Département d'Informatique, groupe Télécom
Université de Fribourg (Suisse)

Towards Active Network Management with *Ecomobile*, an Ecosystem-inspired Mobile Agent Middleware

Design, Implementation, Simulation and Application to Optical Networks

THESE

présentée à la Faculté des Sciences de l'Université de Fribourg (Suisse)
pour l'obtention du grade de *Doctor scientiarum informaticarum*

Daniel Rossier-Ramuz
Ing. info. dipl. EPFL
de
Montagny-les-Monts (FR)

Thèse No. 1392

2002

Imprimerie Uni-Print, Université de Fribourg

Accepté par la Faculté des Sciences de l'Université de Fribourg (Suisse) sur la proposition de

Prof. Marino Widmer	Université de Fribourg, Suisse	Président du Jury
Prof. Beat Hirsbrunner	Université de Fribourg, Suisse	Directeur de thèse
Dr. Rudolf Scheurer	Université de Fribourg, Suisse	Co-directeur
Prof. Samuel Pierre	Ecole Polytechnique de Montréal, Canada	Premier rapporteur
Dr. Daniel Rodellar	Swisscom Innovations AG, Berne, Suisse	Second rapporteur

Fribourg, le 21 Octobre 2002

Le Directeur de thèse: Le Doyen:

Prof. Beat Hirsbrunner Prof. Dionys Baeriswyl

A Catherine, Aline, Emilie, …

A mon Parrain

Résumé

Les futurs réseaux optiques multicouches utilisant le multiplexage en longueur d'onde permettront dans un proche avenir l'introduction de nouveaux services optiques à valeur ajoutée. Le réseau de transport passif conventionnel se transformera ainsi en une couche de transport active et intelligente. Le succès de ces nouveaux services dépendra toutefois de l'efficacité, de la résilience et de la programmabilité de l'infrastructure de gestion mise à la disposition des opérateurs de réseaux et des fournisseurs de service. Une approche de gestion traditionnelle et purement centralisée ne permet pas d'intégrer les contraintes d'hétérogénéité et de dynamicité inhérentes à ces nouveaux réseaux et doit par conséquent évoluer en direction d'une gestion décentralisée et auto-adaptative.

L'introduction d'agents logiciels autonomes est une technique prometteuse, flexible et particulièrement bien adaptée à ce type de système distribué. Des agents réactifs capables de se déplacer dans le réseau permettent d'utiliser des approches bio-inspirées comme celles issues du *comportement émergent*.

Dans le cadre de ce travail, nous avons développé un intergiciel appelé *Ecomobile*, composé d'agents mobiles qui appliquent des principes observés dans des écosystèmes naturels, afin de disséminer et d'activer des tâches coopératives de gestion de réseau. Dans cette perspective, nous avons analysé le comportement d'une communauté d'agents mobiles en examinant l'évolution de leur population, leur propagation, la fréquence de visite des nœuds et des liens, ainsi que différentes stratégies de dissémination de tâches intelligentes dans diverses topologies de réseaux de transport.

Après avoir étudié différents systèmes d'agents mobiles intégrant des modèles de navigation déterministes et stochastiques, nous avons élaboré un modèle original d'agent réactif en séparant les modèles de coordination et de navigation, qui constituent le *schéma de comportement mobile*, du modèle computationel, qui correspond aux *tâches opérationnelles*. L'implémentation d'un écosystème artificiel constitué de ces agents mobiles a été réalisée à l'aide d'un formalisme de programmation réactive.

Les tâches opérationnelles génériques proposées dans notre recherche permettent la composition de tâches de gestion complexes. La réponse de notre écosystème à l'insertion dynamique de ces tâches génériques a été simulée et analysée. Afin de déployer *Ecomobile* dans un réseau actif, nous avons utilisé une plateforme d'agent compatible FIPA appelée *Jade*. Le développement d'une agence particulière sous forme d'un agent *Jade* offre aux agents mobiles un environnement d'exécution adéquat et leur fournit les services de migration nécessaires.

En guise de conclusion, nous avons établi une translation de la sémantique d'*Ecomobile* dans un environnement défini par un nœud de réseau comprenant différentes fonctions optiques de conversion et de routage de longueur d'onde. La définition et l'implémentation d'un service de protection différenciée à l'aide d'une tâche opérationnelle devrait enfin ouvrir la voie à des améliorations et des innovations dans le domaine de la gestion des réseaux de transport optiques multicouches grâce à *Ecomobile*.

Mots-clé: gestion de réseau distribuée, agents mobiles, écosystème artificiel, systèmes réactifs, réseaux optiques

Abstract

The future multi-layer optical networks based on wavelength division multiplexing technology will lead to the creation of new value-added *optical* services, which is bound to transform the traditional passive transport network into an *active* and *intelligent* transport layer. The successful deployment of these complex networking services and the possibility of subjecting them to a dynamic control, however, strongly depend on the management infrastructure, on its resilience and its ability to react to network changes. In this context, traditional platform-centred management systems have lost their attractiveness: a distributed decentralised and self-adaptive network management should constitute an ideal approach, in order to deal with the complexity of a heterogeneous, scalable and continuously evolving network environment.

According to this perspective, the recourse to autonomous software agents can now be considered as one of the most promising and flexible of the distributed processing techniques. The development of *reactive agents* acting as mobility-oriented individuals benefits from bio-inspired approaches such as emergent behaviour. The powerful dynamic and active mechanisms characterizing the evolution of mobile reactive agents enhance the network infrastructure and lead to a self-organizing knowledge-based network environment.

In order to address the numerous challenging issues related to the management of future multi-layer transport networks, we propose to develop an ecosystem-inspired mobile agent middleware called *Ecomobile*, intended for the dissemination and the activation of cooperative management tasks. Particular emphasis will be laid on the transport network management and several topics related to the population of mobile agents will be discussed, such as their propagation within the network infrastructure, the frequency of node and link visits or the dissemination of *intelligent* tasks.

After having identified the main characteristics of conventional mobile multi-agent systems based upon deterministic and stochastic migration strategies, which are particularly relevant to the field of distributed control, we have decided to adopt a novel agent architecture based upon a clear separation between the *mobile behaviour scheme* composed of the navigation and coordination model, on the one hand, and the *task objectives*, which refer to the agent's operational behaviour or to the computational model, on the other hand. The implementation of the ecosystem is realized with a reactive programming formalism.

The generic task objective models we have elaborated correspond to basic networking functions and are intended for the compositional building of more sophisticated tasks. The response of the ecosystem to the dynamic insertion of task objectives will be analyzed by means of a simulation. The deployment of our middleware into active nodes will be achieved with the FIPA-compliant *Jade* agent platform, in which the *Ecomobile* agency provides the mobile agents with the necessary requirements for their migration and for the execution of the task objectives.

We will finally show that the *Ecomobile* middleware can be applied in the field of optical transport network management by means of a pertinent mapping of the *Ecomobile* semantics onto the optical network environment. The definition and the implementation of a new value-added differentiated protection service into a multi-layer optical network should eventually pave the way for further improvements and innovations in the field of transport network management.

Keywords: distributed network management, mobile agents, artificial ecosystem, reactive systems, optical network

Acknowledgements

The achievement of this thesis has been possible thanks to the involvement of several key persons.
I would like to express my deep gratitude to:

- Prof. Beat Hirsbrunner, for accepting to trust me and to give me the opportunity to perform this research work under his supervision and in a stimulating academic environment;

- Dr. Rudolf Scheurer, for supervising my work, for supporting me and for the encouraging discussions which allowed me to overcome the difficulties encountered during my scientific investigations;

- Dr. Hermann Gysel (*Swisscom Fixnet*), for giving me the opportunity to start this thesis in a continuously evolving industrial environment;

- Mr. Dipl.-Ing. Andreas Dürsteler (*Swisscom Innovations*), for trusting me and for giving me access to Swisscom's solid infrastructure, for his patience and his kindness, and for allowing me to reach contractual agreements with Swisscom Innovations;

- Olivier, Sergio, Amine and the colleagues of the *Department of Informatics*, for all the exciting and constructive discussions on the topic of mobile agents and network management,
… and for accepting me in their football team;

- and, last but not least, my wife Catherine, for her precious contribution towards transforming my manuscript into a readable "English-styled" document and, before all, for her patience and her love, which allowed me to overcome my doubts and to quiet down during many sleepless nights.

