered to some entities while also being stored in some log files.

4.3.2 Efficiency requirements

Event tracing must be optimized in order to minimize the amount of consumed resources, as well as the overhead it may produce to the multiagent system. Thus, each entity should be able to limit the maximum amount of resources to be allocated to receive events. In the same way, entities must be able to decide which trace events to receive, and only those trace events which are to be delivered to any entity must be retrieved. In order to do so, it must be possible to classify trace events in different classes or types attending to the information which they represent. It must be possible for entities to dynamically define new event types at run time and, as a consequence, an up-to-date list of available traceable event types and which entities generate them must be available to all entities in the multiagent system. Finally, in order to avoid being a bottle neck for the multiagent system, the support of event tracing must be as decentralized as possible.

4.3.3 Security requirements

Letting tracing entities trace each other's activity has obvious security issues, specially in open multiagent systems. In order to address these issues, each entity must be able to decide which other entities in the system can receive its events, either by means of a direct authorization or either by letting other entity in the system decide in its name. Special circumstances, like normative environments, where certain entities have to be able to access to trace events generated by other entities even without an explicit authorization of the origin entity, must also be addressed. Finally, any trace event being generated in the multiagent system must be susceptible of being traced, independently of the privacy rules defined by each entity.

4.4 The TRAMMAS model

The lack of a standard mechanism which gives support to indirect interaction based on trace events, like the one proposed by the POSIX standard, usually forces developers to design and implement indirect interaction mechanisms as a part of the multiagent application, which makes agent's internal logic more complex and agent applications more difficult to maintain.

This section presents TRAMMAS, a platform independent trace model for tracing events in multiagent systems, considering the set of requirements previously described in Section 4.3, which objective is providing multiagent systems with a mechanism for indirect interaction and communication. Once incorporated to a multiagent system, either at platform or user level, this trace model would let agents and other entities in the system generate and receive trace events generated by other entities in the system. Also, human developers/operators can use these tracing events in order to trace the multiagent system in order to debug it or to verify its functioning.

From the viewpoint of this model, a multiagent system can be considered to be formed by a set of *tracing entities* which are susceptible of generating and/or

receiving certain information related to their activity as *trace events*. Events generated by a tracing entity are recorded and delivered to other tracing entities, so that they can retrieve and process all that information in order to fulfill their corresponding goals. The rest of the section will describe in more detail the TRAMMAS model.

4.4.1 Trace event

This model defines a *trace event* as a piece of data representing an action which has taken place during the execution of an agent or any other component of the multiagent system. Trace events are generated each time the execution flow of an application reaches certain instructions (tracing points) in its source code.

This model defines the following common attributes for each event:

- **Event type:** Trace events can be classified according to the nature of the information which they represent. This event type is necessary for tracing entities in order to be able to interpret the rest of the data attached to the trace event.
- **Time stamp:** Global time at which the event took place, necessary to be able to chronologically sort events produced anywhere in the multiagent system.
- **Origin entity:** The tracing entity which originated the event.
- **Attached data:** Additional data which could be necessary to correctly interpret the trace event. The amount and type of these data will depend on the event type. Some trace events may not need any additional information.

Attending to the origin entity which generates them, trace events can be classified:

- **Domain independent trace events:** These trace events are generated by the multiagent platform itself and thus, they can be present in any multiagent

system. Examples of domain independent trace events could be *new agent in the platform* or *new service request*.

- **Domain dependent trace events:** These trace events are designed as a part of the multiagent system, in order to give support to its specific needs. Within a virtual market, an example of domain dependent trace event could be *sold product*.

Trace events can be processed or even combined in order to generate compound trace events, which can be used to represent more complex information. Both domain dependent and domain independent trace events can also be classified into simple and compound. For instance, within a virtual market, a compound event like *transaction done* could be the result of combining simple trace events *sold product* and *paid product*.

4.4.2 Tracing entities

In this model, a tracing entity is defined as any component of the multiagent system or the multiagent platform which is able to generate or receive tracing information. Thus, from the point of view of the tracing process, any multiagent system is seen as a set of tracing entities. In this trace model, tracing entities can be classified in three main groups:

- **Agents**. Agents are all those autonomous and proactive entities which define the multiagent system behavior. This category includes not only all of the individual application agents in the multiagent system, but also those which may be part of the multiagent platform.

- **Artifacts**. Artifacts are all those passive elements in the multiagent system which are susceptible of generating events at run time or receiving them as an input [97]. Artifacts model elements of the multiagent system such as

databases, resources modeled as web services, physical sensors and actuators and so on. Two or more artifacts can be combined in order to perform more complex tasks and they are also susceptive of generating or receiving trace events as a tracing individual. From the point of view of the tracing system, these combinations of artifacts are also modeled as single artifacts.

- **Aggregations**. If the multiagent system supports aggregations of agents (or agents and artifacts), such as organizational units [14], then such aggregations are modeled by the tracing system as a *single* tracing entities, in the sense that trace events can be generated from or delivered to these entities as tracing individuals.

From the point of view of the model, the multiagent platform can be seen as a set of agents and artifacts. Therefore, elements of the multiagent platform are also susceptible of generating and receiving trace events as any other element in the multiagent system.

4.4.3 Tracing roles

Any tracing entity in the multiagent system is able to play two different roles related to the tracing process (or *tracing roles*): *event source* (*ES*) and *event receiver* (*ER*). ES entities are those which generate trace events as they execute, while ER entities are those which receive these events. The relation between ES and ER entities is many to many: it is possible for events generated by an ES entity to be received by many ER entities, as well as it is also possible for an ER entity to receive events from multiple ES entities simultaneously.

These two tracing roles are not exclusive and any tracing entity can play one or both of them at the same time. Regarding to the time when tracing entities can start and stop playing these roles, there are important differences between agents or agent aggregations and artifacts. On the one hand, agents and aggregations can start or stop

playing any of these roles dynamically according to their current state. On the other hand, artifacts, which are passive/reactive entities, have to adopt the corresponding roles at design time.

The model considers a third tracing role, the *Trace Manager* role (*TM*). The TM role is responsible for controlling and coordinating the entire tracing process: registering tracing entities and event types, as well as giving support to the selective event tracing and security models, further explained in Sections 4.4.4 and 4.4.5. This means that there must be at least one tracing entity playing this role in order to give support to all these necessary features. The TM role can be played by a single entity or by a set of different entities in the multiagent platform at the same time, acting coordinately, even in different nodes of the multiagent system.

When a tracing entity is playing the ER tracing role, the tracing system provides it with a *stream*, which can be seen as a special mailbox where trace events are stored before the ER retrieves them. Streams can either be pieces of memory (in on-line tracing) or log files (in off-line tracing). In both cases, the ER entity which owns the stream can limit its size in order not to overload its resources. In addition, the model defines a set of *full policies* in order to let tracing entities decide what to do with incoming trace events if the stream gets full: stop delivering events to the stream, overwriting previously delivered events in chronological order or flushing events to a log file:

- **Trace until full:** When the tracing stream becomes full, the tracing system stops delivering trace events to that stream and informs the corresponding tracing entity by means of a specific trace event.
- **Trace loop:** When the tracing stream becomes full, the tracing system starts overwriting previously delivered trace events in chronological order, starting from those which were generated first, and informs the corresponding tracing entity by means of a specific trace event.
- **Trace flush:** When the tracing stream becomes full, the tracing system flushes

all events to a log file and continues delivering trace events to the stream after informing the corresponding tracing entity.

Trace until full and *trace loop* policies are specific for on-line tracing, while *trace flush* is a specific policy for off-line tracing.

Figure 4.1 shows all the interactions among ES and ER. In particular, it can be seen how trace events are generated in ES entities before arriving to ER entities, while the TM controls the entire process, interacting with ES and ER entities.

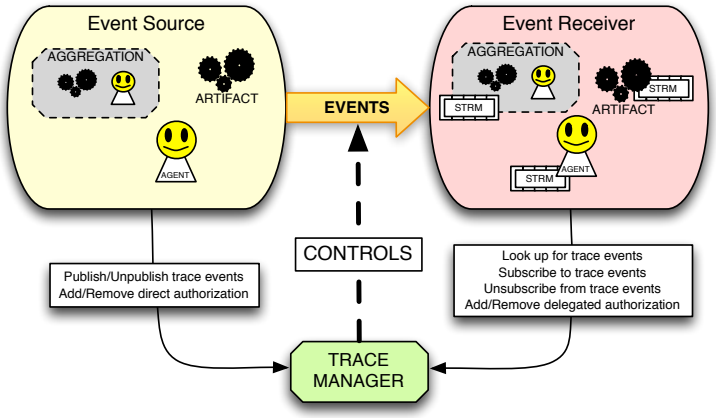


Figure 4.1: Interaction between the different tracing roles in the TRAMMAS model

4.4.4 Selective event tracing

The model defines a *subscription* protocol based on trace event types which helps reducing as much as possible the overhead which tracing information can cause to the multiagent system. ER entities must subscribe to those trace events types which they are interested in. In the same way, once an ER entity is not interested in receiving events of a type to which it had previously subscribed, the ER entity may unsubscribe from them. As a consequence, only trace events of those types to which at least one

ER has previously subscribed are generated and ER entities do not receive any tracing information in which they are not interested in.

Each ES entity has to publish which tracing information it can provide in order to give support to this subscription mechanism at run time. In Figure 4.1 it can be seen how ES entities request the TM to publish and unpublished those trace events they can provide. It can also be appreciated how ER entities are able look for available trace events as well as they can also subscribe and unsubscribe at run time.

4.4.5 Security

When an ES entity publishes its trace events, it has also to specify which roles and/or entities in the multiagent system are authorized to receive such events. In this way, ES entities decide which ER entities can receive their trace events. This is defined as *direct authorization*. When an ER entity wants to receive events of a specific event type which come from a specific ES, it has to be authorized as an entity or it has to be able to assume one of the authorized roles. ER entities which are authorized to receive trace events from certain ES entity can also authorize other roles or entities to receive the same trace events. This is defined as *authorization by delegation*. In this way, the TM maintains an authorization graph for each event type which is being offered by each ES. This authorization graph is dynamic, since tracing entities can add and remove authorizations at run time. When an authorization, direct or by delegation is removed, all those delegated authorizations which depended on the removed one are also removed.

The direct authorization mechanism has the advantage of being conceptually simple; however, asking for an authorization each time an ER entity needs to trace an ES entity can cause an important overhead to ES entities, which may receive too many authorization requests. Authorization by delegation can help reducing the overhead this authorization mechanism can cause to some ES entities while still keeping the security model conceptually simple.

The tracing system does not control which entities can assume each role in order to receive trace events of a specific event type or to add and remove authorizations. The model relies on the multiagent platform to provide the necessary security mechanisms to prevent agents from assuming unappropriated roles.

Figure 4.2 shows an UML-like diagram, where all of the model concepts and relationships commented in Section 4.4 are represented in a formal way.

4.5 Tracing system architecture

This section describes a generic architecture by which it is possible to incorporate the TRAMMAS model to a multiagent system. In particular, the architecture has been designed to be integrated within a generic multiagent *platform* by following a service-oriented approach, with the final goals of taking full advantage of the model and to address efficiency and scalability issues. Once the model is implemented within a real multiagent platform, it can be referred to as the Tracing System, that is, the part of the platform in charge of making it possible that entities running on the platform can trace each other.

The architecture considers the same tracing entities than the model (agents, artifacts and aggregations), and it also considers such tracing entities able to play the *Event Source* (*ES*) and *Event Receiver* (*ER*) tracing roles. According to the model, agents and aggregations will be able to dynamically adopt and abandon such tracing roles at run time, while artifacts will be designed to statically play one of them, or both, and they will not be able to change this at run time. The architecture proposes the ES entities to offer their respective tracing information in the form of *tracing services*, which would be requested by the ER entities that were interested in (and also authorized to) receiving such information. Tracing services are described in Section 4.5.1.

The Trace Manager role is considered by the architecture to be played by the multiagent platform itself. This is accomplished by incorporating the functionality related to this role into the platform, in the form of an extra component called the Trace Manager. Depending on the platform, the Trace Manager may be designed as a single component or as a set of components, possibly distributed among the platform nodes. The Trace Manager component is further described in Section 4.5.2.

Besides incorporating the Trace Manager functionality, the architecture also proposes the platform to be the entity which generates the domain independent trace events, or more precisely, the entity which offers the domain independent tracing services to the rest of tracing entities in the multiagent system. This has two main implications. First, the platform will need to be instrumented at the source code level, in order to generate the trace events corresponding to each domain independent tracing service. And second, the generation of such events at the platform level will allow for the production of tracing information that would not be available otherwise, because of being internal to the platform (e.g., changes in an agent's life cycle). In general, the generation of domain independent tracing information at the platform level presents several advantages, with the most important being efficiency and reliability.

The architecture has been designed to be included in a generic multiagent platform, with no specific requirements other than the support of very general concepts such as 'agent' or 'service'. For example, if the platform does not supports artifacts or aggregations, then the tracing system will not be able to support them as tracing entities, without any other negative effect. On the contrary, having some features available on the platform could make the Tracing System to be easier to implement. For example, if a general authorization scheme is already implanted on the platform, then the Tracing System may be able to use it directly, without the need of implementing its own. In any case, the incorporation of the architecture to the platform will need the availability of the platform's source code, in order to both implementing the Trace Manager and performing the instrumentation which will generate the domain independent trace events.

4.5.1 Tracing services

Event types described in Section 4.4 are modeled in the architecture as *tracing services*. Tracing services are special services which are offered by tracing entities to share their trace events, in a similar way to traditional services. Each tracing entity may offer a set of tracing services, corresponding to the different event types which the tracing entity generates. In the same way as trace events in the model, tracing services can be classified attending to the tracing entity which offers them. Tracing services can also be compound, like trace events in the model, in order to provide more complex tracing information.

When a tracing entity wants to offer any tracing information, it must publish the corresponding tracing service so that other tracing entities can request it if they are interested in its trace events. When a tracing entity does not want to receive certain trace events anymore it only has to cancel the request to the corresponding tracing service. Domain Independent Tracing Services are offered by the multiagent platform and Domain Dependent Tracing Services are offered by tracing entities.

As with traditional services, when tracing services are published, it is also published which agent roles or tracing entities are authorized to request the service. In this way, when an tracing entity wants to request a tracing service, it has to be previously authorized directly or it has to be able to assume an authorized role. Authorizations for a tracing service can be added and removed at run time by the tracing entity which published it by means of updating the corresponding published data on that tracing service. Tracing entities which have assumed a role which is authorized to request a tracing service, can also authorize other roles to request the service.

4.5.2 The Trace Manager

As previously commented, the Trace Manager is not a single component, but a set of components integrated within the multiagent platform, which work together to

coordinate the entire tracing process. Trace Manager functions can be divided in four modules, each of which can be carried out by one or more components in the multiagent platform, even in different nodes in the platform:

- **Trace Entity Module (TEM):** Module in charge of registering and managing all the tracing entities.
- **Tracing Services Module (TSM):** Module in charge of registering and managing all of the tracing services offered by ES entities.
- **Subscription Module (SUBM):** Module in charge of storing and managing subscriptions to each tracing service and ES entity.
- **Authorization Module (AM):** Module in charge of storing and managing the authorization graph for each tracing service and ES.

Figure 4.3 shows how tracing entities interact with the Trace Manager depending on the tracing role that they are playing. These interactions are detailed below:

- **Publish/Unpublish Service:** When an ES entity wants to share its trace events it has to publish the corresponding tracing services before any other entity can request that information. Published tracing services are stored in the TSM. When the ES does not want to offer a tracing service anymore, it has to remove the publication. If the tracing service is the first one offered by the ES entity, then this ES is internally registered in the TEM.
- **Add/Remove Direct Authorization:** ES entities which have published a tracing service can specify which roles have to be assumed by ER entities in order to request that tracing service. ES entities add and remove direct authorizations for each of the tracing services which they provide and the corresponding authorization graph is stored in the AM.

- **Add/Remove Delegated Authorization:** ER entities which have assumed a role which authorizes them to request a tracing service can also authorize other roles to request that tracing service. In the same way, ER entities can remove those delegated authorizations which they previously added. Modifications in the corresponding authorization tree are registered in the AM.

- **Look up for Service:** ER entities can look up in the TSM to know which tracing services are available and which ES entities offer them before requesting any tracing information.

- **Request Service / Cancel Request:** ER entities which want to receive certain trace events from an ES have to request the corresponding tracing service to the Trace Manager. The Trace Manager verifies against the AM that the ER entity has authorization for that tracing service before adding the subscription to the SUBM. When an ER entity does not want to receive events corresponding to a specific tracing service, it has to cancel the request of that service and the corresponding subscription is also deleted in the SUBM. If the ER entity which requests the tracing service was not subscribed to any other tracing service, then this entity is internally registered and listed in the TEM. In the same way, when an ER entity cancels all of its requests, it is internally removed from the TEM. As a consequence, only those trace events for which there is at least one tracing service request in the SUBM are recorded.

Figure 4.3 shows how some of the modules can interact among them in certain circumstances. The first time a tracing entity requests or publishes a tracing service, the SUBM or the TSM registers that entity in the TEM module. In the similar way, when a tracing entity unpublishes a tracing service or modifies its corresponding authorization graph, it may be necessary to cancel subscriptions to that tracing service for certain tracing entities.

4.6 Example

This section will present an example of multiagent system, where different techniques are used to share information among agents. Theoretical costs of transmitting the necessary information can be used as a measure of the efficiency and scalability of each technique and so, they are studied in the best and worst case for each technique.

Let us consider an agent-based GPS system which, apart from suggesting the best route to get to a destination, it lets vehicles share certain information about the state of the road so that other vehicles can find the best route to their destination or modify it if necessary. For instance, important decreases in the speed of vehicles may be indicative of a traffic jam, a change in the direction may be indicative of a blocking of the way, and so on.

Figure 4.4 shows an example of road map, with some nodes (*A* to *F*) connected among them by different roads. In each node there is a station with an agent which receives data from vehicles about the state of its adjacent roads and can also sends recommendations to vehicles about the best route to get to their destination. Each on board device also has an agent in charge of sending, receiving and processing information from the different stations. In the figure, vehicles *1* to *5* departed from different origins to different destinations and have initially been suggested an initial route by the GPS system. Initial routes for each vehicle are also shown in the bottom of the figure. In this case, there is a trouble in the road between *A* and *C* which forces vehicles *1* and *2* to reduce their speed dramatically. The on board GPS system of these two vehicles informs adjacent nodes (nodes *A* and *C*) about this decrease in the speed, and the stations in nodes *A* and *C* should inform to those vehicles which have the road between *A* and *C* in their route so that they are aware of the problem and can make an appropriate decision: find a different way to their destination, go back home or, at least, being alert and avoid having an accident.

From now on, the example will only take consider the transmission of relevant information from stations in each node to those vehicles which may be interested.

The internal reasoning process by which stations receive information from vehicles and determine that there is a traffic jam in a road or that a road is closed is out of scope of this work. The road map will be considered to be in a general situation where there are n_{cars} vehicles in the system and there is a total amount of n_{rem} remarkable situations to be reported to vehicles on the road.

The rest of the section will explain different strategies to solve the problem of sharing all this information among the different vehicles and stations. Two different solutions have been considered: One based on broadcasting information to all vehicles and one based on an event tracing system like the one presented in this paper.

For the solution based on broadcasting, it will be considered that there is a service, available for agents in nodes, in charge of registering agents in the system (like the AMS in FIPA). As a consequence, for each of the n_{rem} remarkable situations, each station would have to ask the service for agents in the market (this implies a message from the station agent to the service provider to ask for the existing agents and the corresponding answer from the service provider to the station agent). After that, the station agent would have to actively send a message to all vehicle agents in the system each time a remarkable situation is detected. Considering the number of vehicles previously specified, the number of messages sent to inform about all of the remarkable situations would be $n_{msg} = (2 + n_{cars}) * n_{rem}$. This solution would not only cause unnecessary information traffic, since messages are sent to vehicles which may not be interested in that information, but also would cause overhead in these non interested vehicle agents, which would also have to process this extra information. In Figure 4.4, vehicles *4* and *5* would be informed of a problem in the road between *A* and *C*, although none of them had it in its route.

To solve this problem using an event tracing system like the one presented in this paper, station agents have to publish data relating to their adjacent roads as tracing services. So, for each adjacent road, station agents publish a tracing service. Vehicles interested in a road request the corresponding tracing service to one of the stations which provides it and, from that moment, they receive a trace event each time a

Table 4.1: Summary of best and worst case costs as a function of the number of vehicles (n_{cars}) for a constant number of remarkable observations ($k_{remarkable}$) with the different techniques: Broadcasting and event tracing.

Number of transmissions for k_{rem} situations		
	Best case	*Worst case*
Broadcast	$k_{remarkable} * (2 + n_{cars})$	$k_{remarkable} * (2 + n_{cars})$
E. Tracing	0	$k_{remarkable} * n_{cars}$

remarkable situation is detected. In this case, no messages are sent, but trace events. For each remarkable situation, the total amount of trace events transmitted (n_{t_events}) would be the number of cars which are interested in that road and requested the corresponding tracing service. In a system where k_{rem} remarkable situations have taken place, the number of trace events transmitted would be $0 \leq n_{t\_events} \leq (k_{rem} * (n_{cars}))$. When there is not any vehicle interested in a road, no trace events are generated and so, the amount of information transmitted is reduced to that which is strictly necessary. Also, since stations do not have to know which vehicles are interested in their adjacent roads, their internal logic remains simple, unlike in previously shown solutions.

Table 4.1 shows the number of transmissions (either messages or trace events) as a function of the number of remarkable situations observed in the system. The number of transmissions in the worst case is in the same order for both techniques. However, the best case is constant for event tracing while it is higher using broadcasting. Also, event tracing simplifies station agents' internal logic, since they only have to process data sent as they drive by vehicle agents and do not have to send the information to all vehicles each time a remarkable situation takes place. Also, vehicle agents can decide about which roads they want to keep informed, which is less overheading for them, since they do not have to process unrequiered information about roads they are not going to drive through or roads they have already passed. In the same way, station agents do not have to spend resources in sending data to vehicles which do not need it.

Theoretical results show that event tracing provides a way to coordinate different

vehicles without having to contact directly with none of them. The amount of information interchanged among agents in the system is reduced to the minimum necessary, which makes the system more efficient and scalable. Station agents' internal logic keeps as simple as possible, which makes the multiagent system be also easier to develop and maintain.

4.7 Conclusions and further work

This paper presents TRAMMAS, an abstract model of an event tracing system for multiagent systems. Unlike most traditional tracing systems, the presented model is not only conceived as a helping tool for multiagent system developers or administrators, but also as an additional indirect communication mechanism which lets agents and other entities in the system generate trace events, as well as receiving events generated by other entities.

By allowing trace event interchange not only among single agents, but also among non-agent entities (modeled as artifacts) and aggregations of agents and artifacts, the proposed model provides a more flexible support for indirect interaction and coordination than message-guided approaches like overhearing. As a consequence, the incorporation of the model to a multiagent system can improve the way in which entities in the multiagent system perceive each other and their environment, which in turn improves the way in which high level social abstractions can be developed and incorporated to the multiagent system.

Along with the trace model, a generic architecture has also been presented. This architecture lets concepts and mechanisms described by the model be incorporated to a multiagent system at the platform level, not only because it is more efficient and flexible than incorporating them at application level, but also because it makes tracing information more reliable, since it has been generated by the multiagent platform itself.

It would also be possible to design a different architecture which were less integrated within the platform and did not required instrumenting the platform source code. However, providing trace event support only at application level would make it very difficult to provide domain independent trace events support, at least in a reliable and efficient way.

Finally, an example where different techniques and strategies have been used to transmit the necessary information among agents has been presented. The analysis performed for each of these techniques shows that event tracing can help reducing the amount of unnecessary information which has to be transmitted and processed, while keeping agents' internal logic as simple as possible and thus, contributing to the scalability and feasibility of multiagent systems.

The trace model presented in this paper has been integrated with the next version of the multiagent platform MAGENTIX, in order to be able test it in a real multiagent system and to compare results using event tracing with other techniques. Currently, the trace model is also being integrated with the multiagent platform SPADE[67].

Acknoledgements

This work is partially supported by projects PROMETEO/2008/051, CSD2007-022, TIN2008-04446 and TIN2009-13839-C03-01.

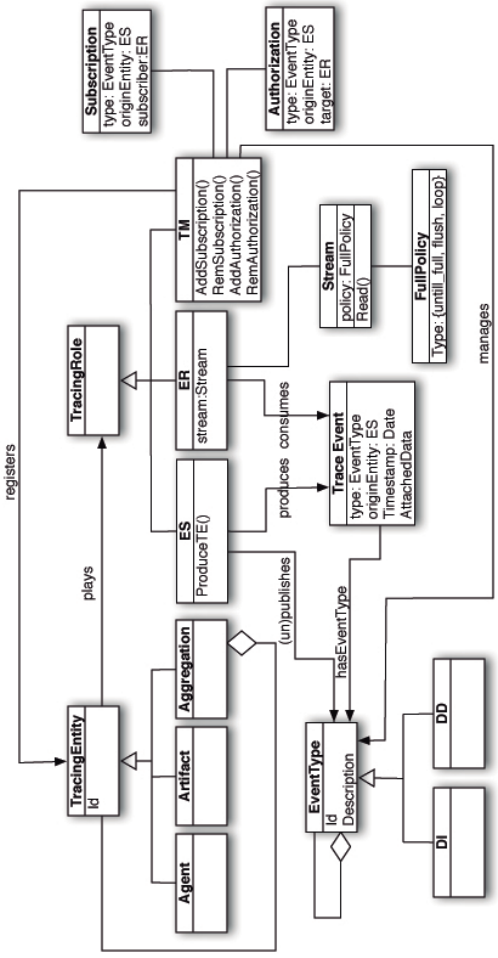


Figure 4.2: TRAMMAS UML model

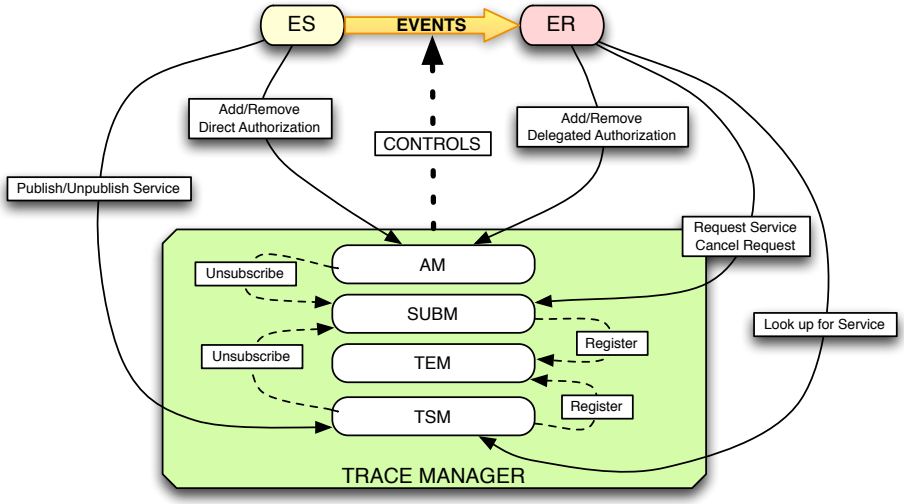
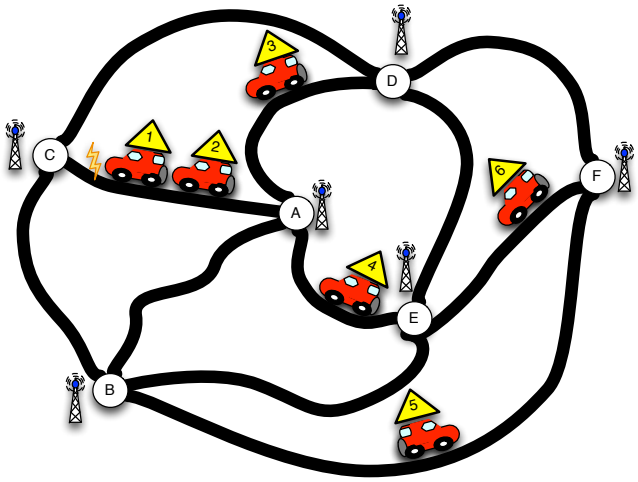


Figure 4.3: Architecture model of the tracing system and interactions among tracing entities depending on their tracing roles and the Trace Manager's internal modules



ROUTE TABLE	
CAR 1: F , E , A , C	CAR 2: E , A , C
CAR 3: D , A , C	CAR 4: E , A , D
CAR 5: C , B , F	CAR 6: F , E , A , C

Figure 4.4: Multiagent system based virtual market where customer and seller agents negotiate before buying/selling products.

CHAPTER

5

Improving the Tracing System in PANGEA Using the TRAMMAS Model

AUTHORS:
LUIS BÚRDALO, ANDRÉS TERRASA, VICENTE JULIÁN, CAROLINA ZATO, SARA RODRÍGUEZ, JAVIER BAJO, JUAN M. CORCHADO
{*lburdalo,aterrasa,vinglada*}*@dsic.upv.es*
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA
CNO/ DE VERA SN
46022 VALENCIA, SPAIN
{*carol_zato,srg,jbajope,corchado*}*@usal.es*
DEPARTMENT OF COMPUTER SCIENCE AND AUTOMATION
UNIVERSITY OF SALAMANCA
SALAMANCA, SPAIN

Abstract

This paper presents the integration of the tracing model TRAMMAS in an agent platform called PANGEA. This platform allows to developed multiagent systems modeled as Virtual Organizations. The concepts of roles, organizations and norms are fully supported by the platform assuring flexibility and scalability. Before TRAMMAS, this platform uses a Sniffer Agent to trace the information reducing its scalability as a centralized mechanism. TRAMMAS proposes the use of event tracing in multiagent systems, as an indi- rect interaction and coordination mechanism to improve the amount and quality of the information that agents can perceive in order to fulfill their goals more efficiently. Moreover, the event tracing system can help reducing the amount of unnecessary information.

5.1 Introduction

Distributed multi-agent systems (MAS) have become increasingly sophisticated in recent years, with the growing potential to handle large volumes of data and coordinate the operations of many organizations [70]. In these systems, each agent independently handles a small set of specialized tasks and cooperates to achieve the system-level goals and a high degree of flexibility [68]. Multiagent systems have become the most effective and widely used form of developing this type of applica- tion in which communication among various devices must be both reliable and efficient. One of the problems related to distributed computing is message passing, which is in turn related to the interaction and coordination among intelligent agents. Conse- quently, a multiagent architecture must necessarily provide a robust communication platform and control mechanisms.

This article presents a multiagent platform based on a Virtual Organization (VO) paradigm. In this paradigm, the social behavior (based on abstractions such as norms, teams, organizations, roles, commitments, etc.) plays and important role and it has to

be incorporated as a decentralized mechanism. This platform called PANGEA (Platform for Automatic coNstruction of orGanizations of intElligent Agents) includes a robust communication model that allows intelligent agents to connect from a variety of devices. On the other hand, TRAMMAS is a tracing model that is incorporated to the platform to improve the amount and quality of the information that agents can perceive from both their physical and social environment, in order to fulfill their goals more efficiently.

The remainder of the paper is structured as follows: the next section introduces some previous works made in tracing systems. Section 5.3 presents an overview of the TRAMMAS model. Section 5.4 explains the inclusion of TRAMMAS inside PANGEA. Next, Sect. 5.5 presents a case study and some results. Finally, Sect. 5.6 shows some conclusions.

5.2 Related Work

The tracing systems within the multiagent architectures have been traditionally used for tasks of "debugging" and the control of certain agents' behavior.

The most outstanding example of this case is the Sniffer Agent and the Introspecter Agent of JADE [21]. The Sniffer Agent allows registering all the messages sent and received by the MAS and later, by means of a log file, to examine its content. The Introspecter Agent allows knowing all the events related to the life cycle of an agent, the messages sent and received as well as its behavior. Nevertheless, in this model all the communication flow is centralized and must pass through this agent to be analyzed and later, registered. Once the information is in the log files, humans must study it since is not prepared for the treatment by agents. The own agents cannot extract log information and the procedure cannot be automated. JADEX [101] provides a Conversa- tion Center, which allows a user to send messages directly to any agent while it is executing and to receive answers to those messages from a user-friendly interface. The JACK [1, 4] multiagent platform supports monitoring communication

between agents by means of Agent Interaction Diagrams. It also provides a Design Tracing Tool, to view internal details of JACK applications during execution, and a Plan Tracing Tool, to trace the execution of plans and the events that handle them. Other examples of tracing facilities provided by platforms are ZEUS's [45] Society Viewer and Agent Viewer, which display organizational inter-relationships among agents and their messages and agent's internal state. Also, JASON [25] provides a Mind Inspector tool to examine agents' internal state.

Apart from those tools provided by multiagent platforms themselves, there are many tracing facilities provided by third party developers. This is the case of Java Sniffer [120], developed by Rockwell Automation based on JADE's Sniffer Agent. Another third party tool based on JADE's Sniffer Agent is ACLAnalyser [28], which intercepts messages interchanged by agents during the execution of the application and stores them in a relational database, which can be lately inspected to detect social pathologies in the MAS. These results can be combined with data mining techniques to help in the multiagent system debugging process [29]. MAMSY, the management tool presented in [109] lets the system administrator monitorize and manage a MAS running over the Magentix multiagent platform [7]. MAMSY provides graphical tools to interact with the MAS and visualize its internal state at run time. In [95], the authors describe an advanced visualization tools suite for MAS developed with ZEUS, although the authors also claim these tools could be used with CommonKADS.

As previously mentioned, the multiagent system that is proposed is based on Virtual Agent Organizations [61]. Consequently, the PANGEA platform makes it possible to create open systems that resolve the inflexibility of a multiagent system. The new open and collaborative architectures require a control focused on the interaction and global knowledge rather than autonomous behaviors. For this reason, traceability has become a key point for the distributed knowledge. As it can be appreciated, tracing facilities in MAS are usually conceived as debugging tools to help in the validation and verification processes. It is also usual to use these tracing tools as help for those

users which have to understand how the MAS works. Thus, generated events are de- signed to be understood by a human observer who would probably use them to debug or to validate the MAS and tracing facilities are mostly human-oriented in order to let MAS users work in a more efficient and also comfortable way. Some multiagent plat- forms provide their own tracing facilities, although there is also important work car- ried out by third party developers. However, even those tracing facilities which were not designed by platform developer teams are usually designed for a specific multiagent platform. This reason leads us to integrate TRAMMAS with our platform to probe its independency and to achieve a distributed way to share knowledge between our PANGEA agents in a distributed way.

5.3 TRAMMAS Overview

Multiagent systems can be considered to be formed by a set of tracing entities or components which are susceptible of generating and/or receiving certain information related to their activity as trace events. A trace event is a piece of data representing an action which has taken place during the execution of an agent or any other component of the multiagent system. Each trace event has these attributes [35]:

- Event type: Trace events can be classified according to the nature of the information which they represent. This event type is necessary for tracing entities in or- der to be able to interpret the rest of the data attached to the trace event.
- Time stamp: Global time at which the event took place, necessary to be able to chronologically sort events produced anywhere in the multiagent system.
- Origin entity: The tracing entity which originated the event.
- Attached data: Additional data which could be necessary to correctly interpret the trace event. The amount and type of these data will depend on the event type. Some trace events may not need any additional information.

Tracing entities can be considered to be playing two different tracing roles. When they are generating trace events, tracing entities are considered Event Source entities (ES). When they are receiving trace events, tracing entities are considered Event Receiver entities (ER). Any tracing entity can start and stop playing any of these two roles, or both, at any time.

This architecture considers three different kinds of tracing entities: Agents, arti- facts and aggregations.

On the one hand, agents are all those autonomous and proactive entities which define the multiagent system behavior. On the other hand, artifacts are all those passive elements in the multiagent system (databases, physical sensors and actuators, etc.) susceptible of generating events at run time or receiving them as an input [97]. Artifacts can combine in order to perform more complex tasks, generating or receiving trace events as a tracing individual. From the point of view of the tracing system, these combinations of artifacts are also modeled as single artifacts.

If the multiagent system supports aggregations of agents (or agents and artifacts), such as teams or organizations, then such aggregations are considered by the tracing system as single tracing entities, in the sense that trace events can be generated from or delivered to these entities as tracing individuals. Agents and artifacts within an aggregation are still tracing entities and thus, they can also generate and receive trace events individually, not only as members of the aggregation.

From the point of view of the architecture, the multiagent platform can be seen as a set of agents and artifacts. Therefore, the components of the platform are also susceptible of generating and receiving trace events.

When a tracing entity is playing the ER tracing role, the tracing system provides it with a stream, which can be seen as a special mailbox where the Trace Manager de- livers the trace events for this ER entity. These streams can either be pieces of memory or log files. In both cases, the ER entity which owns the stream has to limit its size in order not to overload its resources.

Event types are modeled in this architecture as tracing services. A tracing service is a special service which is offered by an ES entity to share its trace events, in a similar way to a traditional service. Each ES entity can offer different tracing services, and the same tracing service can be offered by many different ES entities.

As with traditional services, when an ER entity is interested in receiving trace events of a specific event type, which are generated by a given ES, it has to request the corresponding service. From that moment on, the Trace Manager starts recording the corresponding trace events and delivering them directly to the ER stream until the ER cancels the request. The Trace Manager only records those trace events, which have been requested by an ER entity, so that no resources are spent in recording and delivering trace events, which have not been requested by any ER entity.

The Trace Manager provides a list of all the available tracing services and the ES entities, which offer them. When an ES entity wants to offer any tracing information, it must inform the Trace Manager in order to publish the corresponding tracing service so that other tracing entities can request it if they are interested in its trace events. When a tracing entity does not want to receive certain trace events anymore it has to cancel the request to the corresponding tracing service.