TABLE OF CONTENTS

List of Figures

List of Tables

Abbreviations

AC	Agent Context
ACC	Agent Communication Channel
ACL	Agent Communication Language
AMS	Agent Management System
AN	Active Networks
ASON	Automatic Switched Optical Network
DCN	Data Communication Network
BDI	Belief-Desire-Intention
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
DF	Directory Facilitator
DPS	Dynamic Protection Set-up
ECC	Embedded Communication Channel
FIPA	Foundation for Intelligent Physical Agents
GMPLS	Generalized Multi-Protocol Label Switching
GNMT	Generic Network Management Tool
IETF	Internet Engineering Task Force
ITU	International Telecommunication Union
JIDM	Joint Inter-Domain Management
JVM	Java Virtual Machine
IDL	Interface Description Language
M-Agent	Mobile Agent
MAS	Multi-Agent System
MASIF	Mobile Agent Services Interoperability Facility
MBS	Mobile Behaviour Scheme
MbD	Management by Delegation
MIB	Management Information Base
MMS	Mobility Management System
MO	Managed Object
MPLS	Multiprotocol Label Switching
MTP	Message Transport Protocol
MTS	Message Transport System
NMS	Network Management System
OMG	Object Management Group
OPTIMA	OPTical network management with Intelligent and Mobile Agent
OSI	Open Systems Interconnect
OSS	Operational Support Systems
OTN	Optical Transport Network
QoP	Quality of Protection
QoS	Quality of Service
RCL	Relative Capacity Loss
RDP	Remote Delegation Protocol
RMI	Remote Method Invocation
RWA	Routing and Wavelength Assignment
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
TDM	Time Division Multiplexing
TMN	Telecommunication Management Network
TO	Task Objective
UPI	Universal Place Identifier
VPN	Virtual Private Network
WDM	Wavelength Division Multiplexing
XML	eXtended Mark-up Language

Introduction

Over the last decade, the extraordinary development of the *World Wide Web* has led to the creation of new multi-media services and enterprise applications which are continuously evolving to support more and more facilities. These new services require far more complex network characteristics than simply enough bandwidth: flexibility, reliability, scalability, fast reaction to network changes, and balanced trade-off between quality and cost by means of customized *Quality-of-Service* (QoS) constitute the essential properties of future telecommunication networks.

The deregulation of the telecommunication market implies the definition of new business models leading service providers, which sell customers network services, and network operators, which supply service providers with a network infrastructure, to evolve as distinct parties in a highly competitive environment. The service providers, on the one hand, wish to create and modify services in a dynamic way, to establish flexible *Service Level Agreements* (SLA) with variable QoS parameters, and to monitor real-time service status and performance; the network operators, on the other hand, must be able to plan, create and provision new services, to manage multiple, customer-tailored SLAs, to track and report SLA QoS performance, to establish and manage network policies, and finally to provide for secure network access [BTS+01].

The competitiveness of network operators consequently relies on their ability to create value-added services within the existing as well as the future infrastructure and to deal with new customer requirements. The migration from traditionally passive transport networks to adaptive and intelligent components, and the ensuing *intelligent* transport network, directly result from these transformations. In this context, *Wavelength Division Multiplexing* (WDM) based optical networks and *Fiber to the Home* (FTTH) technology associated to the creation of advanced interactive services, such as digital television services, will probably constitute one of the major revolutions in the telecommunication landscape during the next decades.

The optical transport network is expected to transport data streams at a bit rate exceeding 1 Tbit/s on a single link in the near future and, according to predictions, at a bit rate of 10 Tbit/s by 2010 [LDA+98]. From the point of view of the network operator, optical networks supporting networking functions in the optical domain appear to be the most attractive technology for the creation of new customer-oriented services.

Performance of these complex networking services however strongly depends on the management infrastructure, on its resilience and on its capability to react to network changes. In traditional network management, for example, management information is predefined and standardized, which makes future updating difficult and troublesome. In the future, traditional platform-centred network and service management systems will not meet the expectations of future multi-layer transport networks any more. Inherently distributed management systems will have to match new paradigms implying that a substantial portion of network control and network knowledge is no longer centralized in management systems. In this perspective, decentralised and self-adaptive network management should constitute the ideal approach meeting expectations of network operators and coping with a heterogeneous and continuously evolving network environment.

The software agent paradigm, which is currently gaining increasing attention in the scientific community, arises from the convergence of several disciplines like distributed artificial intelligence, biology, software engineering and telecommunications, and entails powerful mechanisms for the development, by means of intelligent and mobile agents, of complex distributed systems, such as network management applications. An intelligent agent may be defined as an encapsulated computing system situated in the network environment and able to exhibit flexible and autonomous behaviour in order to fulfil its design objectives [J01]. Intelligent agents are characterized by their *social* behaviour and interactions; they exchange knowledge, goals, skills, and plans in order to make jointly short and long-term decisions to solve complex problems. An agent may also have the ability to move that is, to move its code and data from one location to another in order to accomplish its task progressively, in which case it places control and management software processes dynamically at the most appropriate locations within the telecommunication environment [MRK96]. The transfer of a large amount of data between a manager and remote entities can thus be avoided; mobile agents can also continue their work when the connection is temporarily interrupted. Since they enable both temporal and spatial distribution of management activities, mobile agents are particularly well suited to the intelligent adaptation of services, as well as to advanced service interworking and integration. In the context of network management, the mobile agent approach entails a further advantage: while network management protocols must continuously be developed in order to resolve new emerging problems, the resort to mobile agents makes network management protocols between the managing station and the managed device obsolete [ZZ98].

Although mobile agents are perfectly adequate for network management systems, this novel approach raises important open issues mainly concerning the proliferation of agents within the network environment and the control of their density, which may have an influence on their overall performance, as well as on their internal architecture, which often radically differs from one task implementation to the other according to different mobility approaches.

The mobility paradigm and its application to transport network management systems constitute the main subjects of this dissertation, in which we are trying to show that agent mobility associated to network infrastructure may contribute to the elaboration of an intelligent transport network and may thus facilitate the design and deployment of network management tasks within such networks. Our basic approach consists in considering mobile agents and the network environment as an artificial ecosystem, in which the network infrastructure can be considered as its *biotope*, i.e. the environment made up of resources, and the mobile agents can be referred to its *biocenose*, or the individuals' society. This approach allows us to benefit from self-organisation properties which emerge from any natural ecosystem, in order to guarantee the stability and the efficient investigation of the infrastructure for the execution of network management tasks, in particular.

The deployment of mobile agent systems into large-scale network devices raises a number of open issues which will be addressed in our work. In this perspective, manufacturers such as 3Com, Cisco or Nortel Networks have announced the development of network devices endowed with increasing computing resources and liable to support an embedded *Java Virtual Machine* (JVM). In this context, *Active Networks* (AN) technology provides an interesting architectural framework to transform usual passive network nodes into programmable nodes in which code can be dynamically installed and automatically configured. The concept of active network management as it is developed in this thesis refers to a fully decentralised management; this kind of management leads network components to be *active*, like active nodes in AN; more generally, active components must be capable of dynamically hosting software entities and enabling their activation in an appropriate and secure execution environment. In the case of legacy systems, the active node may be composed of a local proxy interacting with legacy components.

MOTIVATIONS

This thesis has been realized at the *University of Fribourg* in an industrial collaborative context involving *Swisscom Innovations,* the research and development unit of *Swisscom Ltd*, in Bern, Switzerland, which is the major Swiss telecommunication operator. The initial motivation of this thesis stems from the necessity to explore new solutions towards the efficient management of the future WDM-based optical transport network. Unlike other network technologies, WDM optical networks deal with large numbers of heterogeneous network components presenting different capabilities and limitations such as optoelectronic conversion, partial matrix switching, wavelength conversion, etc. This diversity makes the implementation of traditional networking algorithms into conventional network management systems particularly complex. The optical layer constitutes the lowest network layer transporting several Tbit/s of client-independent data on a single fibre, which involves serious survivability issues, so that the management system is obviously a vital component of WDM optical networks.

As we have seen, the approach adopted in the context of this work focuses on the software agent technology and, in particular, on mobile agents. Previous investigations from the research community in this domain have led us to believe that mobile agents navigating within the network infrastructure in accordance with physical constraints and impairments induce powerful reactive mechanisms and provide a *natural* software engineering approach to the development of conventional network management tasks, as well as of more sophisticated algorithms devoted to the resource control of optical networks.

The emerging mobile agent technology raises a number of challenging issues, mainly concerning architecture, self-adaptability and pragmatic implementation. This is the reason why we have decided to concentrate our efforts on the development of a mobile agent based software infrastructure specifically devoted to transport network management applications. We propose to place particular emphasis on the development of a mobile multi-agent system able to support a combination of various mobility-oriented approaches which implement both the deterministic migration strategy used by delegation mobile agents and the stochastic migration strategy used by mobile agents exhibiting emergent behaviour.

This document is an account of the conception and elaboration of the mobile agent middleware which we have called *Ecomobile*. Since *Ecomobile* is intended to be implemented within large-scale transport networks, particular attention must be drawn to the ongoing efforts in the field of standardization. In this perspective, strong emphasis will be placed on existing and evolving agent standards, such as the *Foundation of Intelligent Physical Agents* (FIPA); neither should we neglect to deal with legacy network management systems built on top of existing standardized frameworks, such as TMN or SNMP.