In order to let ES entities decide which ER entities can receive their trace events, when an ES entity publishes a tracing service, it has also to specify which agent roles are authorized to request that service to that ES entity (direct authorization). In this way, when an ER entity wants to request a tracing service to an ES, it has to be able to assume one of the authorized agent roles. ER entities which are authorized to request a tracing service to certain ES entity can also authorize other roles to request the same tracing service to that ES entity. This is defined as authorization by delegation. In this way, the tracing system maintains an authorization graph for each tracing service which is being offered by each ES. This authorization graph is dynamic, since tracing entities can add and remove authorizations at run time. When an authorization, direct or by delegation, is removed, all those delegated authorizations which depended on the removed one are also removed.

The tracing system does not control which entities can assume each role in order to request or to add authorizations for a tracing service. It is the multiagent platform which has to provide the necessary security mechanisms no prevent agents from assuming inappropriate roles.

5.4 Description of PANGEA Including TRAMMAS

Developing PANGEA, we are looking for a platform that can integrally create, manage and control VOs. 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. For example, preventing an agent to take an unauthorized role.
- Sniffer: manages the message history and filters information by controlling com- munication initiated by queries.

One of the most important features that characterize the platform is the use of the IRC protocol for communication among agents. Internet Relay Chat (IRC) is a

Real Time Internet Protocol for simultaneous text messaging or conferencing. This protocol is regulated by 5 standards: RFC1459[96], RFC2810 [74], RFC2811 [75], RFC2812[76] y RFC2813 [77]. 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:

$$\textit{prefix command command-parameters}\backslash r\backslash n$$

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. For the diffusion of the defined trace event taking into account the format of the IRC messages, the event attributes have been included as parameters of the messages. The communication platform is able to treat the messages according to its format and to distribute them suitably.

In line with this design, the inclusion of TRAMMAS in PANGEA is relatively easy. As previously commented, a tracing service is a special service which is offered by an ES entity to share its trace events. Therefore, the unique existing condition is that, as far as possible, an ES entity should implement its tracing service as a Web Service. This allows the ServiceAgent of PANGEA to offer the services to all the agents in the rest of suborganizations.

An *EventTracing Suborganization* has been included to create the tracing system. Figure 5.1 shows the agents and its relationships. This suborganization carry out the tasks that the model TRAMMAS assign to the Trace Manager. Four agents form the suborganization:

- TraceEntityAgent in charge of registering and managing all the tracing entities.

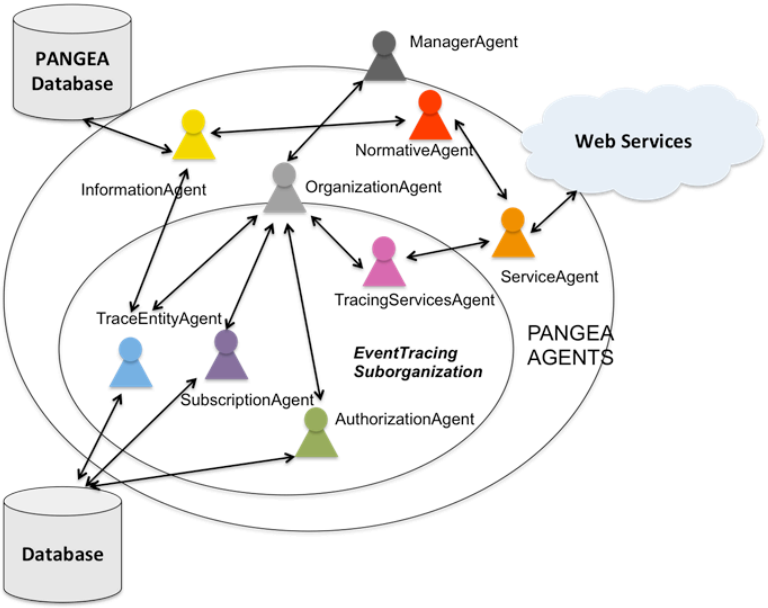


Figure 5.1: Platform overview

- TracingServicesAgent in charge of registering and managing tracing services offered by ES entities.
- SubscriptionAgent, which stores and manages subscriptions to each tracing service and ES entity.
- AuthorizationAgent which stores and manages the authorization needed for each tracing service and ES entity.

Figure 5.2 shows how tracing entities interact with the EventTracing Suborganization.

5.5 Case Study and Results

The case study presents an example of VO, where different techniques are used to share information among agents. The agents created by PANGEA are implemented

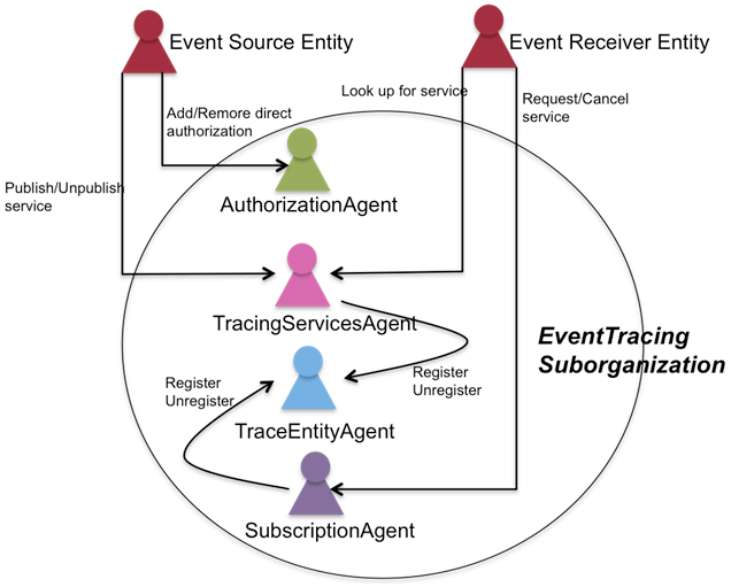


Figure 5.2: Interactions between agents in the EventTracing Suborganization

using different technologies and have different features, among which are the use of sensors. Virtual Organizations of agents are an interesting possibility to handle the large amounts of data provided by sensors because they can provide the necessary capacity to handle open and heterogeneous systems such as those normally found in the information fusion process. Several agents in the VO will be deployed on computers within a LAN and various agents will be on mobile devices.

Theoretically, the cost of transmitting the necessary information between them can be used to measure the efficiency and scalability of PANGEA platform. It also enables to compare the techniques used in the construction of each of the agents.

Let us consider a VO focuses on people detection, specifically developed for a work environment, which can facilitate tasks such as activating and personalizing the work environment; these apparently simple tasks are in reality extremely complicated for some people with disabilities [122].

ZigBee sensors are used to deploy the detection prototype. ZigBee is a low cost, low

power consumption, two-way wireless communication standard that was developed by the ZigBee Alliance [2]. It is based on the IEEE 802.15.4 protocol, and operates on the ISM (Industrial, Scientific and Medical) band at 868/915MHz and a 2.4GHz spectrum.

The proposed proximity detection system is based on the detection of presence by a localized sensor called the control point which has a permanent and known location. Once the Zigbee tag carried by the person has been detected and identified, its location is delimited within the proximity of the sensor that identified it. Consequently, the location is based on criteria of presence and proximity, according to the precision of the system and the number of control points displayed. The parameter used to carry out the detection of proximity is the RSSI (Received Signal Strength Indication), a parameter that indicates the strength of the received signal. This force is normally indicated in mW or using logarithmic units (dBm). 0 dBm is equivalent to 1mW. Positive values indicate a signal strength greater than 1mW, while negative values indicate a signal strength less than 1mW [122].

In our Case Study we have a distribution of computers and laptops in a real office environment, separated by a distance of 2 meters. The activation zone is approximately 90cm, a distance considered close enough to be able to initiate the activation process. It should be noted that there is a "Sensitive Area" in which it is unknown exactly which computer should be switched on; this is because two computers in close proximity may impede the system's efficiency from switching on the desired computer. Tests demonstrate that the optimal distance separating two computers should be at least 40cm.

The agents share certain information about the state of the sensors so that other agents can carry out the detection in an optimal way. For instance, important increases in the RSSI of sensors may be indicative of a proximity to a computer and so on.

The example considers the transmission of relevant information of sensors between agents which may be interested. The internal reasoning process by which agents receive information from sensors is out of scope of this work. The case study will

Number of transmissions for n_{rem} situations		
	Best case	*Worst case*
Broadcast	$k_{remarkable} * (2 + n_{sens})$	$k_{remarkable} * (2 + n_{sens})$
EventTracing Suborganization	0	$k_{remarkable} * (2 + n_{sens})$

Table 5.1: Summary of best and worst case costs as a function of the number of N_{sens} agents for a constant number of remarkable situation ($K_{remarkable}$)

be considered to be in a general situation where there are n_{sens} agents in charge of controlling n sensors in the system and there is a total amount of n_{rem} remarkable situations to be reported to agents. Table 5.1 shows the number of transmissions as a function of the number of remarkable situations occurred in the system. The number of transmissions in the worst case is in the same order for both techniques (broadcast and the *EventTracing Suborganization*). However, the best case is constant for event tracing while it is higher using broadcasting.

Results show that the event tracing technique provides a way to coordinate different agents in charge of sensors without having to contact directly with none of them. The amount of information interchanged among agents in the system is reduced to the minimum necessary, which makes the system more efficient and scalable.

5.6 Conclusions

This paper has presented a platform called PANGEA, which has been improved thanks to TRAMMAS. PANGEA 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. Before TRAMMAS, the Sniffer agent offers services that can be invoked to study and extract message information but this was centralized and limited

if we want to create a platform for building Large-Scale Agent-Based Systems.

TRAMMAS offers an additional indirect communication mechanism which lets agents and other entities in the system generate trace events, as well as receiving events generated by other entities. The incorporation of this model to PANGEA has improved the way in which entities and agents perceive each other and their environment, which in turn improves the way in which high-level social abstractions can be developed and incorporated to the multiagent system.

Finally, the event tracing suborganization can help reducing the amount of unnecessary information which has to be transmitted and processed, while keeping agents' internal logic as simple as possible and thus, contributing to the scalability and feasibility of VOs.

Acknowledgement

This work has been partially supported by the MICINN project TIN 2009-13839-C03-03.

CHAPTER

6

An Adaptive Framework for Monitoring Agent Organizations

AUTHORS:
JUAN M. ALBEROLA, LUIS BÚRDALO, VICENTE JULIÁN, ANDRÉS TERRASA AND ANA GARCÍA-FORNES
{jalberola,lburdalo,vinglada,aterrasa,agarcia}@dsic.upv.es
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN
UNIVERSIDAD POLITÉCNICA DE VALENCIA
CNO/ DE VERA SN
46022 VALENCIA, SPAIN

Abstract

Multiagent technologies are usually considered to be suitable for constructing agent organizations that are capable of running in dynamic and distributed environments and that are 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 mechanism, which helps agents detecting adaptation requirements dynamically at run time, and an Trace&Trigger, which is an adaptation framework for agent organizations. It consists of an event-tracing-based monitoring mechanism that 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 that are present in other approaches by allowing the dynamic specification of the information that has to be retrieved by each agent at each moment for adaptation deliberation, avoiding the transference of useless information for adaptation deliberation. This framework has been integrated in the Magentix2 multiagent platform. In order to test its performance benefits for any agent organization, an example based on a market scenario is also presented.

6.1 Introduction

Nowadays, one of the goals of multiagent systems is to construct systems that are capable of autonomous and flexible decision-making and that can cooperate with other entities within a society. In these scenarios, dynamic agent organizations that are able of adjusting themselves in order to gain advantage in their current environments are likely to become increasingly important [88]. Similar to the needs of human organizations [53], 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. As pointed out by Dignum et

al. in [57], in these cases, the changes in the environment of agents are the ones that trigger reorganization, and, thus, this dynamic adaptation requires that systems be able to evaluate their own health in order to find out when an adaptation is needed. Being able to monitor an agent organization is important in order to determine *why* and *when* an organization needs to be adapted. According to [69], 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. In most of them, 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 [5], 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 that can adapt at run time. It is necessary to be able to count on an adaptive approach that can overcome the monitoring limitations imposed by static designs. Therefore, an adaptive approach should apply not only to the behavior and structure of the system, but it should also apply to the design of the monitoring system [106], especially when dealing with the management of complex systems over long periods of time.

This paper presents Trace&Trigger, which is an agent organization adaptation framework that consists of a dynamic monitoring mechanism and an adaption assistant. The monitoring mechanism helps agents detect 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 6.2 describes in detail both components of the adaptation framework: The event-tracing-based monitoring system and the adaption mechanisms. Section 6.3 shows an example of how to incorporate the framework to an adapting agent organization; and the performance of the organization is evaluated in Section 6.4. Section 6.5 details previous work by other authors in the field of monitoring and adapting multiagent systems. Finally, Section 6.6 presents the main conclusions of this work.

6.2 The Trace&Trigger Framework

The Trace&Trigger framework presented in this work has been designed to run on the Magentix2 multiagent platform [60], which is a platform for open multiagent systems developed in Java. This platform provides a specific mechanism to obtain the information necessary for any adaptation (Section 6.2.1). Specific mechanisms have been incorporated to let agents determine the costs and benefits of performing an adaptation at run time (Section 6.2.2) as well as the mechanisms required to carry out the necessary actions to perform that adaptation (Section 6.2.3).

6.2.1 Magentix2 Support

In addition to the message-based communication layer, Magentix2 also provides communication layer for event-tracing, which allows agents to generate and receive trace events at run time. As a result, agents and other entities running on the Magentix2 platform can not only communicate in a direct way by means of ACL messages, but they can also communicate in an indirect way by means of trace events. These event tracing facilities have been incorporated to the platform according to the TRAMMAS model, which is a platform-independent trace model for tracing events in multiagent systems. Its objective is to provide multiagent systems with

a mechanism for indirect interaction and communication.

The TRAMMAS model conceives the multiagent system as a set of *trace entities* that share information by means of generating and receiving *trace events*. A *trace entity* is any component in the multiagent system that is able to generate and receive *trace events*: agents, non-agents (artifacts, according to the definition in [97]), or aggregations of agent and non-agent entities. However, this work will only consider individual agents. A *trace event* is a piece of data that represents a significant computation that takes place during the execution of any component inside the multiagent system. This model defines the following common attributes for each event:

- **Event type:** Trace events can be classified according to the nature of the information which they represent. So that, the rest of the data attached to the trace event can be interpretated.
- **Time stamp:** Global time at which the event took place; it is necessary to be able to chronologically sort events produced anywhere in the multiagent system.
- **Origin entity:** The trace entity that originated the event.
- **Attached data:** Additional data that could be necessary to correctly interpret the trace event. The amount and type of these data will depend on the event type. Some trace events may not need any additional information.

Trace entities in the multiagent system may participate in the tracing process by playing two different *tracing roles*: the *Event Source* (ES) role and the *Event Receiver* (ER) role. ES entities are those that generate trace events as they execute, while ER entities are those that receive these events. The relation between ES and ER entities is many-to-many: it is possible for events generated by an ES entity to be received by many ER entities; it is also possible for an ER entity to receive events from multiple

ES entities simultaneously. These two tracing roles are not exclusive and any trace entity can play one or both of them at the same time.

The model defines a *publication/subscription* protocol by which: (1) any agent can publish the types of events that it is able to generate (before generating them); and (2) any agent can subscribe to those trace events in which it is interested (before starting to receive them). This protocol helps reduce as much as possible the overhead that tracing information can cause to the multiagent system. ER entities must subscribe to those trace event types that they are interested in. Similarly, once an ER entity is not interested in receiving events of a type to which it had previously subscribed, the ER entity may unsubscribe from them. As a consequence, only trace events of those types to which at least one ER has previously subscribed are generated and ER entities do not receive any tracing information in which they are not interested. 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 *trace services* in a way similar to the way that traditional services are offered in the multiagent system.

A third tracing role, the *Trace Manager* role (*TM*), is also considered in the model. It is responsible for controlling and coordinating the entire tracing process: registering tracing entities and event types, and giving support to the tracing and security models. This means that there must be at least one trace entity playing this role in order to give support to all these necessary features. The model establishes that the TM role can be played by a single entity or by a set of different entities in the multiagent platform at the same time (in coordination) even in different nodes of the multiagent system.

The tracing process and the relations and interactions among tracing roles in the system are shown in Figure 6.1.

The tracing facilities described above have been incorporated to Magentix2 by incorporating a specific agent that plays the *TM* role. Agents have to send an ACL message to the TM agent whenever they want to publish or unpublish their

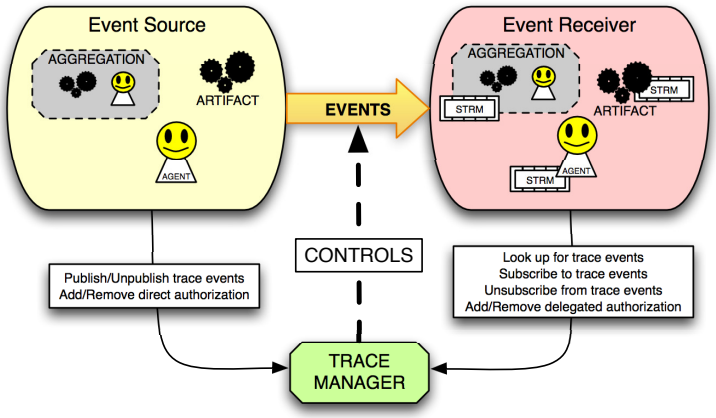


Figure 6.1: Interaction between the different tracing roles in the TRAMMAS model

available trace services and also when they want to subscribe to a trace service or to unsubscribe from it. The TM agent interacts with the Magentix2 communication layer so that trace events generate by an agent are only injected into the network if there is an agent interested in receiving them; no trace event is received by an agent unless the agent has previously requested it.

The TRAMMAS abstract model and the architecture model (which was considered in order to incorporate event tracing facilities to the Magentix2 platform) are described in more detail in [36].

6.2.2 Organization Management Module

Magentix2 provides support to virtual organizations by means of the THOMAS architecture [105], which defines flexible services that can be used by agents. This architecture has been used to define the organization's management and the services provided by agents. The THOMAS architecture is composed of 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 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, and change their relationships or the roles that they play in the organizations. A special agent, which is called manager agent, is defined to manage the execution of each organization. This agent has complete information about the current state of the organization and has permission to interact with the SF and OMS to change it.

6.2.3 Adaptation Module

Since the manager agent is in charge of coordinating every adaptation process in an agent organization, it 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) [8, 12] is the service that is in charge of calculating (at any time) which adaptation has the lowest impact for the system. Individual impacts that are calculated by the organization manager agent are transferred to the RF in order to calculate the adaptation of the organization. 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) [9, 11]. This mechanism calculates transitions in different dimensions (roles, services, relationships, agent population) from the current organization to other organizations. These transitions

have high expected utility based on the cost of the transition to these new organizations. The MTDM decides which transition is finally implemented and provides the sequence of changes required to carry out the transition. We summarize the main components of this model below.

6.2.3.1 Organization

Organization models allow us to represent both the elements that make up the organization and the interactions among these elements. Several approaches can be found in the literature for modeling agent organizations based on the requirements of the applications. Current organization models have been compared and reviewed by works such as Vázquez-Salceda et al. [121], Dignum [56], or Argente et al. [15].

Although several approaches can be used to model organizations, we use the following adaptation of the organization model proposed in [58] since we found it to be appropriate for the requirements of the model proposed.

An *organization* at a specific moment t is defined as a tuple $O^t = \langle O^t_O, O^t_R \rangle$, where O^t_O stands for *Organizational Objects* and represents the individual objects of the organization. It is defined as $O^t_O = \{R^t, S^t, A^t\}$, where R^t represents the set of roles contained in the organization at a specific moment t; S^t represents the services that the organization is offering at a specific moment t; and A^t represents the population of agents at a specific moment t.

O^t_R stands for *Organizational Relationships* and represents relationships of the organization by means of a link between the objects. It is defined as

$$O^t_R = \{offers^t, provides^t, plays^t, acquaintance^t\},$$

where:

- $offers^t = \{(r, s) \in R^t \times S^t\}$ represents the relationships between roles and

services, where (r, s) represents that the role r offers the service s at moment t.

- $provides^t = \{(a, s) \in A^t \times S^t\}$ represents the relationships between agents and services, where (a, s) represents that the agent a provides the service s at moment t.

- $plays^t = \{(a, r) \in A^t \times R^t\}$ represents the relationships between agents and roles, where (a, r) represents that the agent a plays the role r at moment t.

- $acquaintance^t = \{(a, a') \in A^t \times A^t\}$ represents the relationships between a pair of agents, where (a, a') represents that the agents a and a' are connected by an acquaintance relationship at moment t. These relationships define the structural topology of the organization.

Given an organization O^t at a specific moment t, in order for an agent a to be able to play a role r at time t, agent a must provide all the services s that r offers at time t:

$$\forall (a, r) \in plays^t \mid (r, s) \in offers^t \rightarrow (a, s) \in provides^t$$

6.2.3.2 Organization transition

The concept of *organization transition* was first introduced in [52] and allows us to relate two different organizations at different moments, current (c) and future (f). It is the mechanism by which an organization is adapted into a new one. This mechanism is based on individual changes that are applied to the objects and relationships of O^c in order to obtain the objects and relationships of O^f.

A *transition event* (ε) defines each individual change that can be applied to an object or to a relationship during the organization transition in terms of addition or deletion. An addition transition event applied to an object or to a relationship (e.g, $add\_agent(a), add\_provides(a, s)$) causes the object or the relationship to be

added to the specific set of O^f, while a deletion event applied to an object or to a relationship causes the object or the relationship to be deleted from the specific set of O^f. Given two organizations, O^c and O^f, we define $\tau = \{\varepsilon_1 \dots \varepsilon_n\}$ as the *set of transition events* that cause a transition to O^f when all of them are applied to O^c.

6.2.3.3 Organization Transition Impact

In order to calculate the organization with the highest potential for improvement in utility based on the transition cost for several changes, we define the concept of organization transition impact. This impact is a measurement of the effects of an organization transition in terms of organization utility based on the costs for carrying out this transition.

The application of the set of events τ associated to an organization transition provides us with information regarding what changes must be carried out in order to fulfill the transition. Each event $\varepsilon \in \tau$ has an associated impact $i(\varepsilon)$ if ε is applied. This impact represents the costs/benefits that the application of this event produces in the organization. This impact shows the effect of this event in the components involved in the change and also how other components are affected by this event. Moreover, the impact shows the cost for carrying out the application of the event.

For any set of events τ that allows a transition from the current organization O^c to a future organization O^f, we define the impact that is associated to the organizational objects $i(\tau_{O_O})$ as the impact of applying all the events associated to objects. This impact is computed as the aggregated impact of the events that allow a transition from O^c_O to O^f_O and entails operations of addition and deletion of roles, services, and agents:

$$i(\tau_{O_O}) = \sum_{\varepsilon \in \tau_{O_O}} i(\varepsilon)$$

Similarly, we define $i(\tau O_R)$ as the impact of applying all the events associated to relationships. This impact allows a transition from O^c_R to O^f_R and refers to addition

and deletion events of $offers$, $provides$, $plays$, and $acquaintance$ relationships:

$$i(\tau_{O_R}) = \sum_{\varepsilon \in \tau_{O_R}} i(\varepsilon)$$

Finally, we can compute the organization transition impact as:

$$I(\tau) = i(\tau_{O_O}) + i(\tau_{O_R}) = \sum_{\varepsilon \in \tau} i(\varepsilon)$$

If τ is composed by a sequence of ordered subsets $\tau_1, \ldots, \tau_n$, the organization transition impact is represented as the aggregation of the impact of all the subsets:

$$I(\tau) = \sum_{\tau_i \in \tau} I(\tau_i)$$

Each organization transition that is focused on a specific dimension provides the future organization O^f that could be transitioned to, which minimizes the organization transition impact.

6.2.4 Adaptation Life-Cycle

Figure 6.2 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 out that adaptation.

The organization manager needs to obtain certain information that is related to the organization performance at run time. The organizational knowledge that the manager agent possesses is used to estimate the adaptation impacts of individual changes. The monitoring of the organization behavior is carried out by means of the support for event tracing provided by Magentix2. To share their relevant information,

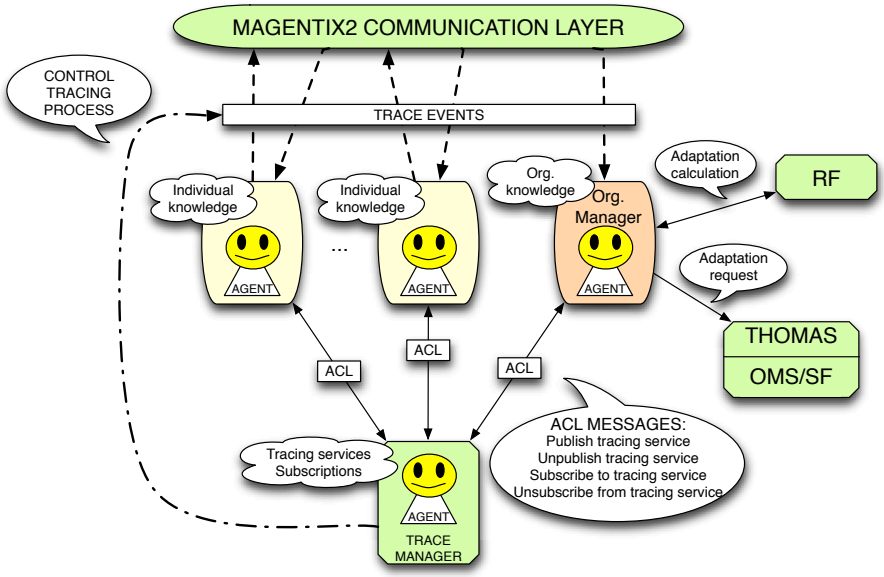


Figure 6.2: Trace&Trigger framework

agents in the organization publish their trace services by sending an ACL message to the TM. Those agents in the system that are interested in that information subscribe to those trace services by requesting it from the TM via an ACL message, too. Since information required for adaptation deliberation can change at run time, the organization manager sends requests to the TM agent for dynamically subscribing or unsubscribing. In this way, the organization manager agent retrieves all the information that is needed 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 trace services and unsubscribe 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 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.

6.3 Case study

To help demonstrate how the Trace&Trigger framework can improve the adaptation capabilities of an organization, a case 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 provide stock to other enterprises. The enterprises represent an organization whose objective is to make as much profit as possible. The organization can improve its performance at run time by adapting to the needs and demands of the market. Similar to the domain proposed in this case study, other domains with restrictions could also be used ([87]).

The organization has been modelled following the notation presented in Section 6.2.3.1. At a given moment t, the agent organization is composed of a set of agents $A^t = \{a_1 \dots a_n\}$, which represent the 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 a current stock $stock^t(a_x, s_y)$ associated to it. 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 them to share their stock with other agents. An acquaintance relationship $acquaintance^t(a_z, a_x)$ allows an agent a_z to be 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 a list of stock providers associated to it, which is represented as $SP^t(a_x, s_y) = \{a_z, \dots, a_n\}$.

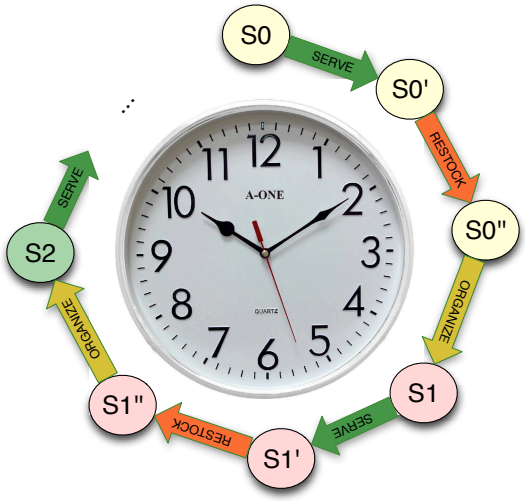


Figure 6.3: Life cycle of the agent organization

At each time step, the organization passes through three states: *Serve* (state S), *Restock* (state S'), and *Reorganize* (state S''). Figure 6.3 shows the transitions between these states. In the state S, each agent a_x receives a number of requests for each service s_y that it provides, which is represented as $requests^t(a_x, s_y)$. Sales of a product s_y that are carried out by a provider a_x at time t are represented as:

$$sales^t(a_x, s_y) = \begin{cases} requests^t(a_x, s_y) & \text{if } stock^t(a_x, s_y) > requests^t(a_x, s_y) \\ stock^t(a_x, s_y) & \text{otherwise} \end{cases}$$

and therefore, the stock is reduced: $stock^t(a_x, s_y) - sales^t(a_x, s_y)$.

After receiving the requests, the organization reaches the state S', in which agents 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. Providers are requested sequentially until one of them agrees to restock 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, which is represented as $tcost(a_z, a_x, s_y)$. If none of the stock providers is able to restock the demand, finally the agent a_x restocks the product directly from the specific factory $F(s_y)$ as a last resort. This requires a higher transportation cost, which is represented as $tcost(F(s_y), a_x, s_y)$.

Agent a_x may in turn receive requests for restocking a service s_y from another agent a_w. In this case, agent a_x only agrees to restock agent 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 in turn restock this amount from its provider. If the restock is agreed, 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 based on the stock required to be transferred $transf^t(a_x, s_y)$. This term represents the amount of products transferred to other agents.

Finally, the last state S'' represents the adaptation deliberation. In this state, the organization manager tries to distribute the services to agents in order to improve the profit of the organization. This profit is measured as the sales carried out by each agent a_x for each service s_y based 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 6.2.3, the manager must estimate the impact of these changes.

6.3.1 Estimation of the Adaptation Impact

As stated in Section 6.2.3.3, the impact estimation involves the benefits produced by the adaptation, the costs associated to the adaptation, and how this adaptation

would influence all the components of the organization. In this example, we focus on addition and deletion events regarding services, but other organizational dimension could also be changed [11].

The addition of a new service in an agent a_x implies the reduction of the maximum stock of the rest of the services provided by this agent, from $\frac{SMAX}{n}$ to $\frac{SMAX}{n+1}$, with n being the number of services provided by the agent and $SMAX$ being a constant defined for all the agents. The effect of this is estimated based on 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, producing 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 had been 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 the 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 had been included. 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 the 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 the factory, which represents the worst 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 loss for these agents. Finally, the addition of a new service has an associated fixed cost for setting up this service, which can be represented as $up(a_x, s_n)$. By aggregating all this information, the impact of adding a new service s_n to an agent a_x is represented as $I_A(a_x, s_n)$:

$$I_A(a_x, s_n) =$$

$$P(estimated^t(a_x, s_n)) - \sum_{s \in S^t(a_x)} \big(P(o\_sales^t(a_x, s)) + P(o\_transf^t(a_x, s)\big) -$$

$$- \sum_{a_z \in A^t} P(o\_sales^t(a_z, s_n)) - up(a_x, s_n)$$

In contrast to the addition of a service, the deletion of a service implies increasing the maximum stock of the rest of the services, from $\frac{SMAX}{n}$ to $\frac{SMAX}{n-1}$, with n being 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 has 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 a specific profit associated to it $P(estimated^t(a_x, s_y))$.

Furthermore, if an agent had not provided a service s_p, 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 not have been carried out. Therefore, the specific profit associated to the deletion of a service can be estimated depending on whether or not the service s_p has been restocked from factory.

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 by these agents. Finally, the deletion of a service has an associated fixed cost for turning off this service, which can be represented as $off(a_x, s_p)$. Therefore, the impact of deleting a service s_p that is already being provided by agent a_x is represented as $I_D(a_x, s_p)$:

$$I_D(a_x, s_p) =$$

$$\sum_{s \in S^t(a_x)} P(estimated^t(a_x, s)) - \big(P(sales^t(a_x, s_p)) + P(transf^t(a_x, s_p))\big) -$$

$$- \sum_{a_z \in A^t} P(o\_sales^t(a_z, s_p) - off(a_x, s_p)$$

Depending on the number of services provided by each agent, the manager considers the possibility of adding a service if the agent provides less than δ services and none of the services provided are restocked with more that what is actually needed. This would mean that it could be beneficial for the agent to add another 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. This would mean that it could be beneficial to delete some of the current services in order to increase the stock capacity of this highly demanded service. In order to deal with how all the information required for adaptation deliberation is retrieved, in the following section we show how the monitoring mechanism based on event tracing is used.

6.3.2 Event Tracing Specification

By using event tracing, agents can publish, request, and cancel subscriptions dynamically in order to send and retrieve only the information that is interesting at each moment. To allow every agent to know the stock that is available in its stock providers, each agent a_x publishes 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* trace event. All the agents that are interested in receiving this information (i.e., the agents that have a_x associated 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:

$$TE.type = STOCK\_AVAILABLE \wedge TE.source = a_x, a_x \in P^t(a_z, s_y)$$

In order to manage the information required for adaptation deliberation of the organization, 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 from those agents that can add a new service (those that provide fewer than δ services) and have requested more restock than actually required. This is implemented by publishing the *RESTOCK* trace event in the state S'. By using this trace event each agent publishes the information of the restock that it is carrying out. Therefore, the manager is interested in receiving information from those agents that can add a new service and that have requested a restock amount that is higher than the one required:

$$TE.type = RESTOCK \wedge TE.source = a_x, |S^t(a_x)| < \delta \wedge TE.value > \tau \quad (1)$$

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, based on the stock that is still available. Therefore, this threshold is initially defined as: $\tau = \left(\frac{SMAX}{n} - stock(a_x, s_y)\right) - stock(a_x, s_y)$. For the deletion of services, the manager is interested in receiving information from those agents that are able to delete a service (those that provide δ services or more) and have sold all the stock of a service. We can use the *STOCK_AVAILABLE* trace event published by agents to obtain this information:

$$TE.type = STOCK\_AVAILABLE \wedge TE.source = a_x, |S^t(a_x)| \geq \delta \wedge TE.value \leq \sigma \quad (2)$$

Similar to the above trace event, we define a threshold that could be modified at run time. This threshold is initially defined as $\sigma = 0$, to represent the notification of any event of positive stock available.

Finally, the manager also needs to know how many restocks are carried out from factories in order to calculate the profit based on the transportation costs. This is represented as a *FACTORY_REQUEST* trace event that is published by agents and is sent when the agent carries out a request to an specific factory:

$$TE.type = FACTORY\_REQUEST \wedge TE.source = any$$

6.4 Evaluation

In this section, we analyze different experiments to measure the performance of the adaptation framework. For these experiments, we define an organization of agents, which can have zero, one, or two different stock providers. As we stated in Section 6.2.2, the agent manager is responsible for managing the organization dynamics. This agent determines whether or not changes are required, depending on the information obtained from providers through events. Depending on the conditions of the environment, subscribed events might not reveal relevant information that is

different from previous events.

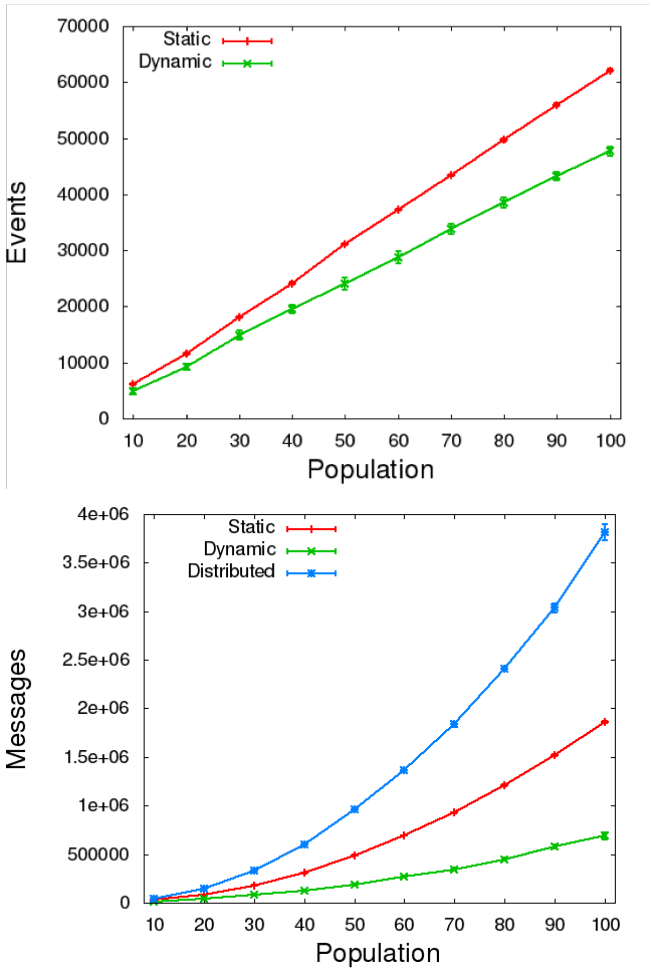


Figure 6.4: Messages/events received: (a) in the manager; (b) in the whole organization

To start, we would like to present some experiments regarding traffic reduction by using adaptive monitoring. Figure 6.4 (a) shows the number of events that are received by the manager in a static and a dynamic monitoring strategy over 50 iterations [1]. In this experiment, the thresholds σ and τ remain constant. As

[1] In the case of static monitoring, events are represented as messages.

can be observed, the number of events received is greater without using event tracing, and the differences between the two approaches becomes greater as the population of agents increases. This proves that the information that needs to be monitored in dynamic systems may be different throughout the organization's life span. Therefore, 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 needed to be monitored must be specified at design time, and therefore, a lot of information is transferred that is finally not used by the manager. Figure 6.4 (b) shows the performance of the three approaches based on the messages exchanged in the whole organization. In this experiment, it can be observed that dynamic monitoring clearly outperforms a static approach.

In the following subsections, we analyze how the thresholds τ and σ influence the profit of the organization and the number of events that are received in accordance with the user demand. These thresholds can be adapted in accordance with demand in order to reduce the number of events without losing profit. As an example, in a stable scenario in which the user demand is similar over several iterations, the reception of events may not be relevant unless the demand changes significantly. Therefore, the initial threshold can be extended to a range of acceptable values. In contrast, in a scenario in which the user demand is more variable, this range should be tighter because the information provided by new events may be quite different from the previous events. What is more, these restrictions could be different for all the agents of the organization.