Rather than a deep and pure theoretical study devoted to a specific subject, this thesis reflects exploratory work and must be seen as an attempt to identify and to develop issues related to mobile agent based network management systems: we have chosen to develop the issues which seemed most relevant to our target application from a network operator perspective. In this context, the development of *Ecomobile*, which constitutes the central part of this thesis, should provide a powerful self-adaptive framework for the implementation of bio-inspired algorithms intended for the resource control of future optical networks. The development of such algorithms is an important objective of the OPTIMA[1] project for which this thesis constitutes a preparatory work.

[1] *OPTical network management with Intelligent and Mobile Agents* (see Section 7.1)

GUIDELINES

This document is divided into three main sections: the first part, which is devoted to the technology insight, is composed of Chapters 1 and 2. The second part, which contains Chapters 3, 4 and 5, describes the design, implementation and simulation of *Ecomobile*, while the final part of this document is devoted to optical network management, our use case study presented in Chapters 6 and 7.

Chapter 1 contains a description of conventional approaches considered in today's network management systems. Our summary of the most significant mobile agent based approaches is followed by a survey of several popular agent platforms.

Chapter 2 presents a decomposition of mobile multi-agent systems into three abstraction models: the computational, the coordination and the navigation models. This decomposition implies a refinement of mobile agent systems modelling with respect to their main characteristics and leads to the particular architectural model of *Ecomobile*, in which the coordination and navigation models forming the mobile behaviour scheme are separated from the computational model forming the task objective. We will then proceed to describe a particular bio-inspired approach which points out the interactions of our mobile agents with their local environment

The main components of our middleware are introduced in Chapter 3. We shall first introduce the *Ecomobile* model and its main components in order to examine the conceptual framework, which is based on *mobile behaviour schemes* on the one hand, and on the *task objectives* on the other hand. We shall then describe the ecosystem principles that have been retained for the control of the mobile agent population. After the computational model consisting in the task objective has been thoroughly examined, some generic task objective models corresponding to basic network management functions, and which can be used for the compositional building of more sophisticated tasks, will be presented.

Chapter 4 is devoted to the realization of the *Ecomobile* concepts by means of reactive programming. Different issues related to the implementation of reactive behaviours into reactive instructions will be pointed out and we shall finally propose a viable deployment of *Ecomobile* within a FIPA-compliant environment by means of the *Jade* agent platform.

Chapter 5 begins with a short introduction to the *Generic Network Management Tool* (GNMT), which has been developed in the context of this thesis. GNMT provides a functional simulation framework for the study of agent-based solutions intended for multi-layer optical network management. Simulation results issued from various experiments performed with GNMT on different network configurations will then be discussed and a behavioural analysis of the ecosystem and of its response to the dynamic insertion of specific task objectives will also be examined.

The basis elements of optical networks are presented in Chapter 6, which introduces the optical network components and networking functions, as well as current approaches towards the management of the *Optical Transport Network* (OTN). Optical networks constitute our main application domain.

Chapter 7 provides an overview of emerging *optical* services and shows how our infrastructure can be implemented into optical components by means of a pertinent mapping of the *Ecomobile* semantics. We then present the implementation and the simulation results for a task objective dedicated to a new value-added differentiated protection service.

In our conclusions, we finally point out open issues and possible extensions to *Ecomobile* for future research activities.

Part I

Technology Insight

Chapter 1
Mobile Agents and Network Management

In this work we present a reactive mobile agent based middleware called *Ecomobile* which is partially inspired from principles issued from natural ecosystems. *Ecomobile* is tailored to a decentralized network management with particular emphasis on future transport networks, such as optical networks. The different issues which will be addressed in this thesis are concerned with the realization of a self-organized population of mobile entities, the composition of network-oriented tasks with an efficient dispersal of these tasks within the network infrastructure and the deployment of *Ecomobile* into active and heterogeneous network components, such as optical nodes with different capabilities.

The conceptual approach that we have adopted to design *Ecomobile* consists in taking into account different approaches using mobile agent technology for network management which turned out to be particularly relevant, from our point of view, to the development of an intelligent transport network. In order to place *Ecomobile* in its context, according to past and present research, we wish to develop along this chapter a number of basic concepts around mobile agents and network management.

Network management aims at deploying, integrating, and coordinating all the resources necessary in order to configure, monitor, test, analyze, evaluate, and control the communication network, so that service-level objectives are met at a reasonable cost [BBB$^+$99].

The design and the implementation of new network management systems inevitably rely on existing object-oriented and distributed technologies with *open interfaces*. Interoperability between network entities involved in management processes is ensured by the adoption of international standards[1]. The *Object Management Group* (OMG)[2] and the *International Telecommunication Union* (ITU)[3] - jointly with the *International Organization for Standardization* (ISO)[4] - provide object-oriented approaches and architectural frameworks for open distributed systems, and therefore play a central role in the development of network components and management systems. However, the implementation choice advocated in the context of most standards is left open to the manufacturers and leads to a considerable amount of proprietary software components obviously raising a large number of interoperability issues.

In this chapter, we propose to examine how network management systems are traditionally implemented. We shall also have a look at the most famous mobile agent based approaches which enable to address the management of transport networks in a perspective of interoperability with existing network technologies and the migration towards a decentralized approach for a highly dynamic, scalable and heterogeneous network infrastructure[5].

[1] We do not make any difference between standards and recommendations, which are considered as equivalent.
[2] http://www.omg.org
[3] http://www.itu.int
[4] http://www.iso.org
[5] In this chapter, we do not yet deal with optical network management, which constitutes the main topic of Chapters 6 and 7.

1.1 NETWORK MANAGEMENT SYSTEMS

1.1.1 General Components

Network Management Systems (NMS) - also called *Operational Support Systems* (OSS) - provide the network operators with the operational functions necessary to control the network resources on the one hand and with the service management on the other hand. Network components are interconnected according to specific *network topologies* and are subject to various configuration changes. In future networks, an efficient NMS will have to be increasingly distributed, flexible, scalable, and able to support inter-networking with heterogeneous systems [E712_99].

Since software entities may be located everywhere in the network, from the manager console to the network devices supplied by the manufacturers, a network management system is inherently distributed over the network. The fact that most network management applications require a distributed infrastructure does *not* mean, however, that the management logic, the *intelligence* itself, is distributed. In this context, the approach is based upon a centralized management.

Network management systems have been influenced by the *Open System Interconnection* (OSI) layered model[1], for several years, which has led to the definition of two popular and word-wide spread frameworks: the *Telecommunication Management Network* (TMN) from ITU-T [M3100_96] – used for the management of the transport network - and the *Simple Network Management Protocol* (SNMP) from the *Internet Engineering Task Force*[2] (IETF) [C$^+$93] – used for the management of Internet networks.

MANAGER-AGENT PARADIGM

TMN and SNMP rely on a *client-server* (C/S) communication model between a *manager* (OSI manager) and an agent (OSI agent). The manager is responsible for maintaining the global view of the entire network and for providing the *operator* with the control functions; the manager is located in the management applications and communicates with the manageable resources. Each manageable resource is subordinated to an OSI agent which is responsible for the access to the locally available attributes and functions for management purposes. In this context, the resource may refer to hardware (network card, physical port, etc.) or abstract components, such as end-to-end connections or switching matrixes.

The manageable resources in the network are called *managed objects* and are described by a collection of attributes and functions. Managed objects are manipulated by the OSI manager via the OSI agents through a standardized protocol (CMIP/SNMP) using a standardized notation, respectively (GDMO/SMI). The managed objects can be defined in an object-oriented way so that the managed object model gives a logical representation of the manageable entity. The managed objects are stored in a *Management Information Base* (MIB) which is generally located at the same place as the agents and the managers. The general communication architecture between manager and agent is depicted on Figure 1-1.

[1] OSI reference model ISO/IEC 7498 available at http://www.iso.org
[2] http://www.ietf.org

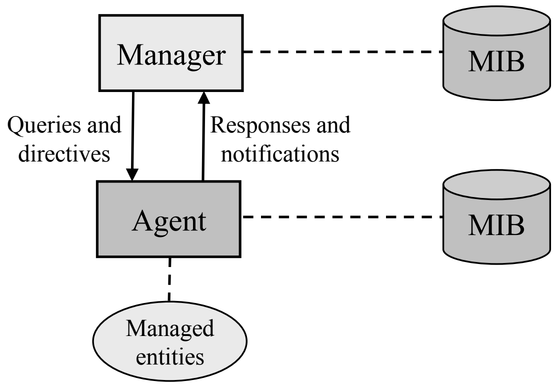


Figure 1-1. Interactions between manager, agent and MIB

According to the manager-agent communication model and the C/S paradigm, the agent acts as the server while the manager acts as the client, so that additional entities can easily have access to the MIB - i.e. to the device configuration - by declaring themselves clients to the corresponding agent. In a mobile agent system perspective, the MIB is part of the environment of mobile entities and can be accessed by querying the OSI agent in charge of the MIB through a C/S communication model.

The manager-agent model is a platform-centred approach relying on a pure client-server paradigm; it entails drawbacks in scalability, reliability, efficiency and flexibility, and is therefore unsuitable for large and heterogeneous networks [BPW98]. Consequently, the deployment of new value-added services in future transport networks, for example, reveals to be difficult in the context of this model.