An average value associated to an event is calculated by considering the last values. The number of values that are taken into account to obtain this average defines the estimation period length ϕ. A threshold range ρ is also defined as a parameter that determines the allowed fluctuation of the threshold of the monitored event. Therefore, events whose values are ranged in the interval between $[\phi - \rho, \phi + \rho]$ will not be retrieved.

In the experiments described in the following subsections, we test the performance of

an adaptive strategy. Each time-step when a *RESTOCK* or a *STOCK_AVAILABLE* event of a specific agent is not received, the values ρ and ϕ are increased; when an event is received, these values are decreased. We also test the performance of different static threshold ranges (from $\rho = 0$ until $\rho = 20$) for two different estimation period lengths ($\phi = \{5, 10\}$) as well as the performance of a classic monitoring strategy in which the thresholds are as defined initially.

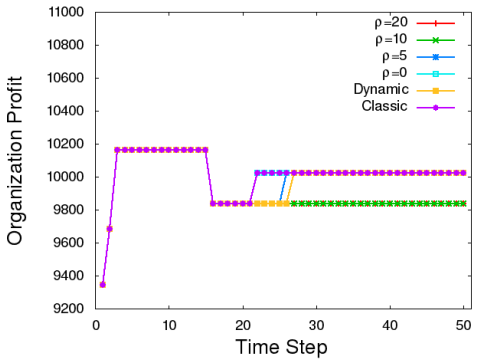
We test these strategies in different scenarios with different patterns of user demand: (1) when demand changes at a specific moment; (2) when demand changes progressively over a period of time; (3) when demand remains stable; (4) when demand changes slightly; and (5) demand changes quickly.

6.4.1 Specific change in demand

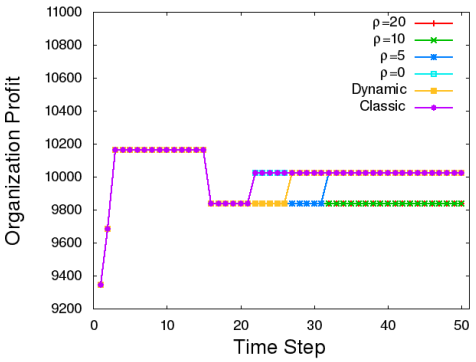
The first experiment represents a scenario in which the user demand remains constant until the time-step $t = 15$, when a change in the demand occurs and from then on remains constant. Figure 6.5 shows the profit of the organization and the events received for the different monitoring strategies. Figures 6.5(a) and 6.5(b) represent the profit at each time-step achieved for an estimation period length of $\phi = 5$ and $\phi = 10$, respectively. Figures 6.5(c) and 6.5(d) represent the number of accumulated events that are received at each time-step.

It can be observed that the threshold range and the estimation period length influence the organization's performance. The lower the value of ρ, the shorter the time period required by the organization to react to the change. This occurs because changes in the demand cause the threshold range to be exceeded earlier when this range is tight. For the highest values of ρ (10 and 20), the change in the demand is not large enough to receive the event. As a result, the organization does not adapt and a higher profit is not achieved. The classic strategy carries out the adaptation earlier than other strategies because this strategy retrieves all the subscribed events that exceed the initial thresholds. The problem with this strategy is that it receives a similar

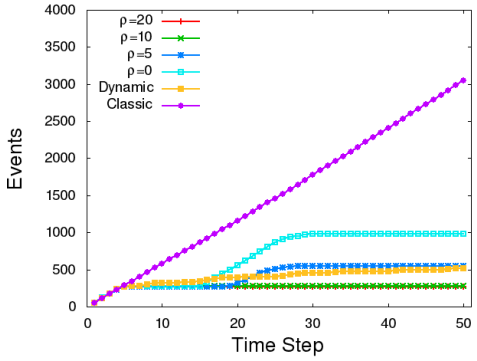
number of events every time-step, even when the demand remains constant (from $t = 15$ on). In contrast, the rest of the strategies detect that the demand stabilizes and the number of events received also eventually stabilizes without lowering the profit of the organization.



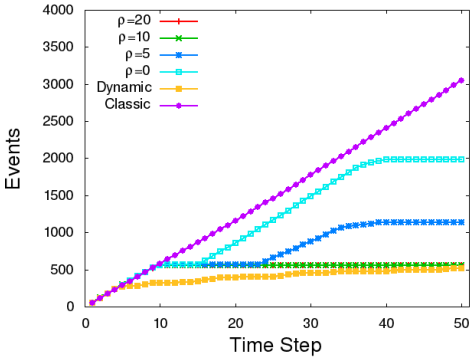
(a) Organization profit for $\phi = 5$



(b) Organization profit for $\phi = 10$



(c) Events received for $\phi = 5$



(d) Events received for $\phi = 10$

Figure 6.5: Profit of the organization and events received for a specific change in demand.

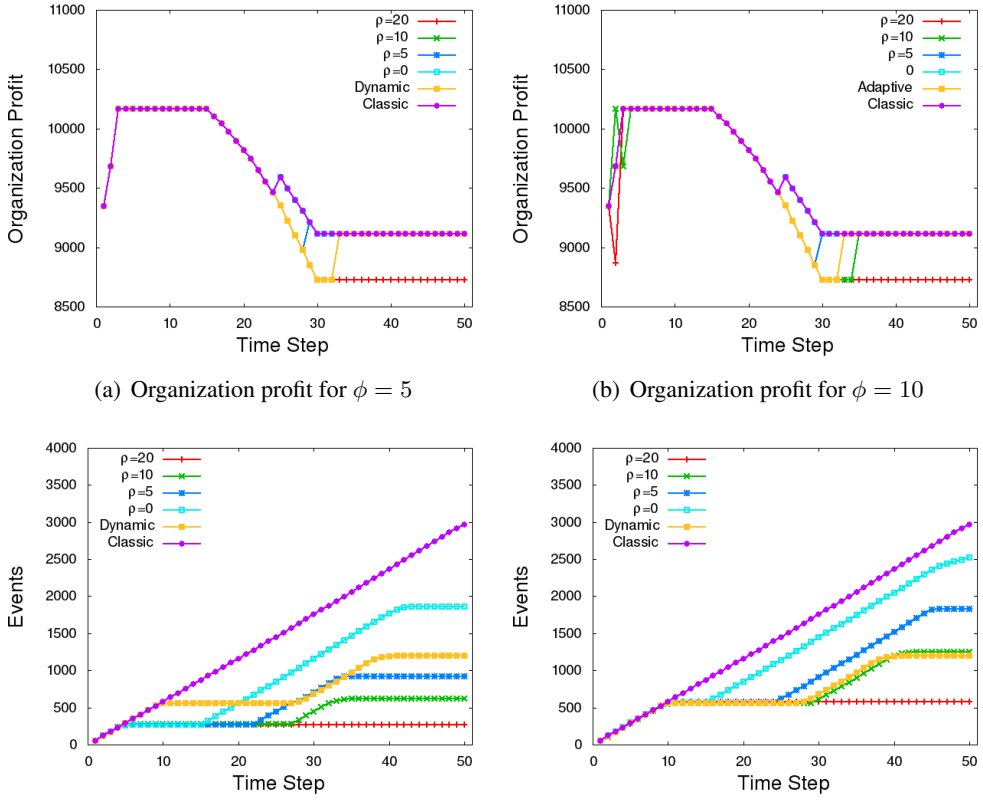
The influence of the estimation period length on the profit of the organization can be also observed A period of $\phi = 5$ causes some strategies (such as $\rho = 5$) to detect significant changes in demand earlier, and therefore to carry out the adaptation earlier as well. The number of events required to stabilize the system is also lower than for

$\phi = 10$. This is because a greater number of measures are considered to estimate the average value. It can be observed that the profit for the dynamic strategy is the same as the profit for the highest performing strategies. Despite the fact that some strategies can converge converge earlier than the dynamic strategy to the best profit, the dynamic strategy requires fewer events. As an example, the profit of the dynamic and the classic strategies is the same from $t = 27$ on; however, the dynamic strategy stops the reception of events when the demand stabilizes.

6.4.2 Progressive change in demand

The second experiment represents a scenario in which the user demand changes slightly and progressively from time-step $t = 15$ to time-step $t = 30$ (Figure 6.6). The objective of this experiment is to compare the moment at which the manager receives events and decides whether or not an adaptation is required. Similar to the first experiment, strategies with the lowest values of ρ receive events earlier. As in the first experiment, the dynamic strategy is able to reduce the values of ρ and ϕ when the demand is changing and to increase them when the demand is stable until this strategy stops receiving events from $t = 41$ on.

In summary, the lower the value of ρ, the earlier events are received because the threshold is exceeded earlier. Thus, a higher profit is achieved earlier. In addition, event though the number of events that are received is higher for strategies with low ρ values, this number eventually stabilizes. Similarly, low values of ϕ may obtain a higher profit earlier than larger values. Nevertheless, the number of events received is also important. As observed in this second experiment, different values of estimation period length may carry out the adaptation at the same moment by requiring a lower number of events. In other cases, the adaptation moment may be earlier (for $\rho = 10$), but this would not be too significant if the demand is stable over a long period of time.



(a) Organization profit for $\phi = 5$

(b) Organization profit for $\phi = 10$

(c) Events received for $\phi = 5$

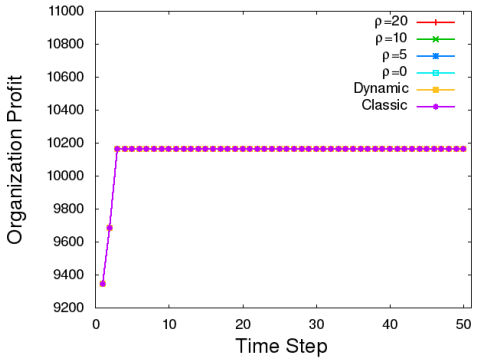
(d) Events received for $\phi = 10$

Figure 6.6: Profit of the organization and events received for a progressive change in the demand.

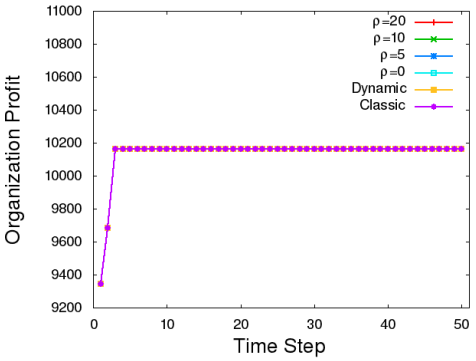
6.4.3 Stable demand

Figure 6.7 shows a scenario in which the demand is stable during the entire execution. This causes all the strategies to find the configuration with the highest profit in the former iterations. However, the classic monitoring strategy is constantly receiving events even though these events do not provide relevant information for adaptation. In contrast, dynamic strategy increases the values of ρ and ϕ when the demand stabilizes, which causes the dynamic strategy to stop receiving events from $t = 12$

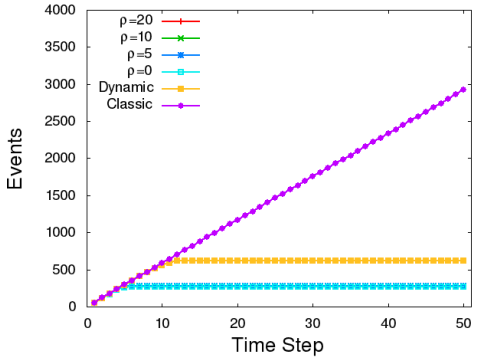
on.



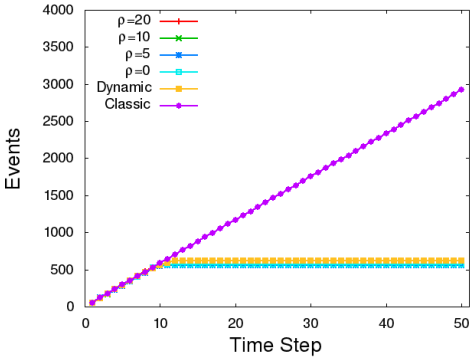
(a) Organization profit for $\phi = 5$



(b) Organization profit for $\phi = 10$



(c) Events received for $\phi = 5$



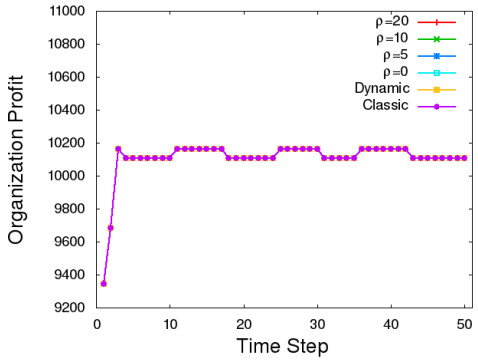
(d) Events received for $\phi = 10$

Figure 6.7: Profit of the organization and events received for stable demand.

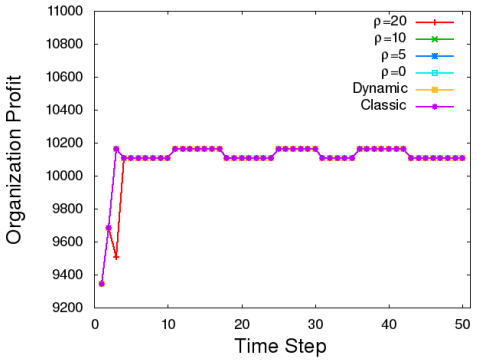
6.4.4 Slight change in demand

Figure 6.8 shows an experiment in which slight changes in demand occur at different moments, causing the profit of the organization to periodically slightly increase and decrease. However, these slight changes are not enough to require an adaptation. Therefore, all the strategies obtain the same profit, but those with the highest values

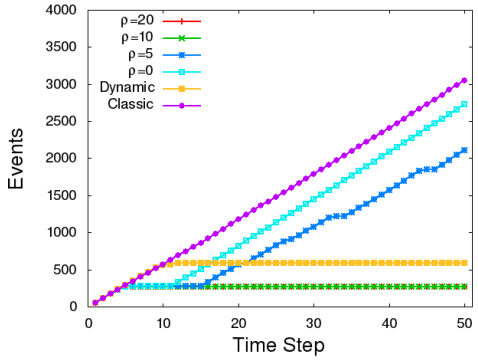
of ρ and ϕ require fewer events than other strategies. As an example, for a ρ value of 5, the number of events received stabilizes for $\phi = 10$ but not for $\phi = 5$. Similarly, for the same value of ϕ, in strategies with a ρ value of 20 or 10, the number of events received stabilizes, while for $\rho = 0$ the number of events increases. In scenarios of this type it is not useful to use a threshold range that is too tight because slight changes cause the reception of events that are not significant for adaptation. The dynamic monitoring strategy is able to fit scenarios both when the demand changes and when it stabilizes.



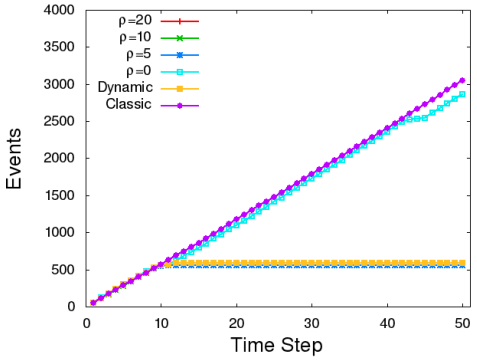
(a) Organization profit for $\phi = 5$



(b) Organization profit for $\phi = 10$

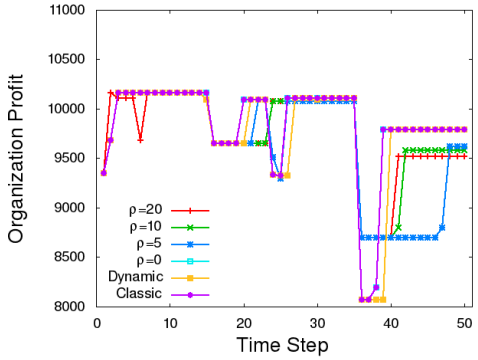


(c) Events received for $\phi = 5$

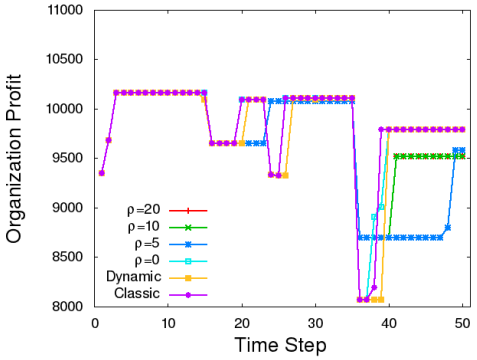


(d) Events received for $\phi = 10$

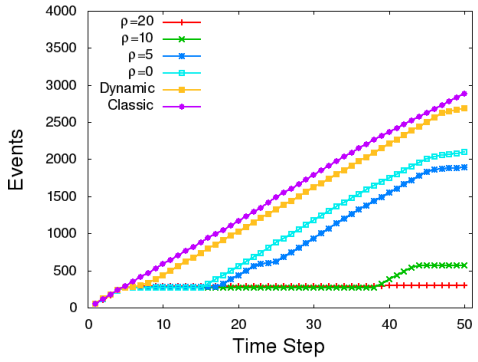
Figure 6.8: Profit of the organization and events received for slight changes in demand.



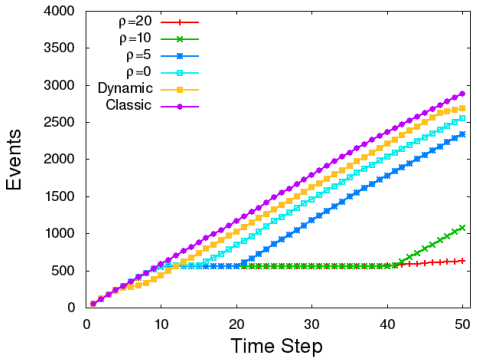
(a) Organization profit for $\phi = 5$



(b) Organization profit for $\phi = 10$



(c) Events received for $\phi = 5$



(d) Events received for $\phi = 10$

Figure 6.9: Profit of the organization and events received for quick change in demand.

6.4.5 Quick change in demand

For the last experiment, Figure 6.9 shows a scenario in which the demand changes quickly, and therefore the number of events increases while the demand is not stable. As the figure shows, the profit obtained by the dynamic strategy is practically the same as the profit obtained by the classic strategy, carrying out the adaptation only one or two time-steps later than the classic strategy. In addition, similar to the rest of the experiments, the dynamic strategy is able to stop the reception of events when the

demand stabilizes.

In summary, Table 6.1 shows the average profit during all the iterations and the average number of events received for all the strategies. We must point out that the dynamic strategy obtains a profit that is similar to the profit of the classic strategy; however, the dynamic strategy requires fewer events than the classic strategy. This compromise between profit and event traffic is more remarkable in fairly stable scenarios (stable and slight changes), in which the difference between the events received is quite significant and the profit obtained is the same. It can also be observed that, depending on the scenario, some static values for ρ and ϕ achieve a good profit, but these are only suitable for specific scenarios. In contrast, the dynamic monitoring strategy is able to adapt these values based on the user demand.

6.5 Related Work

In this section we discuss the most relevant work carried out in relation to tracing in multiagent systems (Section 6.5.1), indirect communication in multiagent systems (Section 6.5.2), and adaptation in agent organizations (Section 6.5.3).