Finally, management based on the manager-agent model is focused on monitoring network infrastructures rather than on managing the applications delivered by the network. On the other hand, the popularity and the world-wide spread of such NMS force the future NMS generations to deal with legacy systems and to consider them as part of the *environment* in which they have to interact. The NMS must therefore become increasingly open and flexible to handle numerous types of interfaces.

The successful deployment of a new NMS into current telecommunication networks will strongly depend on its ability to deal with the management interfaces of existing components and technologies. A proper access to the interfaces will enable the NMS to reach the MIBs or any location-dependent information.

Nowadays, distributed NMS are mainly developed on top of two pre-dominant distributed technologies: CORBA on the one hand and *Java* RMI on the other hand.

CORBA

The *Common Object Request Broker Architecture* (CORBA) from the *Object Management Group* (OMG)[1] provides an open, scalable and flexible distributed system framework which is promoted by ITU-T and by the *TeleManagement* forum[2] [NMF98].

[1] http://www.omg.org

[2] The *TeleManagement* Forum (TM Forum) is a non-profit global organization that provides leadership, strategic guidance and practical solutions to improve the management and operation of communications services. Further information available at http://www.tmforum.org

The central component of CORBA is the *Object Request Broker* (ORB). ORB encompasses the entire communication infrastructure necessary to identify and locate objects, handle connection management and deliver data. In general, the ORB is not required to be a single component; it is simply defined by its interfaces. The ORB Core is the most crucial part of the Object Request Broker; it is responsible for the communication of requests issued from the client to the server and vice-versa. The interface of serving objects is described in a platform-neutral language called *Interface Description Language* (IDL). Attributes and method signatures belonging to remote objects are expressed in IDL.

The basic functionality provided by the ORB consists in passing the requests from the clients to the object implementations on which they are invoked. In order to make a request the client can communicate with the ORB core either through the IDL *stub* or through the *Dynamic Invocation Interface* (DII). The *stub* provides the mapping between the client's implementation language and the ORB core. As long as the implementation of the ORB supports this mapping, the client can be written in any language. The ORB core then transfers the request to the object implementation which receives the request as an invocation through either an IDL skeleton, or a dynamic skeleton.

Underneath the ORB, the *Internet Inter-ORB Protocol* (IIOP) enables the client/server to exchange information using IP networks.

JAVA RMI

SUN Microsystems propose the *Remote Invocation Method* (RMI)[1] for *Java* based distributed applications. The server and the client are fully programmed in *Java* and usually reside on different *Java virtual machines*. RMI provides a naming service and relies on the *Java* native security mechanisms.

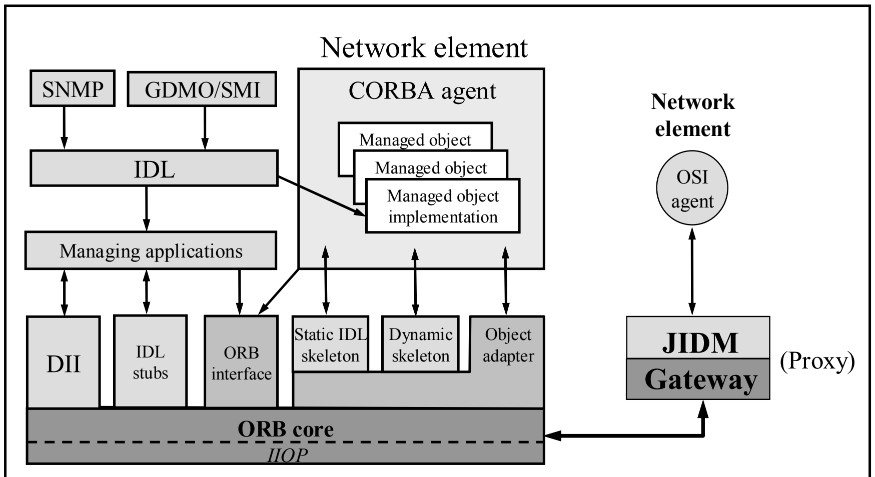
The interoperability between RMI and CORBA objects can be addressed through IIOP. The combination of RMI and IIOP (RMI-IIOP) enables a *Java* client to deal with a CORBA server for example. In a more general case, RMI-IIOP allows programmers to develop CORBA applications for the Java platform without using IDL to describe remote interfaces, and to directly take advantage of RMI features such as *passing object by value* between application components; the CORBA programming model only supports *passing object by reference*.

CORBA AND TMN/SNMP INTERWORKING

Joint Inter-Domain Management (JIDM)[2] is a technology that defines how network management components based on TMN and SNMP can interoperate with CORBA based components. The interoperability first requires a definition of model equivalencies between the two domains. This definition is given in the S*pecification Translation* document [JIDM97], which explains how information models can be translated from one representation to another (ASN.1/GDMO to/from IDL). A second document [JIDM98], called *Interaction* Translation, defines how to perform OSI-like (and SNMP like) services in CORBA. An overall architecture of a CORBA-based management system is depicted on Figure 1-2.

JIDM actually provides a way to develop *Java* objects able to deal with the OSI agents. The implementation of mobile agent systems, mostly developed in the *Java* language in a legacy TMN/SNMP-based network component, can be achieved through the use of a proxy and a JIDM gateway so that the communication between the mobile agent and the OSI agent becomes possible.

[1] http://java.sun.com/rmi
[2] http://www.jidm.org



SMI	Structure of Management Information	IDL	Interface Description Language
CMIP	Common Management Information Protocol	IIOP	Internet Inter-ORB Protocol
CORBA	Common Object Request Broker Architecture	ORB	Object Request Broker
DII	Dynamic Invocation Interface	OSI	Open Systems Interconnection
GDMO	Guidelines for the Definition of Managed Objects	SNMP	Simple Network Management Protocol

Figure 1-2. CORBA-based approach towards network management integrating TMN/SNMP agents

A fully CORBA-based NMS may be possible if all network elements implement CORBA agents; generally, standardized distributed object technologies allow manufacturers to develop low-cost platforms and tools. However, the large investments of network operators in traditional systems based on CMIP/SNMP will compromise such an approach. Two scenarios therefore have to be considered: either the network element supports CORBA interfaces and provides defined IDL-based information models, or the network element does not support CORBA; in this case, a JIDM gateway on the manager side can perform necessary translations, so that managing object-oriented applications can seamlessly access managed objects in a common way. According to this approach, we can envisage to implement mobile agents which are able to deal with - and to control - legacy managed objects.

As we will explain in Section 1.2, most agent platforms are implemented on top of a CORBA or RMI system.

1.1.2 Centralised versus Decentralised Network Management

Network management systems are usually composed of a manager implementing most of the service logic. The information concerning network resources is retrieved from the network by querying the agents residing in the network devices. When an alarm occurs, the manager is informed by the agent and the whole decision making is performed at the manager level; it is a centralised system.

A centralised system approach based on a manager-centric approach gives a complete picture of the network. Using that information, the system can easily implement algorithms leading to global optima in case of allocation of network resources or load-balancing for example. Such an approach generally requires message-passing mechanisms imposing a large overhead. It is also more vulnerable to system collapse when failure occurs, i.e. there is no localised control policy in action and control only occurs via the central management system [Shu00].

It has been previously established that most NMS currently use a platform-centred client-server paradigm which does not fit the emerging communication networks any more, and in particular the transport network. Figure 1-3 shows an example of an optical transport network managed by a TMN-based NMS. In TMN, the communication between the manager and the agents is achieved via a separate *Data Communication Network* (DCN). The agents are physically implemented in the optical devices whereas the manager resides at the operator level.