6.5.1 Tracing in Multiagent systems

Event-tracing facilities in multiagent systems are usually conceived as debugging tools to help in the validation and verification processes. It is also common to use these tracing tools as help for those users who have to understand how the multiagent system works. Therefore, generated events are usually destined to be understood by a human observer.

Most of the current multiagent platforms provide their own tracing facilities through different kinds of support. As an example, JADE[21] and Zeus[45] provide support for keeping track of all the messages exchanged among agents; JASON[24, 25], JADEX[101], and Magentix [7, 109] provide support for examining the internal

Table 6.1: Average profit and events received for each scenario.

		Profit		Events	
Scenario	Strategy	$\phi = 5$	$\phi = 10$	$\phi = 5$	$\phi = 10$
Specific change in demand	$\rho = 20$	9909±48	9909±48	5±4	11±6
	$\rho = 10$	9909±48	9909±48	5±4	11±6
	$\rho = 5$	10002±44	9979±46	11±5	23±7
	$\rho = 0$	10018±41	10018±41	20±7	41±8
	classic	10018±41		61±1	
	dynamic	9998±44		10±4	
Progressive change in demand	$\rho = 20$	9376±177	9359±178	5±4	12±6
	$\rho = 10$	9511±142	9495±147	13±6	25±8
	$\rho = 5$	9542±131	9535±133	19±7	37±8
	$\rho = 0$	9566±127	9566±127	38±8	51±5
	classic	9566±127		59±1	
	dynamic	9511±140		24±8	
Stable demand	$\rho = 20$	10138±38	10138±38	5±4	11±6
	$\rho = 10$	10138±38	10138±38	5±4	11±6
	$\rho = 5$	10138±38	10138±38	5±4	11±6
	$\rho = 0$	10138±38	10138±38	5±4	11±6
	classic	10138±38		115±1	
	dynamic	10138±38		13±6	
Slight change in demand	$\rho = 20$	10106±37	10093±44	5±4	11±6
	$\rho = 10$	10106±37	10106±37	5±4	11±6
	$\rho = 5$	10106±37	10106±37	83±7	11±6
	$\rho = 0$	10106±37	10106±37	108±6	113±4
	classic	10106±37		121±1	
	dynamic	10106±37		11±6	
Quick change in demand	$\rho = 20$	9761±125	9764±126	6±5	13±6
	$\rho = 10$	9759±130	9764±126	11±6	21±8
	$\rho = 5$	9663±160	9634±168	38±8	46±7
	$\rho = 0$	9834±142	9833±135	42±7	51±5
	classic	9834±142		58±2	
	dynamic	9770±160		54±3	

state of agents and organizations; and JACK[4] provides support for monitoring communication among agents and other introspecting tools.

Apart from the tools provided the multiagent platforms themselves, there are other tracing facilities provided by third parties, such as Java Sniffer [120] (which provides support for reasoning about the messages exchanged) or ACLAnalyser [28] (which stores messages in a relational database in order to examine them). In addition, the results obtained by ACLAnalyser can be used together with data-mining techniques to debug multiagent systems [29, 111]. Tracking messages can also be used to help designers detect protocol violations and inconsistencies [98]

Although there are important works that are related to tracing facilitates, as we stated above, most of them are focused on human users rather than agents. Therefore, there is no standard general tracing mechanism that allows agents and other entities in the system to trace each other as they execute.

6.5.2 Indirect communication in Multiagent systems

The problem of providing support for additional ways of indirect communication and coordination has already been addressed by other authors using overhearing techniques. Overhearing is normally defined as an 'indirect interaction whereby an agent receives information for which it is not addressee' [83, 78, 54]. These techniques have already been used to maintain social situational and organizational awareness [107], to allow team organization [84, 83], to monitor teams in a non-intrusive way [78], and to develop advising systems [6, 39].

Most of the work in overhearing is modeled and implemented using message broadcasting. However, using broadcasting to perform overhearing is contradictory. This is because the definition of overhearing (and the definition of the overhearer role defined for multi-party dialogues [91]) implies that the sender is not always aware of who is receiving its messages, apart from the specified receivers.

The authors in [124] claim that the environment is a first-class abstraction that is complementary to agents in the system. At the same time, the environment is independent from the agents since it provides the surrounding conditions for agents to exist, as well as an exploitable design abstraction for building multiagent system applications. As pointed out in [100] and [110], the environment should give support to both direct and indirect interaction in multiagent systems, and therefore, overhearing should be managed by the environment in order to close the gap between the overhearer and the sender. In contrast to current approaches for overhearing, our proposal considers an indirect reception of agent-to-agent messages and also uses the tracing facilities.

6.5.3 Adaptation in agent organizations

For adaptation in agent organizations, monitoring the behavior of the organization is a crucial phase in determining that an adaptation is required [10]. Some current approaches define the monitoring process as an automatic response when changes occur. Other approaches provide an implicit mechanism for reasoning about the current state of the organization.

MACODO [125] is a middleware that provides support for the management of organization adaptation. In this approach, adaptation is triggered by external events (e.g., when an agents stops playing a role) and by other changes in the environment. Laws are specified to trigger the adaptation when these are satisfied. However, this specification is carried out at design time and cannot be changed at run time. Similar to this approach, some approaches specify adaptation by means of triggers that cannot be modified at run time [108, 92, 93, 123, 71].

Other works provide a more proactive decision mechanism for detecting that an adaptation is required, such as 2-LAMA [41, 42]. In 2-LAMA, the monitoring process is carried out by several agents, who perceive information about a subset of agents and share the information in order to make decisions. However, the

information that needs to be shared is also specified at design time and cannot be changed while the system is running. Similarly, the adaptation support provided by Moise [72] allows the implementation of agents that are in charge of determining when an adaptation is required. Nevertheless, the monitoring mechanisms are not defined at the adaptation approach level and must be implemented by the system designer (providing his own methods and tools). A similar issue associated to the static specification of the information that is monitored appears in other works such as [52, 30, 31].

The work of Kota et al. [79, 80] provides a self-adaptive model for task-solving environments in which each pair of agents is able to evaluate and change its relationships. In this approach, the information that is monitored can change during execution since it depends on the current relationships of each agent. However, this dynamic monitoring is only valid for task-solving domains of this kind, in which other organizational dimensions cannot be considered. Similar to Kota's approach, the work presented in [63] is specifically focused on changes in the relationships.

The restriction of considering the internal adaptation logic and the information to be monitored as predefined at design time forces systems to be designed in which the requirements for adaptation must be known in advance. This impedes the development of applications in which these requirements may not be specifically known or which could even be different throughout the organization's life-span. As stated in [5], adaptive systems may cause monitoring requirements to also change. Therefore, an adaptive approach should apply not only to the behavior and structure of the system but also to the design of the monitoring system [106], especially when dealing with the management of complex systems over long periods of time. Static mechanisms that do not consider changes regarding which information has to be monitored may be useful in small application domains with a priori, well-known organizational structures; however, they would not be suitable for large-scale or complex systems. As the number of agents in the organization and their complexity grow, much more information is exchanged among agents. Most of this information

might not be useful at every moment of the execution and only contributes to considerably increasing the traffic in the system. This is especially critical for approaches in which a middleware or centralizing entity is responsible for adaptation deliberation or implementation (e.g., as in [43]). Also, the internal logic of agents can become very complex and costly.

In contrast to the above approaches, our proposal uses event tracing as a dynamic monitoring mechanism to allow agents to determine at run time the events that are received during the execution. In the experiments presented, the reception of the events depends on the values of ρ and ϕ, which change according to whether the demand is changing or stable. In addition, the amount and type of information that each agent has to deal with depend on the actual state of the organization and changes as the system changes.

6.6 Conclusions

Monitoring an agent organization becomes essential when determining whether or not an adaptation is required at a specific moment. Most monitoring techniques for agent organizations that can be found in the literature either rely on predefined rules (which cannot be changed at run time), or they assume that the information that has to be monitored does not change as the system executes. Consequently, these approaches are only applicable in small domains that have a small number of agents and a priory, well-known organizational requirements; they 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 at each moment. This enables the organization to adapt to these changes. This monitoring is complemented by an adaptation module, which allows the sequence of the most promising changes to be obtained. This module also determines the potential improvement and cost of carrying out these

changes so that the organization can adapt to the current requirements.

The experimental results show how the performance of the agent organization improves when using the proposed framework for adapting to changes in service demands. On the one hand, the scalability of the system is improved by reducing the traffic when the information that is retrieved is determined at run time. On the other hand, when the adaptation logic is also changed at run time according to the state of the system (in our case, the demand), the reception of events can stabilize without losing profit.

As future lines of research, we plan to include more thorough studies on agent organizations, not only considering adaptation to changes in services provided by each agent, but also considering changes in the agent population or in the relationships among these agents. We also plan to include strategies for improving monitoring requirements based on past experience.

Acknowledgements

This work has been supported by projects TIN2011-27652-C03-01 and TIN2012-36586-C03-01.

Part III

Discussion

CHAPTER

7

General Discussion of the Results

In this chapter, the main results achieved in this thesis are discussed. Experimental results obtained by using event tracing in the field of real-time systems are commented in Section 7.1. Section 7.2 summarizes the list of requirements to be taken into account in order to develop an event tracing mechanism which any entity in a multiagent system can use as an appropiate mechanism for indirect communication. Sections 7.3 and 7.4 respectively discuss the TRAMMAS abstract model and the TRAMMAS architecture proposed in this thesis to integrate event tracing into a multiagent platform. The proposed model and architecture were integrated into two multiagent platforms. Section 7.5 discusses the main aspects of the incorporation of TRAMMAS in the PANGEA multiagent platform. Section 7.6 summarizes the main aspects of the integration of TRAMMAS in the multiagent platform Magentix2 and it also presents a brief discussion on the main results obtained using event tracing in order to detect important changes which may require the multiagent system to adapt.

7.1 Results on Event Tracing in Real-Time Systems

Chapter 2 presents an empirical study which compared different scheduling policies running over the real-time operating system RT-Linux [20]. The study was carried out using the framework described in Section 2.3. The event tracing support incorporated to the real-time operating system kernel, as described in [117], was used to extract information from running applications in order to compare their behavior and performance using different scheduling policies under different load conditions. Other previous results obtained using this framework were also published in [32] and [33].

The study in Chapter 2 was aimed to determine to which extent different scheduling policies, well known in the area of Real-Time Systems, are affected by the presence of gain time in the system, depending on the number of tasks, the amounts of hard and soft load and the percentage of gain time in hard tasks. Results were publisehd in [37] and they show that, in general, the fact that hard tasks consume less

execution time than their estimated WCETs (which in turn produces the availability of gain time) considerably reduces the performance benefit of using some of the most representative scheduling policies for soft tasks in fixed-priority preemptive systems compared to serving the soft tasks in background, which is considered the theoretical worst case. This is also true even for those policies that are specifically designed to efficiently reclaim and use gain time. Under some conditions, this performance benefit is so small, or even negative, that the use of a specific scheduling policy for soft tasks becomes questionable.

7.2 The Tracing Process and the Tracing System Requirements

The first steps in the development of a general event tracing system which agents could use as an effective information source were to define the tracing process and those entities which take part in it, as well as to identify the requirements and needs of such tracing system. Both steps were described and published in [34], which is included as Chapter 3 of this thesis.

As explained in Chapter 3, from the point of view of the tracing process, the multiagent system is seen as a set of entities which are susceptible of generating and/or consuming trace events. These entities (*tracing entities* from this point) may be individual agents or any kind of agent aggregation supported by the multiagent platform. Furthermore, the multiagent platform itself and its components can also be seen as tracing entities.

The study in Chapter 3 identified the requirements and proposed them to be classified in the following taxonomy:

- **Functional requirements:** Requirements related to which entities in a multiagent system can generate and/or receive trace events, as well as the way in which trace events are offered by and delivered to entities in the multiagent

system and the time at which information has to be generated and delivered to entities.

- **Efficiency requirements:** A minimal set of requirements related to the efficiency of a tracing system for multiagent systems, in order to make this system realizable and useful. These requirements focus on dealing with the potential overload and loss of decentralization which an event tracing support can cause to the multiagent system itself and to the entities in it.

- **Security requirements:** Requirements related to the ownership, privacy and security issues of the information which may be exchanged by entities in the multiagent system and how these entities have to exchange their tracing information taking these issues into account.

7.3 The TRAMMAS Model

Considering the requirements described in Chapter 3, an abstract, platform independent model for event tracing in multiagent systems was developed. The TRAMMAS model was published in [36], which is included in this thesis as Chapter 4.

According to the model, trace events can be classified in two main groups attending to the source entity which generates them: *domain independent* trace events, which are generated by the multiagent platform and thus, they can be present in any multiagent system, and *domain dependent* trace events, usually intended to give support to specific needs of the multiagent system.

The model also defines the concept of *tracing entity* to identify those entities in the multiagent system which are susceptible of generating and/or receiving tracing information during their lifetime. Inspired by the *Agents&Artifacts* taxonomy published in [97], the following different tracing entities are considered in the model:

- **Agents:** Autonomous, proactive entities that encapsulate control and are in charge of the goals/tasks that altogether define and determine the whole multiagent system behavior.

- **Artifacts:** Passive, reactive entities in charge of the services and functions that make individual agents work together in a multiagent system.

- **Aggregations:** Both agents and artifacts can associate and generate or receive tracing information as a single entity. The multiagent platform is seen as an aggregation of agents and artifacts and therefore, elements of the multiagent platform are also susceptible of generating and receiving trace events as any other element in the multiagent system.

Tracing entities can play two different *tracing roles* in the tracing process: *Event Source* (*ES*) and *Event Receiver* (*ER*), depending on if they are generating or receiving trace events. A third, special role is also considered in the model: the *Trace Manager* (*TM*). The TM role is responsible for controlling and coordinating the entire tracing process: registering tracing entities and event types, as well as giving support to the selective event tracing and security models. The TM role can be played by a single entity or by a set of different entities in the multiagent platform at the same time, acting coordinately, even in different nodes of the multiagent system. ER entities are provided with a special mailbox, called *stream* (*STRM*), where trace events are stored before the ER retrieves them. Both in on-line and off-line tracing, ER entities can control the amount of resources comsumed by the tracing process by limitating the size of their stream.

Selective event tracing and reception are supported in the model through a *publish/subscribe* protocol: ES entities publish at run time the tracing information they can provide and ER entities subscribe to those trace events they are interested in and unsibscribe from them when they are not, also at run time. In this way, the possible overhead which tracing information could cause to the multiagent system is reduced as much as possible.

The model also provides an authorization mechanism by means of which ES entities specify which ER entities are allowed to receive their trace events (*direct authorization*) and also by which ER entities which are authorized to receive trace events from certain ES entity can also authorize other entities to receive the same trace events (*authorization by delegation*). The TM maintains an authorization graph for each event type which is being offered by each ES. This graph is dynamic and changes at run time, as ES entities add or remove authorizations.

Figure 7.1, extracted from [36], shows the interactions among the different tracing roles and how trace events are exchanged among them.

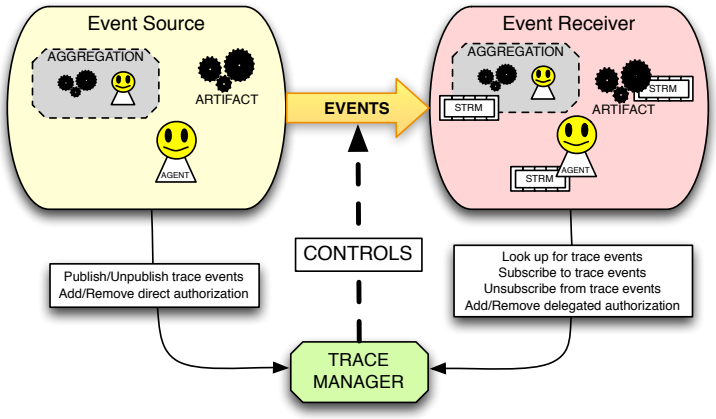


Figure 7.1: Interaction between the different tracing roles in the TRAMMAS model.

7.4 The TRAMMAS Architecture

Chapter 4 describes the TRAMMAS generic achitecture, designed to be integrated within a generic multiagent platform to allow tracing entities in the model to exchange trace events. This architecture follows a service-oriented approach. ES entities offer their tracing information as *tracing services*, which they have to publish. ER entities can request a tracing service and, if they are authorized to do so, they will

start receiving trace events of that service, when such events are produced.

In the proposed architecture, the domain independent tracing services are offered by the multiagent platform. This requires the multiagent platform to be instrumented at the source code level, which means that the platform source code has to be available in order to incorporate the event tracing support. However, this intrumentation presents two main advantages. On the one hand, it makes the tracing system more efficient and reliable. On the other hand, it makes certain internal information available as tracing services (for instance, information regarding agents lifecycle).

The architecture establishes that the TM role in the abstract model is to be played by the multiagent platform itself, in the form of an extra component, also called the Trace Manager. This component may be designed as a single component or a set of components, which may also be distributed, depending on the particular platform. Internally, the TM component is formed by different modules, which are in charge of different functionalities. Figure 7.2, extracted from [36], shows how tracing entities interact with these modules when they have to register/unregister tracing entities and services, to request/cancel tracing services, or to add/remove authorizations. A detailed description of the TM component and its functionality is presented in [36].

7.5 Integration of TRAMMAS in the PANGEA Multiagent Platform

The TRAMMAS model and architecture proposed in this thesis were succesfully incorporated to the multiagent platform *PANGEA* [129] in order to improve the way in which entities and agents perceive each other and their environment. First results of this development were published in [38], which is included as Chapter 5 of this document.

The multiagent platform PANGEA has a great potential to create open systems, and more specifically, virtual agent organizations. The communication mechanisms

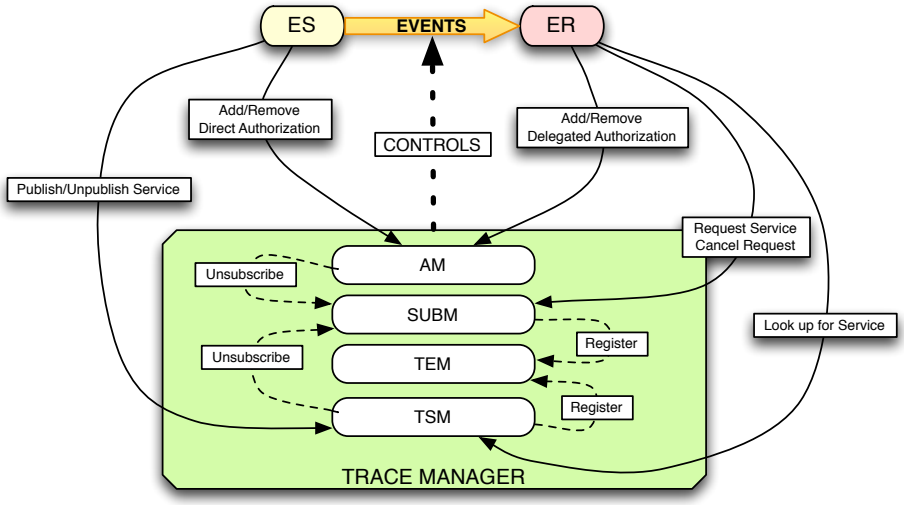


Figure 7.2: TRAMMAS architectural design of the tracing system and interactions between the TM component's internal modules and the rest of entities in the system, depending on their tracing roles.

offered by the platform are based on the IRC standard protocol [96], which allows for handling a large number of connections while also facilitating the incorporation of potential extensions. A Sniffer agent, which offers services to extract information from exchanged messages, is also available in the platform; however, this centralized approach dramatically limited the efficiency and scalability of multiagent systems in many circumstances.