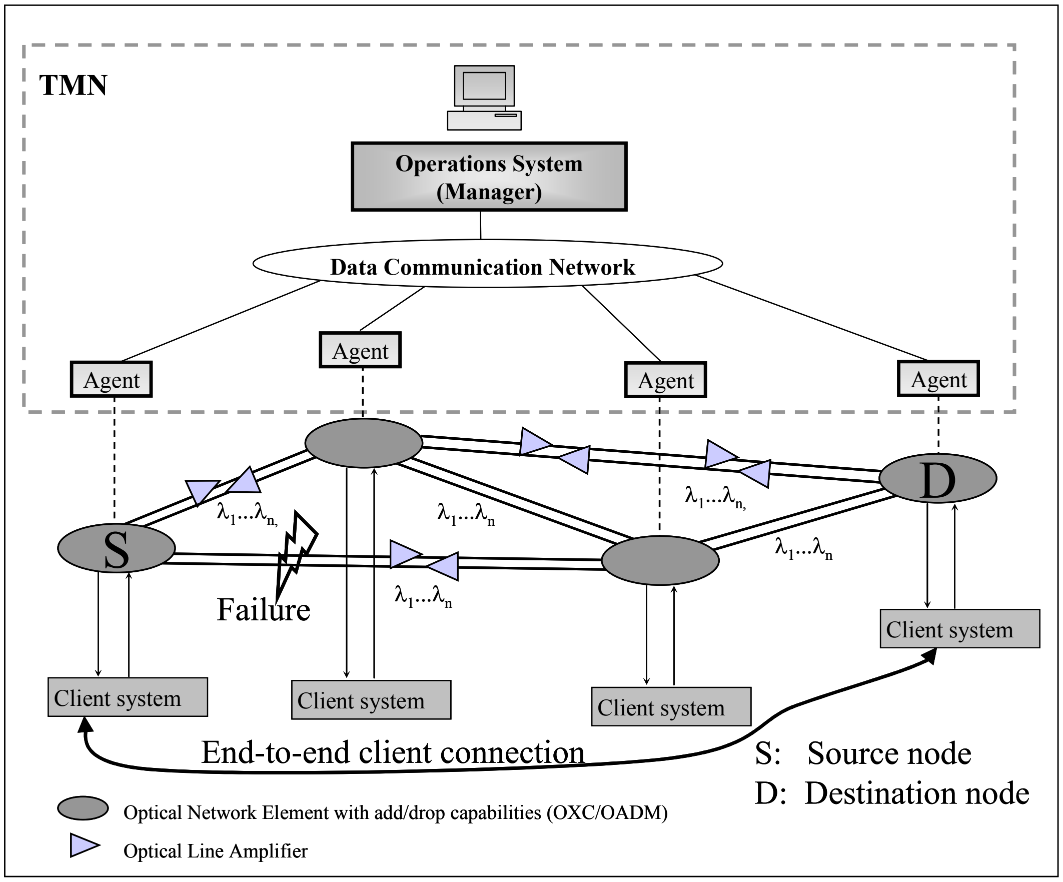


Figure 1-3. A TMN-based centralised approach for the optical transport network

A simple example of the drawbacks of a centralised approach can be illustrated with the following scenario: let us suppose an end-to-end client connection from node *S* to node *D*; during the running connection, a layer of an intermediate node fails, so that the wavelength transporting client data is not available any more. All the optical devices involved in the connection from node *S* to node *D* immediately detect a problem due to the absence of any signal and raise an alarm; according to the state of the local MIB, the residing agent of each node then informs the manager via the DCN. Consequently, the client systems also detect the problem and generate a *service disruption* alarm on the customer side. At this moment only, the manager can start an alarm correlation process in order to analyze the received alarm messages, to identify the defect node and to take appropriate action. According to the number of components involved in a connection, the manager obviously has to face a considerable amount of messages and may require complex algorithms. The manager will then send the re-configuration decisions back to the agent, including possible interaction with the customer equipments for synchronization purposes. Service restoration may consequently require significant time and thus have serious consequences on business costs.

On the contrary, a decentralised NMS do not require any communication with a manager; in other words, there are no critical components related to the network management. The decision logic – namely the *intelligence* – has to be implemented within the network component itself with the appropriate knowledge of the running service. A local processing in the network devices considerably reduces the number of alarm messages which have to be sent to the manager and customer equipments in case of failure. It favours fast reaction speed and rapid service restoration, so that the customer does not realize that there has been any service disruption.

Decentralised approaches, however, entail drawbacks of their own: a lot of computational entities are distributed over the network, and deploying and controlling these entities is not trivial; it is also more difficult to reach global optima.

Generally, *centralisation* presents serious limitations on scalability and resiliency of management. If the number of network components, the number of managed objects, or the speed of the network increases, or if management communications rates are bounded, the system quickly becomes unmanageable [KKL99]. When the network topology changes – i.e. network devices are added or removed - the legacy NMS experiences problems with the synchronisation of its MIB with the actual state of the network. Furthermore, the service logic is often implemented at the design time, which makes future changes difficult.

1.1.3 Distributed Management by Delegation

Distributed *Management by Delegation* (MbD) has been proposed as an alternative NMS allowing to achieve decentralised network management [YGY91][GY95]. MbD is based on the notion of delegation agents. Delegation agents are programs that can be dispatched to remote processes and dynamically linked and executed under local or remote control. They are used to perform tasks such as real-time monitoring, analysis and control of network resources. MbD relies on the concept of *elastic* processing and on an application-layer protocol called *Remote Delegation Protocol* (RDP). An elastic process is defined as an executing incarnation of a program that can be modified, extended and/or contracted *during* its execution by means of delegation agents. The delegators use RDP to *transfer* the code of a delegation agent to an elastic process and to control its execution. RDP also makes the communications between delegating processes possible. Examples of operations supported by RDP are: *delegate/delete*, *instantiate/terminate*, *suspend/resume* and *getstate/setstate*.

Distributed MbD can be used to distribute the management logic into network components. The delegated agents enhance a network component and its OSI agents with advanced processing so that management operations – configuration, monitoring, fault detection – can take place in the device itself. Several delegated agents can be dynamically instantiated and cooperate with one another.

1.2 SOFTWARE AGENTS

Intelligent and mobile agents constitute a natural extension of the MbD concept through the addition of advanced interaction mechanisms and knowledge manipulation, two important research fields of distributed artificial intelligence.

1.2.1 Intelligent Agents

Intelligent agents result from the conjunction of two major research streams: *distributed artificial intelligent* and *software engineering*. This approach includes numerous research works in other fields such as decision theory, network communication, biology and psychology. Intelligent agents are considered as the most promising approach to address issues related to distributed applications in the rapidly expanding communication industry [HaB99].

As we focus on network management, our analysis will be restricted to specific properties of *Multi-Agent System* (MAS) which are relevant to the definition and the implementation of management functions such as resource allocation, routing, monitoring and fault detection. In this thesis, mobile MAS refer to multi-agent systems composed of mobile agents. From the point of view of implementation, we shall focus on *Java*-based MAS because of the current world-wide adoption of *Java* in the development of agent systems. In addition to RMI, the *Java* virtual machine provides efficient mechanisms for code serialization that are intensively exploited by mobile agent systems.

There is no world-wide adopted definition of what an intelligent agent is. We can however propose a working definition of agents and intelligence as follows: *"Intelligent agents are defined as being a software program that can perform specific tasks for a user and possesses a degree of intelligence that permits it to perform parts of its tasks autonomously and to interact with its environment in a useful manner."* In this context, "*The intelligence means that the agent is provided with knowledge of the user's wishes and also makes use of this knowledge*" [BZW98].

It is commonly accepted that an intelligent agent must have the following properties [WJ95]:

Autonomy – the agent is capable of following its goal autonomously that is, without interactions or commands from the environment. The agent must have both control over its actions and internal states, and be provided with the resources and capabilities required to perform its tasks.

Reactivity – the agent is capable of reacting appropriately to influences or information from its environment. The agent must therefore possess its own internal environment model in order to be able to react to changes in its environment.

Pro-activity – Under specific circumstances, the agent can take the initiative to perform appropriate actions. For example, a predictive system will lead the agent to make decisions automatically in order to reach better performance.

Social ability – the agent is able to communicate with other agents and to interact with its environment in order to fulfil its tasks. At the agent level, there is no distinction between client and server, although the underlying mechanism resorts to the previously described distributed system architecture. The social ability also refers to the ability of an agent society to perform a common ultimate goal although the agent itself has no knowledge of this goal.

In addition to these properties, an agent can be *mobile*, so that mobile agents can migrate from one location to another. If it is not mobile, the agent is a *stationary agent*. Additional features such as proxy, rational, unpredictable, transparent and accountable, etc. [OMG0] may characterize the intelligent agent, but they are not considered in this thesis.

We believe that the agent's social ability constitutes a fundamental property making the agent *intelligent* and achieving self-organization in distributed environments. Social ability enables the agents to exchange information – also called *knowledge* – with other agents in a structured way, so that the multi-agent system exhibits a "social" behaviour and organizes itself in order to reach the objectives assigned to the agents. The agents must therefore have a representation of their environment, as well as capabilities to "understand" messages issued by other agents.

Communication in MAS is achieved by means of an *Agent Communication Language* (ACL), thanks to which the agents are able to "discuss". The speech act theory and other research fields in Artificial

Intelligence focusing on communication within human societies have provided agent communication with communicative acts – or *performatives* – which clearly separate the intention or the action defined by the message from the message content itself. The agents are endowed with the ACL and can therefore perform conversations with other agents and participate in *coordinated* activities. These interactions can be associated with different approaches, according to the individual agent behaviour. Either the agent acts as a self-interested entity competing with the other agents, or the agent's activities participate in the elaboration and achievement of common objectives in a cooperative or collaborative way [HB01].

We therefore propose to alter the above-mentioned definition of *intelligence* and to integrate the agent's ability to deal with other agents in a coordinated way so that the whole system exhibits social behaviour.

While distributed systems are generally based on a *top-down* approach, multi-agent systems promote a *bottom-up* process during the development phase that is, they concentrate on the form of the actual agents instead of being primarily concerned with the division of problems. Multi-agent systems should therefore not be developed for a specific task, but for the common solution of problems. According to this approach, the extension of the system with new agents does not require the existing infrastructure to be changed or the running agents to be interrupted so that the current system can be improved and new functionalities can easily be added. This perspective of extensive collaboration mechanisms allows services to be composed dynamically; experiments in that direction are conducted within the *Agentcities* project (see Section 1.5), for example.

AGENT MODEL

Beside the communication and interaction mechanisms proper to multi-agent systems, the implementation of the agent behaviour can follow different approaches according to the agent model.

A famous agent model is the *deliberative* agent introduced by Rao and Georgeff [RG$^+$95] and based on the *Belief-Desire-Intention* (BDI) model [WJ95][PNJ99]. The deliberative agent has an internal representation of the environment (belief). The efforts undertaken by the agent to attain its goals result from its desires. Desires are typically generated in response to changes in the environment or interactions with other agents; a special process then selects specific desires that will become the agent's intentions for future endeavours.

Another architecture model based on *reactive* agents has been proposed by Brooks [Bro86] and is mainly inspired from robotics and reactive systems: this quite simple architecture is composed of different competence modules linked with sensors (input, perception of the environment) and actuators (output, actions in the environment). Information can be exchanged between the competence modules and the reaction to external changes occurs faster than in the deliberative agent model.

Both models are depicted on Figure 1-4.

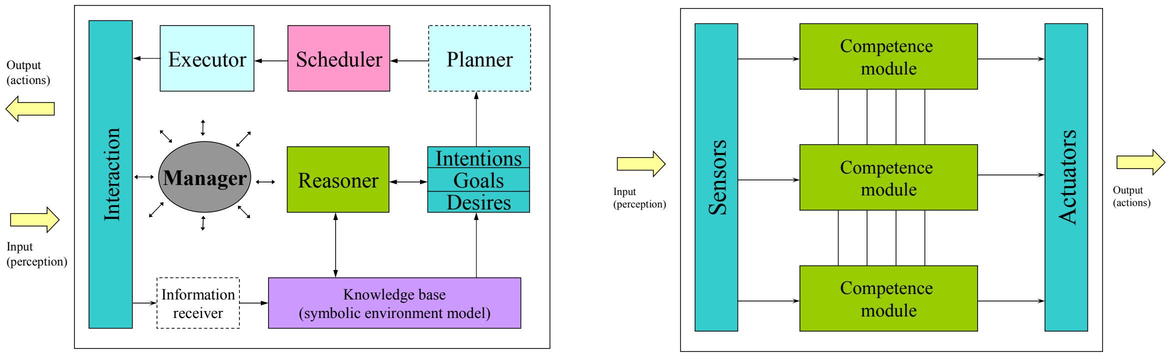


Figure 1-4. Two famous agent models: the deliberative agent (left) and the reactive agent (right)

In the field of network management, multi-agent systems implementing advanced negotiation mechanisms often resort to the deliberative agent model. In this context, market-based approaches are used to negotiate bandwidth or Quality-of-Service in telecommunication networks. The agents enter a virtual market place and negotiate goods. Examples of projects in this area are IMPACT[1], in which agents negotiate bandwidth in an ATM network, and SHUFFLE[2] in which agents manage the interactions between service providers and network operators. In the latter case, the agent model is a hybrid deliberative/reactive agent model called *Interrap* architecture.

Deliberative agents are stationary agents; they generally have a respectable code size and are not suitable for migration; the objectives for which they are designed generally do not require explicit mobility within the network. These agents nevertheless play an important role in the network since they are able to reason on a large data size, to make predictions, to compute and to activate action plans, so that they can be regarded as "more intelligent" than reactive agents, intelligence also referring in this context to the interaction mechanisms taking place within the multi-agent system.

Ontology

In the context of intelligent agents, *ontology* is a collection of terms and rules defining and governing a certain domain. Agents use ontologies to limit the scope of their interactions and focus on a specific semantic world; ontologies are therefore important components of the agent communication. The set of terms and rules is then described by means of a *content language* such as *Semantic Language* (SL0)[3] or XML/RDF as an integral part of the ACL.

Intelligent Agents and OSI Agents

In the previous sections, we have introduced the OSI agent which provides and controls the access to managed objects (MIB). The term "agent" in this case denotes a process which has its own execution context, internal state and data, which is able to react to specific external events, such as alarms or notifications, and which can communicate with a manager. OSI agents however must not be considered as intelligent agents as they rely on a pure client-server paradigm and do not support any form of social

[1] http://www.acts-impact.org
[2] http://www.ist-shuffle.org
[3] FIPA SL Content Language Specification available at http://www.fipa.org

ability in interaction with other agents. OSI agents are autonomous processes which implement a simple computational model based on a finite state machine most of the time.

INTELLIGENT AGENTS IN NETWORK MANAGEMENT

Intelligent agents have been considered for network management in numerous research projects[1] [HB99]. As we have already seen, multi-agent systems may be used to negotiate network resources such as bandwidth or QoS in a fully dynamic and automatic way [GJ98]. The automatic negotiations take place between user agents, resource agents, brokering agents, etc. Network resource utilisation is optimized and the resource price is kept as low as possible with respect to the objectives and contractual agreements pre-established between the client, the service and the network provider through a *Service Level Agreement* (SLA). This approach has been considered for example in IP and ATM networks, and more recently in UMTS networks [LRD$^+$00]; in the latter, the negotiation takes place between business entities such as service providers and network operators.

Among other things, intelligent agents are also investigated for on-line routing algorithms. A possible approach consists in defining a hierarchical society of intelligent agents which are responsible for maintaining the route configuration dynamically between multiple management domains [CFF99].

Despite their significant contribution towards a decentralized network management, stationary intelligent agents may require extensive computing resources when they are dedicated to long-term planning for example, and their deployment is not easy since dynamic remote instantiation of an agent is generally not allowed by the agent platform; the agent therefore needs to be instantiated manually in appropriate network locations where it must reside; upgrading a new release of agent code is consequently not trivial and requires human intervention at each agent site.

The computational model of agents is one of the major issues addressed in this thesis as will be shown in Chapter 3; we propose to decouple the agent's tasks from the agent lifecycle so that the tasks can be loaded dynamically without any intervention on network devices.

1.2.2 Mobile Agents

The mobile agent paradigm has received particular attention over the last few years and the community of researchers concerned with this subject is growing steadily. Mobile code, as well as mobile agents, will be a critical near-term part of the Internet because it provides a general framework in which distributed information-oriented applications can be implemented in a natural and efficient way with advanced useful features [KG99]. The mobile agent technology can be considered as a powerful extension of the object-oriented paradigm [E712_98]. Considering mobile agents in network management entails numerous advantages: mobile agents reduce the requirements regarding traffic load and regarding the availability of the underlying networks, they reduce the time and effort required for the installation, operation, and maintenance of service intelligence for resource control and management; they allow on demand provisioning of customized services and they lead to more decentralized realization of service control and management software, by bringing control or management agents as close as possible to the resources [BHM98][BPW98]. In that sense, mobile agents match perfectly the distributed *Management by Delegation* (MbD) approach described in Section 1.1.3.

[1] An excellent overview of project activities concerning intelligent agents for network management can be found in [HB01].

Mobile agents are closer to reactive agents than deliberative agents. Reactive agents are generally small, having a reasonable code size with a low processing time, and are used for modelling mobile entities (robots). The completion of objectives in a mobile agent system consequently relies on the mobility paradigm rather than on reasoning. If mobile agents require more processing, such as reasoning mechanisms using a rule engine for example, they need to delegate the reasoning task to a local *stationary* component requiring further interactions with the environment.

We therefore emphasise that mobile agents must be able to interact with stationary agents by sharing a common environment. As we will discover in the implementation part of this document (Section 4.4), mobile and stationary agents can share a common agent platform.

The mobility paradigm enhances multi-agent systems with new capabilities such as *migration*, *cloning*, *emergent behaviour* or *meeting-based coordination*. These functions will be exploited in *Ecomobile* to address several issues related to mobile multi-agent behaviour, such as population size, dissemination and activation of tasks, inter-agent coordination and inter-tasks cooperation.

The agents' mobility allows the code, state and data to be moved from one location to another; the agent is executed in an agent system which provides migration, security, localisation and additional services facilities. The location of a mobile agent can refer to the agent system location (physical mobility) or to a virtual location within the same agent system (virtual mobility).

Figure 1-5 shows a general environment of a mobile MAS as proposed by the OMG-MASIF specification (see Section 1.2.4).