Figure 7.3 shows how the tracing system was incorporporated to the platform: tracing services are offered by agents as web services and an *EventTracing Suborganization* was incorporated to the platform in order to carry out all of the functions of the TRAMMAS Trace Manager.

The case of study included in [38] shows how event tracing can contribute to the scalability and feasibility of virtual organizations by reducing the amount of unnecessary information which has to be transmitted and processed, while keeping agents' internal logic as simple as possible. Table 7.1 summarizes results of the

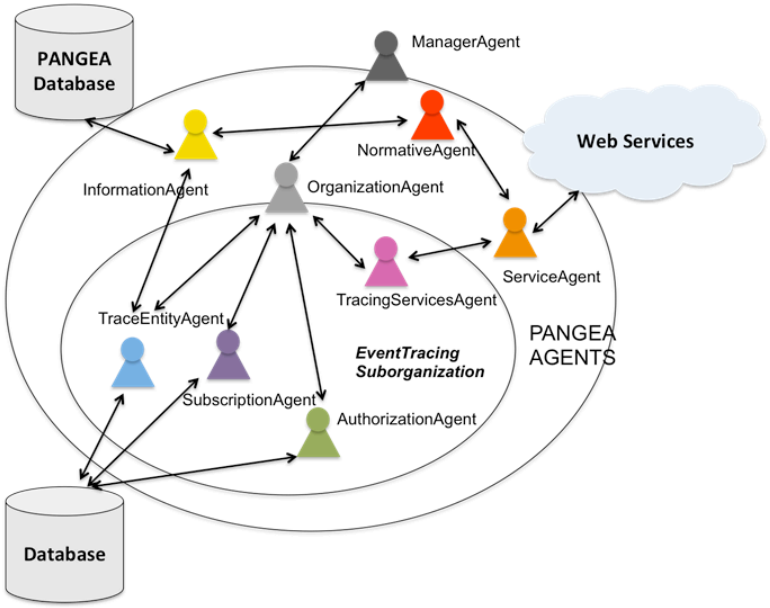


Figure 7.3: Incorporation of TRAMMAS based tracing support to the PANGEA multiagent platform.

comparison study between a broadcasting based and a event tracing based solutions carried out in [38]. In the worst case, when all entities in the system are interested in information regarding all of the rest of entities, theoretical costs of both studied techniques (broadcast and the *EventTracing Suborganization*) are in the same order. However, in the best case, when none of the entities in the system is interested in receiving any information from the rest, theoretical costs using broadcast are in the same order as those in the worst case, while theoretical costs of using event tracing is in the order of 0. These results in Section 5.5, show that event tracing provides a way for agents to coordinate, without them having to contact directly with each other. This reduces the amount of information exchanged among agents in the system to the minimum necessary, which makes the multiagent system more efficient and scalable.

Number of transmissions for n_{rem} situations		
	Best case	*Worst case*
Broadcast	$k_{remarkable} * (2 + n_{sens})$	$k_{remarkable} * (2 + n_{sens})$
EventTracing Suborganization	0	$k_{remarkable} * (2 + n_{sens})$

Table 7.1: Summary of best and worst case costs as a function of the number of N_{sens} agents for a constant number of remarkable situation ($K_{remarkable}$)

7.6 Integration of TRAMMAS in the Magentix2 Multiagent Platform: *Trace&Trigger*

The use of event tracing as an indirect communication mechanism for agents, as well as to improve their environmental knowledge, was also published in [13], which is included as Chapter 6 of this document. *Trace&Trigger*, the framework presented in [13], is a framework for the adaptation of agent organizations to important changes in their social or physical environment which may be produced at run time. The framework consists of a dynamic monitoring mechanism, which helps agents detecting adaptation requirements dynamically at run time, and an adaption assistant, which provides organizational agents with information related to the costs and benefits of carrying out an adaptation at any moment during the execution.

The framework was developed to run on the *Magentix2* multiagent platform [60, 115], a platform for open multiagent systems developed in Java. It makes use of three components: the virtual organization support for agents provided by the *THOMAS* architecture [105], the TRAMMAS based event tracing support incorporated to the platform and the *Reorganization Facilitator service* (*RF*) [8, 12]. Figure 7.4, taken from Chapter 6, shows the developed *Trace&Trigger* framework and how the different components of THOMAS and TRAMMAS interact with the Magentix2 platform and with running agents in the system.

The TRAMMAS tracing facilities were incorporated to Magentix2 by means of a specific agent (TM), which carries out all of the functions of the TRAMMAS Trace Manager. Agents have to send an ACL message to the TM agent whenever they want

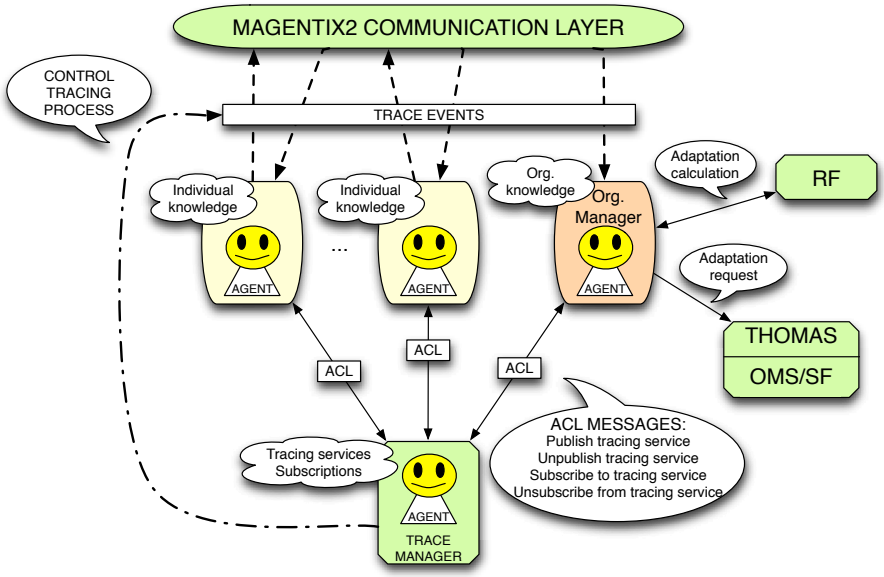


Figure 7.4: Trace&Trigger framework

to publish or unpublish their available trace services, and also when they want to subscribe to a trace service or to unsubscribe from it. The TM agent interacts with the Magentix2 communication layer so that trace events generated by an agent only cause information traffic if there is an agent interested in receiving them. The TM also interacts with the communication layer so that no trace event is received by an agent unless the agent has previously requested it. As a consequence, only the strictly necessary event tracing traffic is generated.

Section 6.3 presents a case of study based on a market domain, where a virtual organization of agents has to maximize the global profit that its agents make by selling products. In order to do so, each seller agent has to adapt the catalogue of products that it sells and the stock of each product that it has to maintain. The organization manager receives information from seller agents about sales, demand and so on. With this information from each seller agent, the organization manager decides if a seller has to stop offering a product or if it has to change the amount of

stocked units of a product because its sales are expected to go up/down in the market. This case of study was implemented to run over Magentix2 using the Trace&Trigger framework to help the virtual organization of agents adapting to changes in the virtual market, while maximizing the global amount of profit.

Experimental results in the virtual market domain, using synthetic load, are presented in Section 6.4. Different strategies to detect when the virtual organization has to change in order to adapt to variations in the market were considered. Table 7.1, extracted from Section 6.4, shows numeric results of all experiments, where both the final profit and the amount of exchanged messages for the five studied scenarios. As it can be appreciated, profit values obtained using dynamic adaptation based on event tracing are very similar to the ones obtained using a classical broadcasting (static) techniques, while considerably reducing the amount of exchanged information, not only when the product demand is stable, but also in dynamic scenarios where product demand varies in time.

Table 7.1: Average profit and events received for each scenario.

		Profit		Events	
Scenario	Strategy	$\phi = 5$	$\phi = 10$	$\phi = 5$	$\phi = 10$
Specific change in demand	$\rho = 20$	9909±48	9909±48	5±4	11±6
	$\rho = 10$	9909±48	9909±48	5±4	11±6
	$\rho = 5$	10002±44	9979±46	11±5	23±7
	$\rho = 0$	10018±41	10018±41	20±7	41±8
	classic	10018±41		61±1	
	dynamic	9998±44		10±4	
Progressive change in demand	$\rho = 20$	9376±177	9359±178	5±4	12±6
	$\rho = 10$	9511±142	9495±147	13±6	25±8
	$\rho = 5$	9542±131	9535±133	19±7	37±8
	$\rho = 0$	9566±127	9566±127	38±8	51±5
	classic	9566±127		59±1	
	dynamic	9511±140		24±8	
Stable demand	$\rho = 20$	10138±38	10138±38	5±4	11±6
	$\rho = 10$	10138±38	10138±38	5±4	11±6
	$\rho = 5$	10138±38	10138±38	5±4	11±6
	$\rho = 0$	10138±38	10138±38	5±4	11±6
	classic	10138±38		115±1	
	dynamic	10138±38		13±6	
Slight change in demand	$\rho = 20$	10106±37	10093±44	5±4	11±6
	$\rho = 10$	10106±37	10106±37	5±4	11±6
	$\rho = 5$	10106±37	10106±37	83±7	11±6
	$\rho = 0$	10106±37	10106±37	108±6	113±4
	classic	10106±37		121±1	
	dynamic	10106±37		11±6	
Quick change in demand	$\rho = 20$	9761±125	9764±126	6±5	13±6
	$\rho = 10$	9759±130	9764±126	11±6	21±8
	$\rho = 5$	9663±160	9634±168	38±8	46±7
	$\rho = 0$	9834±142	9833±135	42±7	51±5
	classic	9834±142		58±2	
	dynamic	9770±160		54±3	

CHAPTER

8

Conclusions and Future Work

A study of the state of the art in the field of multiagent systems revealed important limitations in current approaches when it comes to properly supporting agents' environmental and social knowledge. These limitations are partially due to the use of traditional approaches, usually based on agent messages, which fall short as indirect coordination and communication mechanisms for agents. Previous experience in the field of real-time systems proved that event tracing techniques could be succesfully used to extract information from critical and embeded systems at run time, causing a minimal interference in traced systems. This PhD thesis proposes the use of event tracing techniques to improve indirect communication and interaction among entities in a multiagent system and to serve as an appropiate environmental and social knowledge provider for them. In this way, high level social abstractions in the literature could be properly supported from the first stages in the design and development of multiagent systems, allowing these systems to develop their full potential.

A set of requirements to be taken into account when developing such an event tracing mechanism for entities in a multiagent system had to be compiled. These requirements address, not only functional aspects of the event tracing mechanism, but also efficiency and security aspects, in order to minimize the impact of such an event tracing mechanism in the feasibility of the instrumented multiagent system. From these requirements, the TRAMMAS abstract model was developed. Inspired by the *Agents&Artifacts* taxonomy in [97], the model lets any entity in the multiagent system play one or both tracing roles, *Event Source* (*ES*) and *Event Receiver* (*ER*), in order to exchange their information. Together with the model, a generic TRAMMAS architecture is also proposed in order to incorporate the concepts in the model to an actual multiagent platform. Tracing facilities are offered by the TRAMMAS architecture following a service-oriented paradigm, so that entities playing the ES role offer their tracing information as *tracing services* which are requested by those entities playing the ER role when they want to receive it in a similar way to which traditional services are offered and requested.

The proposed model and architecture were integrated in two multiagent platforms: PANGEA and Magentix2. The integration in Magentix2 was used in conjuntion with the THOMAS virtual organization support and the reorganization support provided by the multiagent platform itself in order to develop an adaptation support framework. This framework could be used by agents in a virtual organization in order to identify variations in their physical or social environment which may require them to change their behavior. A comparative study was carried out to compare different adaptation strategies in a virtual market domain. Experimental results using synthetic load show that event tracing can be very effective in reducing information traffic in the system with a minimal performance loss.

Contributions in this PhD thesis prove that event tracing can be used as an effective indirect mechanism communication for entities in a multiagent system. In combination with traditional agent messages, it has also proved to be a good social knowledge provider, more versatile and efficient than using only traditional messages. High level components in the multiagent system can take advantage of this improvement in the amount and quality of social and environmental knowledge, in order to acomplish their mission in the multiagent system with a minimal additional interference due to the event tracing process.

Regarding future lines of research, incorporating event tracing to the multiagent system developing process from its very early stages requires introducing trace events in multiagent system methodologies as an additional mechanism for indirect communication and interaction, at the same level as traditional messages. In this way, high level social and environmental aspects would benefit from an appropiate support from these early stages too and take part in this process as first class entities, without requiring developers to think of *ad hoc* solutions to incorporate them to multiagent systems.

One of the high level abstractions which would benefit from the additional knowledge provided by an event tracing support is the concept of norm, and its application in the so-called normative systems [47, 46]. In this way, we are also interested in

using the proposed model as a way to improve the development of the regulatory mechanisms needed in normative systems, as future work. Regulatory mechanisms are needed in order to guarantee a globally efficient coordination in open systems (i.e., ensuring that the different virtual corporate policies are fulfilled), but they need to take into account the impossibility of directly controlling (the majority of) the members of a virtual organization. These regulatory mechanisms assume that all the required information is available in order to control the agents' actions, but in most cases, part of this information is hidden, or simply not available on time. An event tracing approach, like the one proposed in this thesis, can facilitate the obtention of the required information for such systems, as it has been shown in other scenarios.

Bibliography

[1] (2005). *JACK Intelligent Agents Teams Manual*. Agent Oriented Software Pty. Ltd., P.O. Box 639, Carlton South, Victoria, 3053, AUSTRALIA.

[2] (2005). ZigBee specification.

[3] (2006). *Specification and Verification of Dynamics in Cognitive Agent Models*, Washington, DC, USA. IEEE Computer Society.

[4] (2008). *JACK Intelligent Agents Tracing and Logging Manual - Release 5.3*. Agent Oriented Software Pty. Ltd., P.O. Box 639, Carlton South, Victoria, 3053, AUSTRALIA.

[5] Abdu, H., Lutfiyya, H., and Bauer, M. A. (1999). A model for adaptive monitoring configurations. In *Proceedings of the 6th IFIP/IEEE IM Conference on Network Management*, pages 371–384.

[6] Aiello, M., Busetta, P., Donà, A., and Serafini, L. (2002). Ontological overhearing. In *ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, pages 175–189, London, UK. Springer-Verlag.

[7] Alberola, J., Mulet, L., Such, J., García-Fornes, A., Espinosa, A., and Botti,

V. (2007). Operating system aware multiagent platform design. *Fifth European Workshop On Multi-Agent Systems (EUMAS 2007)*, pages 658–667.

[8] Alberola, J. M., Julián, V., and García-Fornes, A. (2011). Cost-aware reorganization service for multiagent systems. In *Proc. 2nd Int. Workshop ITMAS11*, pages 60–74.

[9] Alberola, J. M., Julián, V., and García-Fornes, A. (2012). Multi-dimensional transition deliberation for organization adaptation in multiagent systems. In *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS-12)*, pages 1379–1380.

[10] Alberola, J. M., Julián, V., and García-Fornes, A. (2013a). Challenges for adaptation in agent societies. *Knowledge and Information Systems*, page In Press.

[11] Alberola, J. M., Julián, V., and García-Fornes, A. (2013b). Multi-dimensional adaptation in mas organizations. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, **43**(2), 622–633.

[12] Alberola, J. M., Julián, V., and García-Fornes, A. (2013c). Using cost-aware transitions for reorganizing multiagent systems. *Engineering Applications of Artificial Intelligence*, **26**(1), 63–75.

[13] Alberola, J. M., Búrdalo, L., Julián, V., Terrasa, A., and García-Fornes, A. (2014). An adaptive framework for monitoring agent organizations. *Information Systems Frontiers*, **16**(2), 239–256.

[14] Argente, E., Julián, V., and Botti, V. (2009a). Mas modeling based on organizations. *Agent-Oriented Software Engineering IX: 9th International Workshop, AOSE 2008 Estoril, Portugal, May 12-13, 2008 Revised Selected Papers*, pages 16–30.

[15] Argente, E., Julián, V., and Botti, V. (2009b). MAS Modeling based on Organizations. In *Post-Proceedings 9th International Workshop AOSE'08*, volume 5386, pages 16–30. Springer.

[16] Audsley, N., Davis, R., Burns, A., and Wellings, A. (1994). Appropriate mechanisms for the support of optional processing in hard real-time systems. In *11th IEEE Workshop on Real-Time Operating Systems and Software*, pages 23–27.

[17] B. Sprunt, L. S. and Lehozky, J. (1989). Aperiodic task scheduling for hard real-time systems. *The Journal of Real-Time Systems*, (1), 27–60.

[18] Banus, J. M., Arenas, A., and Labarta, J. (2002). An efficient scheme to allocate soft-aperiodic tasks in multiprocessor hard real-time systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications - Volume 2*, pages 809–815.

[19] Banus, J. M., Arenas, A., and Labarta, J. (2003). Dual priority algorithm to schedule real-time tasks in a shared memory multiprocessor. *Parallel and Distributed Processing Symposium, International*.

[20] Barabanov, M. (1997). *A Linux-based Real-Time Operating System*. Master's thesis, Institute of Mining and Technology, New Mexico.

[21] Bellifemine, F., Caire, G., Trucco, T., Rimassa, G., and Mungenast, R. (2007). *Jade administrator's guide*.

[22] Bernat, G. and Burns, A. (1999). New results on fixed priority aperiodic servers. In *IEEE Real-Time Systems Symposium*, pages 68–78.

[23] Bini, E. and Buttazzo, G. C. (2005). Measuring the performance of schedulability tests. *Real-Time Systems*, **30**, 129–154.

[24] Bordini, R. and Hübner, J. (2007). Jason: A java-based interpreter for an extended version of agentspeak. page 31.

[25] Bordini, R., Hübner, J., and Vieira, R. (2005). Jason and the golden fleece of agent-oriented programming. In G. Weiss, R. Bordini, M. Dastani, J. Dix, and A. Fallah Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of

Multiagent Systems, Artificial Societies, And Simulated Organizations, pages 3–37. Springer US.

[26] Bordini, R., Dastani, M., and Winikoff, M. (2007). Current issues in multi-agent systems development (invited paper). *Post-proceedings of the Seventh Annual International Workshop on Engineering Societies in the Agents World*, pages 38–61.

[27] Bosse, T., Lam, D., and Barber, K. (2008). Tools for analyzing intelligent agent systems. *Web Intelligence and Agent Systems*, **6**(4), 355–371.

[28] Botia, J., Hernansaez, J., and Skarmeta, F. (2004). *Towards an approach for debugging MAS through the analysis of ACL messages*, volume 3187/2004, pages 301–312. Springer Berlin / Heidelberg.

[29] Botia, J., Hernansaez, J., and Gomez-Skarmeta, A. (2007). On the application of clustering techniques to support debugging large-scale multi-agent systems. *Programming multi-agent systems*, **4411/2007**, 217–227.

[30] Bou, E., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2006). Adaptation of autonomic electronic institutions through norms and institutional agents. In *Engineering Societies in the Agents World. Number LNAI 4457*, pages 300–319. Springer.

[31] Bou, E., López-Sánchez, M., and Rodríguez-Aguilar, J. A. (2007). Towards self-configuration in autonomic electronic institutions. In *COIN 2006 Workshops. Number LNAI 4386*, pages 220–235. Springer.

[32] Búrdalo, L., Espinosa, A., García-Fornes, A., and Terrasa, A. (2006). Framework for the development and evaluation of new scheduling policies in rt-linux. In *OSPERT 2006*, pages 42–51.

[33] Búrdalo, L., Espinosa, A., Terrasa, A., and García-Fornes, A. (2007). Experimental results of aperiodic fixed-priority policies in rt-linux. In *Workshop*

on Operating Systems Platforms for Embedded Real-Time applications, pages 10–19.

[34] Búrdalo, L., Terrasa, A., García-Fornes, A., and Espinosa, A. (2009a). Supporting social knowledge in multiagent systems through event tracing. *Journal of Physical Agents*, **3**(3).

[35] Búrdalo, L., Terrasa, A., and García-Fornes, A. (2009b). Towards providing social knowledge by event tracing in multiagent systems. In *HAIS '09: Proceedings of the 4th International Conference on Hybrid Artificial Intelligence Systems*, volume 5572/2009 of *Lecture Notes in Computer Science*, pages 484–491, Berlin, Heidelberg. Springer-Verlag.

[36] Búrdalo, L., Terrasa, A., Julián, V., and García-Fornes, A. (2011). Trammas: A tracing model for multiagent systems. *Engineering Applications of Artificial Intelligence*, **24**(7), 1110–1119.

[37] Búrdalo, L., Terrasa, A., Espinosa, A., and García-Fornes, A. (2012a). Analyzing the effect of gain time on soft-task scheduling policies in real-time systems. *Software Engineering, IEEE Transactions on*, **38**(6), 1305–1318.

[38] Búrdalo, L., Terrasa, A., Julián, V., Zato, C., Rodríguez, S., Bajo, J., and Corchado, J. (2012b). Improving the tracing system in pangea using the trammas model. In J. Pavón, N. Duque-Méndez, and R. Fuentes-Fernández, editors, *Advances in Artificial Intelligence – IBERAMIA 2012*, volume 7637 of *Lecture Notes in Computer Science*, pages 422–431. Springer Berlin Heidelberg.

[39] Busetta, P., Donà, A., and Nori, M. (2002). Channeled multicast for group communications. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1280–1287, New York, NY, USA. ACM.

[40] Calandrino, J. M., Baumberger, D. P., Li, T., Hahn, S., and Anderson, J. H. (2007). Soft real-time scheduling on performance asymmetric multicore

platforms. In *IEEE Real Time Technology and Applications Symposium*, pages 101–112.

[41] Campos, J., López-Sánchez, M., and Esteva, M. (2009). 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.

[42] Campos, J., Esteva, M., Lopez-Sanchez, M., Morales, J., and Salamo, M. (2011). Organisational adaptation of multi-agent systems in a peer-to-peer scenario. *Computing*, **91**, 169–215.

[43] Carvalho, G., Almeida, H., Gatti, M., Vinicius, G., Paes, R., Perkusich, A., and Lucena, C. (2006). Dynamic law evolution in governance mechanisms for open multi-agent systems. In *2nd Workshop on Software Engineering for Agent-oriented Systems*.

[44] Chodrow, S., Jahanian, F., and Donner, M. (1991). Run-time monitoring of real-time systems. In *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, pages 74–83.

[45] Collis, J., Ndumu, D., Nwana, H., and Lee, L. (1998). The zeus agent building tool-kit. *BT Technology Journal*, **16**(3), 60–68.

[46] Criado, N., Argente, E., and Botti, V. J. (2013). THOMAS: an agent platform for supporting normative multi-agent systems. *J. Log. Comput.*, **23**(2), 309–333.

[47] Criado, N., Argente, E., Noriega, P., and Botti, V. (2014). Reasoning about norms under uncertainty in dynamic environments. *International Journal of Approximate Reasoning*, **55**(9), 2049 – 2070. Weighted Logics for Artificial Intelligence.

[48] Davis, R. and Wellings, A. (1995). Dual priority scheduling. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 100–109.

[49] Davis, R. I. (1994). Dual priority scheduling: A means of providing flexibility in hard real-time systems. Technical Report YCS230.

[50] Davis, R. I. (1995). *On exploiting spare capacity in hard real-time systems*. Ph.D. thesis, Department of Computer Science, University of York.

[51] Davis, R. I., Tindell, K., and Burns, A. (1993). Scheduling slack time in fixed priority preemptive systems. In *IEEE Real-Time Systems Symposium*, pages 222–231.

[52] DeLoach, S., Oyenan, W., and Matson, E. (2008). A capabilities-based model for adaptive organizations. *Autonomous Agents and Multi-Agent Systems*, **16**, 13–56.

[53] Deokar, A. V. and El-Gayar, O. F. (2011). Decision-enabled dynamic process management for networked enterprises. *Information Systems Frontiers*, **13**(5), 655–668.

[54] Dignum, F. and Vreeswijk, G. (2004a). *Advances in Agent Communication*, volume 2922 of *Lecture Notes in Computer Science*, chapter Towards a Testbed for Multi-party Dialogues, pages 212–230. Springer Berlin / Heidelberg.

[55] Dignum, F. and Vreeswijk, G. A. (2004b). Towards a test bed for multi-party dialogues. In F. Dignum, editor, *Workshop on Agent Communication Languages*, volume 2922 of *Lecture Notes in Computer Science*, pages 212–230. Springer Berlin / Heidelberg.

[56] Dignum, V., editor (2009). *Multi-agent Systems: Semantics and Dynamics of Organizational Models*. IGI Global.

[57] Dignum, V., Dignum, F., and Sonenberg, L. (2004). Towards dynamic reorganization of agent societies. In *In Proceedings of Workshop on Coordination in Emergent Agent Societies*, pages 22–27.

[58] Esparcia, S. and Argente, E. (2011). Formalizing Virtual Organizations. In *3rd Int. Conf. on Agents and Artificial Intelligence (ICAART 2011)*, volume 2, pages 84–93. INSTICC.

[59] Faggioli, D., Bertogna, M., and Checconi, F. (2010). Sporadic server revisited. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 340–345.

[60] Fogués, R. L., Alberola, J. M., Such, J. M., Espinosa, A., and García-Fornes, A. (2010). 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.

[61] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, **15**(3), 200–222.

[62] García-Fornes, A., Terrasa, A., and Botti, V. (1997). Planificación de tareas aperiódicas en sistemas de tiempo real estricto. *NOVATICA*, **Septiembre**, 22–30.

[63] Gaston, M. E. and desJardins, M. (2005). Agent-organized networks for dynamic team formation. In *Proceedings of the 4th International Joint Conference on Autonomous agents and Multiagent Systems (AAMAS05)*, pages 230–237.

[64] Gaujal, B., Navet, N., and Migge, J. (2003). Dual-priority versus background scheduling: A path-wise comparison. *Real-Time Systems*, (25), 39–66.

[65] Ghazalie, T. M. and Baker, T. P. (1995). Aperiodic servers in a deadline scheduling environment. *Real-Time Syst.*, **9**(1), 31–67.

[66] Goossens, J. and Macq, C. (1999). Performance analysis of various scheduling algorithms for real-time systems composed of aperiodic and periodic tasks. In *CISSAS'99*.

[67] Gregori, M. E., Cámara, J. P., and Bada, G. A. (2006). A jabber-based multi-agent system platform. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1282–1284, New York, NY, USA. ACM.

[68] Gruver, W. (2004). Technologies and applications of distributed intelligent systems.

[69] Guessoum, Z., Ziane, M., and Faci, N. (2004). Monitoring and organizational-level adaptation of multi-agent systems. In *In Proc. of AAMAS 2004*, pages 514–521. ACM Press.

[70] Helsinger, A., Thome, M., Wright, T., and Technol, B. (2004). Cougaar: a scalable, distributed multi-agent architecture. *2004 IEEE International Conference on Systems*.

[71] Hoogendoorn, M. and Treur, J. (2006). An Adaptive Multi-agent Organization Model Based on Dynamic Role Allocation. In *Proc. of the IAT '06*, pages 474–481.

[72] Hübner, J. F., Sichman, J. S., and Boissier, O. (2004). Using the MOISE+ for a Cooperative Framework of MAS Reorganisation. In *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (SBIA'04)*, volume 3171, pages 506–515.

[73] IEEE (2004). *1003.1, 2004 EDITION IEEE Standard for Information Technology Portable Operating System Interface (POSIX)*.

[74] Kalt, C. (2000a). Internet Relay Chat: Architecture. RFC 2811 (Informational).

[75] Kalt, C. (2000b). Internet Relay Chat: Channel Management Protocol. RFC 2811 (Informational).

[76] Kalt, C. (2000c). Internet Relay Chat: Client Protocol. RFC 2812 (Informational).

[77] Kalt, C. (2000d). Internet Relay Chat: Server Protocol. RFC 2813 (Informational).

[78] Kaminka, G. A., Tambe, D. V. P. M., Pynadath, D. V., and Tambe, M. (2002). Monitoring teams by overhearing: A multi-agent plan-recognition approach. *Journal of Artificial Intelligence Research*, **17**.

[79] Kota, R., Gibbins, N., and Jennings, N. R. (2009). Decentralised structural adaptation in agent organisations. In *Organized Adaptation in Multi-Agent Systems*, pages 54–71.

[80] Kota, R., Gibbins, N., and Jennings, N. R. (2012). Decentralised approaches for self-adaptation in agent organisations. In *ACM Transactions on Autonomous and Adaptive Systems*, pages 1–28.

[81] Lam, D. and Barber, K. (2005a). Comprehending agent software. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 586–593, New York, NY, USA. ACM.

[82] Lam, D. and Barber, K. (2005b). Debugging agent behavior in an implemented agent system. In R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors, *Programming Multi-Agent Systems, Second International Workshop ProMAS 2004, Selected Revised and Invited Papers*, volume 3346 of *Lecture Notes in Computer Science*, pages 104–125. Springer.

[83] Legras, F. and Tessier, C. (2003). Lotto: group formation by overhearing in large teams. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 425–432, New York, NY, USA. ACM.

[84] Legras Onera, F. (2002). Using overhearing for local group formation. In *AAMAS '02 Workshop 7 "Teamwork and Coalition Formation"*, pages 8–15.

[85] Lehoczky, J. P. and Ramos-Thuel, S. (1992). An optimal algorithm for scheduling soft-aperiodic tasks fixed-priority preemptive systems. pages 110–123.

[86] Lehoczky, J. P., Sha, L., and Strosnider, J. K. (1987). Enhanced aperiodic responsiveness in hard real-time environments. pages 261–270.

[87] Li, C. and Li, L. (2012). Collaboration among mobile agents for efficient energy allocation in mobile grid. *Information Systems Frontiers*, **14**(3), 711–723.

[88] Luck, M., McBurney, P., Shehory, O., and Willmott, S. (2005). *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing)*. University of Southampton.

[89] Luckham, D. C. (2001). *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[90] Mafik, V. and Pechoucek, M. (2004). *Social knowledge in multi-agent systems*, volume 2086/2001 of *Lecture Notes in Computer Science*, pages 211–245. Springer Berlin / Heidelberg.

[91] Malouf, R. M. (1995). Towards an analysis of multi-party discourse. Talk.

[92] Mathieu, P., Routier, J. C., and Secq, Y. (2002a). Dynamic organization of multi-agent systems. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS02)*, pages 451–452.

[93] Mathieu, P., Routier, J.-C., and Secq, Y. (2002b). Principles for dynamic multi-agent organizations. In *Proceedings of the 5th Pacific Rim International Workshop on Multi Agents: Intelligent Agents and Multi-Agent Systems (PRIMA02)*, pages 109–122.

[94] Michelson, B. (2006). Event-driven architecture overview. *Patricia Seybold Group*.

[95] Ndumu, D., Nwana, H., Lee, L., and Collis, J. (1999). Visualising and debugging distributed multi-agent systems. In *AGENTS '99: Proceedings of the*

third annual conference on Autonomous Agents, pages 326–333, New York, NY, USA. ACM.

[96] Oikarinen, J. and Reed, D. (1993). RFC 1459: Internet relay chat protocol.

[97] Omicini, A., Ricci, A., and Viroli, M. (2008). Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, **17**(3), 432–456.

[98] Padgham, L., Winikoff, M., and Poutakidis, D. (2005). Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, **18**(2), 173–190.

[99] Petters, S. M. (2007). Execution-time profiles. Technical report, NICTA, Sydney, Australia.

[100] Platon, E., Sabouret, N., and Honiden, S. (2006). Overhearing and direct interactions: Point of view of an active environment. In *Environments for Multi-Agent Systems II*, pages 121–138. Springer.

[101] Pokahr, A. and Braubach, L. (2007). *Jadex Tool Guide - Release 0.96*.

[102] Potiron, K., El Fallah Seghrouchni, A., and Taillibert, P. (2013). *From Fault Classification to Fault Tolerance for Multi-Agent Systems*. SpringerBriefs in Computer Science. Springer.

[103] Ramos-Thuel, S. and Lehoczky, J. P. (1993). On-line scheduling of hard deadline aperiodic tasks in fixed priority systems. In *14th IEEE Real-Time Systems Symposium*, pages 160–171.

[104] Ramos-Thuel, S. and Lehoczky, J. P. (1994). Algorithms for scheduling hard aperiodic tasks in fixed priority systems using slack stealing. In *15th IEEE Real-Time Systems Symposium*, pages 22–35.

[105] Rebollo, M., Giret, A., Argente, E., Carrascosa, C., Corchado, J. M., Fernandez, A., and Julián, V. (2009). 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.

[106] Ringold, P., Alegria, J., Czaplewski, R., Mulder, B., Tolle, T., and Burnett, K. (1996). Adaptive Monitoring Design for Ecosystem Management. *Ecological Applications*, **6**(3), 745–747.

[107] Rossi, S., Rossi, S., and Busetta, P. (2004). Towards monitoring of group interactions and social roles via overhearing. *Proceedings of CIA-04*, **3191**, 47–61.

[108] Routier, J., Mathieu, P., and Secq, Y. (2001). Dynamic Skill Learning: A Support to Agent Evolution. pages 25–32.

[109] Sánchez-Anguix, V., Espinosa, A., Hernández, L., and García-Fornes, A. (2009). Mamsy: A management tool for multi-agent systems. *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, pages 130–139.

[110] Saunier, J., Balbo, F., and Badeig, F. (2006). Environment as active support of interaction. In D. Weyns, H. V. D. Parunak, and F. Michel, editors, *E4MAS*, volume 4389 of *Lecture Notes in Computer Science*, pages 87–105. Springer.

[111] Serrano, E., Gómez-Sanz, J. J., Botía, J. A., and Pavón, J. (2009). Intelligent data analysis applied to debug complex software systems. *Neurocomputing*, **72**, 2785 – 2795.

[112] Sprunt, B., Lehoczky, J., and Sha, L. (1988). Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings of the Real-Time Systems Symposium*, pages 251–258.

[113] Stanovich, M., Baker, T., Wang, A.-I., and Harbour, M. (2010). Defects of the posix sporadic server and how to correct them. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 35–45.

[114] Strosnider, J. K., Lehoczky, J. P., and Sha, L. (1995). The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, **1**(44), 73–91.

[115] Such, J. M., GarcíA-Fornes, A., Espinosa, A., and Bellver, J. (2013). Magentix2: A privacy-enhancing agent platform. *Eng. Appl. Artif. Intell.*, **26**(1), 96–109.

[116] Terrasa, A. and Bernat, G. (2004). Extracting temporal properties from real-time systems by automatic tracing analysis. In J. Chen and S. Hong, editors, *Real-Time and Embedded Computing Systems and Applications*, volume 2968 of *Lecture Notes in Computer Science*, pages 466–485. Springer Berlin Heidelberg.

[117] Terrasa, A., Espinosa, A., and García-Fornes, A. (2008). Lightweight POSIX Tracing. *Software Practice and Experience*, **38**(5), 447–469.

[118] Thane, H. (2000). Monitoring, testing, and debugging of distributed real-time systems. PhD Thesis.

[119] Tia, T.-S., Liu, J. W.-S., and Shankar, M. (1996). Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. *Real-Time Syst.*, **10**(1), 23–43.

[120] Tichý, P. and Slechta, P. (2006). *Java Sniffer 2.7 User Manual*.

[121] Vázquez-Salceda, J., Dignum, V., and Dignum, F. (2005). Organizing Multiagent Systems. *Autonomous Agents and Multi-Agent Systems*, **11**, 307–360.

[122] Villarrubia, G., Sánchez, A., Barri, I., Soler, E. R., del Viso, A. F., Sánchez, C. R., Cabo, J. A., Álamos, T., Sanz, J., Seco, J., Zato, C., Bajo, J., Rodríguez, S., and Corchado, J. M. (2012). Proximity detection prototype adapted to a work environment. In P. Novais, K. Hallenborg, D. I. Tapia, and J. M. C. Rodríguez, editors, *ISAmI*, volume 153 of *Advances in Intelligent and Soft Computing*, pages 51–58. Springer.

[123] Wang, Z. G. and Liang, X. H. (2006). *A Graph Based Simulation of Reorganization in Multi-agent Systems*.

[124] Weyns, D., Omicini, A., and Odell, J. (2007). Environment as a first class abstraction in multiagent systems. *Autonomous Agents and Multi-Agent Systems*, pages 5–30.

[125] Weyns, D., Haesevoets, R., Helleboogh, A., Holvoet, T., and Joosen, W. (2010). The MACODO middleware for context-driven dynamic agent organizations. *ACM Transaction on Autonomous and Adaptive Systems*, **5**, 3:1–3:28.

[126] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., and Stenström, P. (2008). The worst-case execution-time problem – overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, **7**, 36:1–36:53.

[127] Yodaiken, V. (1999). The RTLinux manifesto. In *Proc. of The 5th Linux Expo, Raleigh, NC*.

[128] Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **12**(3), 317–370.

[129] Zato, C., Villarrubia, G., Sánchez, A., Barri, I., Rubión, E., Fernández, A., Rebate, C., Cabo, J., Álamos, T., Sanz, J., Seco, J., Bajo, J., and Corchado, J. (2012). Pangea – platform for automatic construction of organizations of intelligent agents. In S. Omatu, J. F. De Paz Santana, S. R. González, J. M. Molina, A. M. Bernardos, and J. M. C. Rodríguez, editors, *Distributed Computing and Artificial Intelligence*, volume 151 of *Advances in Intelligent and Soft Computing*, pages 229–239. Springer Berlin Heidelberg.