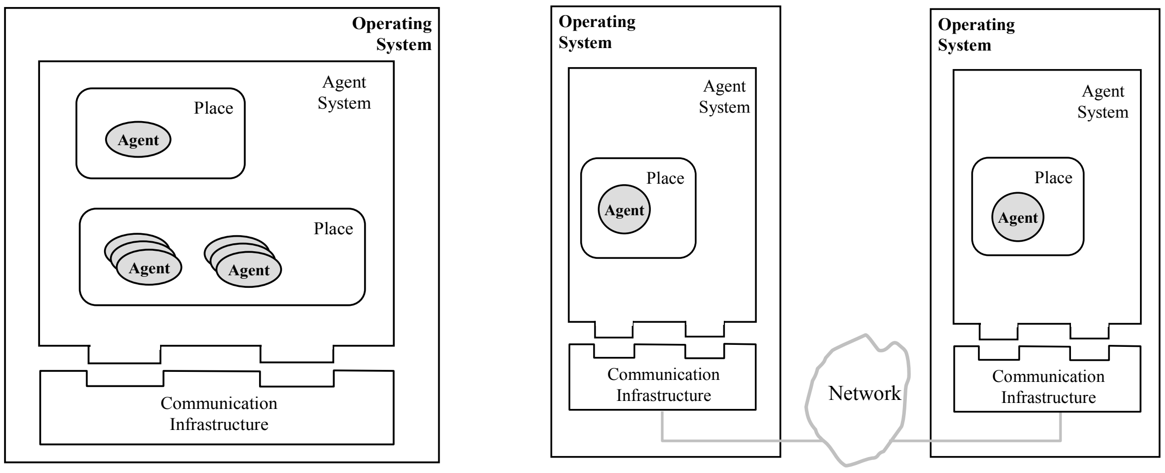


Figure 1-5. Mobile agent environment (left) and communication between agent systems (right)

The agent system can manage one or several places depending on the application and the organisational architecture; the communication infrastructure relies on standard distributed system technologies.

An efficient implementation of a mobile MAS will therefore depend on the ability of network devices to *host* the agent system or to interact with external operating systems (for example via a proxy)[1].

[1] An overview of mobile agents based applications can be found in [PieJan01].

1.2.3 Mobility Functions

In addition to the concepts developed in Section 1.2.1 devoted to Intelligent Agents, the mobility paradigm refers to a wide range of new concepts. *Migration* is undoubtedly the most important of these concepts. Migration allows an agent to move from one location – or *place* – to another, the place being a physical place – a node, a network device, a machine – or a virtual place, in which case the application requires specific semantics of place; for instance, a place can represent a class of service, a type of communication channel, a specific medium, etc. The migration of a mobile agent requires the agent system to support execution stopping, state collection, data serialisation and transfer, data de-serialisation and execution resuming. From this point of view, mobile agents strongly rely on mobile code technology.

Another important concept is agent *cloning*: the agent can clone itself that is, a new mobile agent is created as a copy of the parent. A pure cloning operation implies that the cloned agent has the same behaviour (code) and the same knowledge (data) as the parent agent. A post-cloning operation can initialize specific values in the cloned agent which starts its lifecycle in the same execution environment as the parent. Its location can however be different, according to the agent systems; some of them offer the possibility to start the execution of a child agent in a location different from the parent's one.

Spatial and/or meeting-based coordination mechanisms are new concepts issued from mobile MAS: since mobile agents are moving within the network, they can meet other agents everywhere in the network. *Emergent behaviour*, which is related to the coordination of the mobile MAS, will be detailed in Chapter 2. All these concepts will be explained and developed over this thesis.

1.2.4 Agent Standards

Agent standards enable the interoperability between agent platforms so that intelligent agents can communicate and achieve their objectives according to standardized specifications. The development of agent standards in telecommunication is therefore a *sine qua non* condition for the successful deployment of software agents in large-scale networks. In this section, we examine the two most popular agent standards: *FIPA* and *OMG-MASIF*.

FIPA

The *Foundation for Intelligent Physical Agents* (FIPA)[1] was formed in 1996 to produce software standards for heterogeneous interacting agents and agent-based systems. Currently, FIPA appears to be the dominant standards organization in the area of agent technology. Important efforts have been made to address the inter-operability issues between the agent platforms. Figure 1-6 presents the overall architecture of an agent system as specified by FIPA. The message transport is the main underlying mechanism devoted to the ACL-based communication between agents; at this stage, mobile agents are not supported. The message transport itself relies on standard communication techniques used by distributed system framework such as CORBA or *Java* RMI.

Both *Agent Management System* (AMS) and *Directory Facilitator* (DF) are FIPA agents: the AMS is responsible for the core management activities of the agent platform whereas the DF acts as a *yellow page* service. Agents are registered in the DF and can be localised from their types by other agents. In addition, the agent communication is ensured through the *Message Transport System* (MTS) including the *Message Transport Protocol* (MTP) and the *Agent Communication Channel* (ACC) which directly provide agents

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

with specific services for communication. The ACC may access information provided by the other agent platform services such as the AMS and DF to carry out its message transport tasks.

From the communication point of view, the agents can interact via *intra-platform* communication; all agents participating in the interaction are managed by the same platform; they *reside* in the same node. On the other hand, the agents can be distributed over several nodes; in this case, they interact via an *inter-platform* communication mechanism. In both cases, agents communicate via ACL messages and use the services provided by the ACC.

In addition to the agent system reference model, there are FIPA specifications concerned with ACL message format, ontology, interaction protocols, etc.

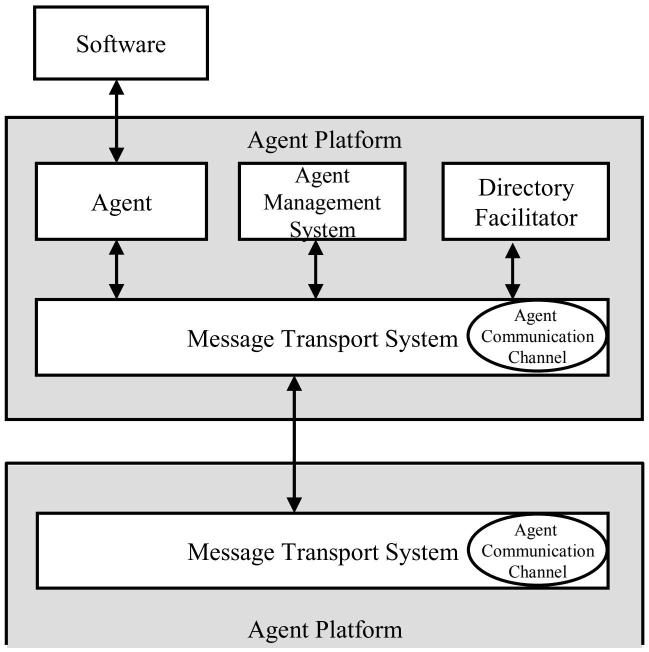


Figure 1-6. Agent system reference model of FIPA.

Although mobile agents are currently not supported by FIPA, we can find a specification for the minimum requirements and technologies allowing agents to take advantage of mobility. The specification includes a wrapping mechanism for existing mobile agent systems in order to promote interoperability [FIPA01]. In short, the specification defines mobility protocols addressing: 1) agent *migration*, 2) agent *cloning* and 3) agent *invocation*.

Little work towards the development of a FIPA compliant mobility framework has been accomplished so far. For example, a special agent called *Mobility Management System* (MMS) has been designed to provide mobile agents with mobility services [Mak00]; in this work, however, it is unclear how the mobile agents are defined in terms of architecture and how they physically migrate from one location to another.

Nevertheless, current FIPA specifications, under certain conditions, perfectly fit the requirements to deploy mobile agents in a FIPA environment as we will discover with the deployment of *Ecomobile* in Section 4.4 and with the *FIPA-mob* project in Section 4.5.

OMG-MASIF

In 1997, the OMG released a draft version of the *Mobile Agent System Interoperability Facilities* (MASIF)[1] [MAF98]. MASIF proposes a specification of the communication infrastructure as well as interfaces defined in IDL to access mobility services in order to promote the interoperability and the diversity of mobile agent platforms. From the interoperability and heterogeneity perspective, OMG follows the same objectives as FIPA. The objectives in term of requirements and functionalities are clearly different, however. Whereas FIPA is concerned with a message based communication infrastructure, MASIF has to take into account the migration of the agent and must consequently focus on the way to dynamically *create the agent* that is, to instantiate a new object at the right place and with the right class.

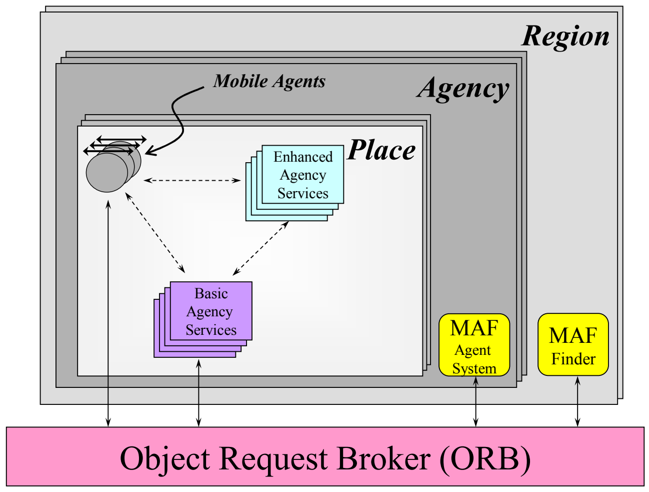


Figure 1-7. General architecture of OMG-MASIF mobile agent system

On Figure 1-7, the MASIF architecture appears to be a hierarchical organisation of regions, agencies and places [BM98]. The *place* is a context within an agent system in which an agent can execute its tasks and provide local access control to mobile agents. A place is associated with a location, which consists of the place name and the address of the agent system within which the place resides. The *agency* represents the agent system itself or is the core part of the agent system. At a higher level, the *region* is a set of agent systems that have the same authority, but are not necessary of the same type.

Considering its origin, MASIF strongly relies on a CORBA architecture and therefore on the ORB. The services provided by the region, agency and place are defined through IDL interfaces; the most important interfaces are the *MAFFinder* and the *MAFAgentSystem*: whereas the *MAFFinder* supports the localisation of agents, agent systems and places in the scope of a region or in the whole environment, the *MAFAgentSystem* interface provides operations for the management and transfer of agents. In MASIF, the agent's migration requires the transfer of the agent class so that the agent can be properly instantiated. Different mechanisms are proposed to achieve the class transfer. Either the agent class (including all dependent classes) are automatically transferred when the migration is invoked or, when the class is not known to the destination agent system yet, the class is transferred on demand.

[1] Originally, MASIF was called *Mobile Agent Facility* (MAF) by OMG

In this thesis, we borrow the concept of *agency* from MASIF and define it as the entity responsible for providing the mobile agents with mobility services and with the execution environment. We define the place as *location concept* and *coordination space*. These concepts will be described in details in Chapter 3.

INTEGRATION OF FIPA WITH MASIF

In the previous sections, we have presented the ongoing efforts realised by FIPA and OMG to promote the interoperability between agent platforms. Thanks to these efforts, agents residing on different nodes should be able to communicate even if they are not using the same agent system. In a powerful exploitation of agent based applications, mobile agents are expected to interact with stationary agents, using an ACL language for example. This kind of interoperability however suffers from the lack of standards. Whereas the agent reference model defined by FIPA does not depend on any implementation, MASIF heavily relies on CORBA; furthermore, most mobile agent platforms are developed in *Java* and rely on its serialization mechanism. No standards for agent transport and agent encoding are available at the moment. Still, FIPA has proposed a specification for mobility support by combining the agent platform with the MASIF architecture, as described in Figure 1-8. The AMS fits the *MAFAgentSystem*, the DF fits the *MAFFinder* and communication relies on ORB-IIOP.

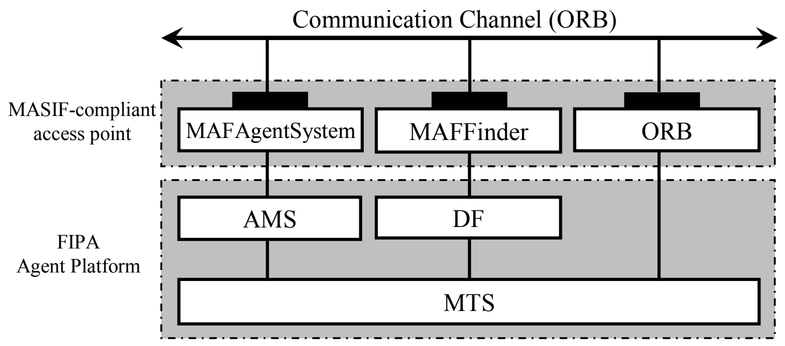


Figure 1-8. Integration of the FIPA mobility support with OMG-MASIF

The relationships between FIPA and MASIF components depicted on the figure require particular attention because they are not to be considered as direct agent-to-agent interactions; whereas the FIPA components correspond to FIPA agents, the MASIF components correspond to IDL interfaces (*MAFAgentSystem* and *MAFFinder*) which do not include implementation, on the one hand, and to the ORB, on the other hand. The associations therefore express a functional equivalency rather than a communication scheme.

At the moment, there is no further activity concerning MASIF, whereas FIPA gathers a growing community of agent researchers. This is why we believe that a successful deployment of mobile MAS will depend on the adoption of a clear concept of the mobility paradigm by the FIPA community and on the definition of new specifications in this area.

1.3 MOBILE PROCESSING IN NETWORK MANAGEMENT

In this section, we shall first present the various mobile agent based approaches which have been considered to implement management functions for different network environments. We will then try to highlight the most important characteristics of each mobile agent based solution in order to define a range of interesting properties; our reflections will lead to the choice of three abstraction models and to their implementation in Chapter 2.

Particular emphasis will be laid on future network technologies in which the management of resources and services constitutes a particularly interesting field of application and which provide rational execution environments for the support of mobile MAS.

1.3.1 Main Characteristics of Mobile Agent Based Approaches

Mobile agents for network management constitute an emerging research field and new projects related to this domain constantly appear in academic and industrial research labs. Although it seems impossible to give an exhaustive account of this research in the scope of this work, we will try to cover the most important approaches for mobile agent based network management relevant to the context of *Ecomobile*.

THE PERPETUUM MOBILE PROCURA PROJECT

The *Perpetuum Mobile Procura* (PMP) Project [Bie97][Riv00] from Carleton University in Ottawa (Canada) aimed at the implementation of mobile agents for the management of networks. Although the concepts developed in the scope of this project are generic enough for any kind of fixed network, the project mainly focused on IP networks.

In the context of this project, a taxonomy of mobile code leads to the definition of various kinds of mobile agents [BP98] which are supposed to evolve in the same infrastructure in order to fulfil all the functional areas defined by the OSI management model: *fault*, *accounting*, *configuration*, *performance*, and *security management*. Since the project is concerned with IP networks, the agents have the capability to deal with SNMP agents.

We now propose an overview of the main agents defined in the PMP project.

The ***netlet*** is a mobile agent supposed to move in the network in a permanent way by executing specific tasks in each visited node and therefore never terminates. Typical applications of *netlets* are automatic network discovery and network monitoring. The development of *netlets* implies considering various concerns such as *security* and *density control*: the proliferation of *netlets* in the network has to be avoided; migration patterns or policies also have to be elaborated in order to determine how the agents can migrate.

The ***deglets*** are used to perform a specific task in a network node and can consequently be associated to the concept of delegation agent in MbD. In the context of PMP, *deglets* are typically used to interact with OSI agents (CMIP/SNMP agents).

The ***extlet*** is a downloadable or uploadable code extension that expands the receiving party, but does not extend its interface protocol .

The ***servlet***[1] is an uploadable code extension that expands the capabilities of a remote server by extending its interface protocol.

The ***applet***[2] is a mobile code that represents a downloadable application.

[1] *Servlet* reflects concepts which are similar to Java Servlet technology.
[2] *Applet* is similar to Java applet technology, in which the application code is moving from a Web server to a Web client.

The ***piglet*** is a mobile code that has been intercepted and maliciously altered.

All these agents run in an environment called *Mobile Code Environment* (MCE), which is depicted on Figure 1-9 and which provides several components, such as a mobile code daemon, a migration facility, an interface to managed resources, a communication facility, and a security facility. It has to be noted that the MCE does not explicitly rely on existing standards.

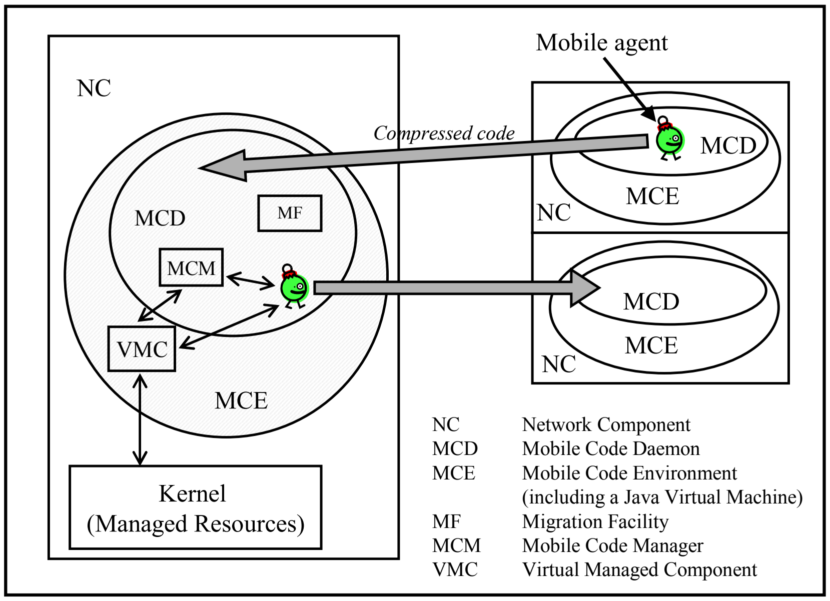


Figure 1-9. MCE Components

It is assumed that a mobile code daemon runs within a *Java* virtual machine on each network component. The *Virtual Managed Component* (VMC) provides *get*, *set*, *event* and *notification* facilities with an access control list mechanism used to enforce security. VMCs are designed to contain MIB and vendor-related information.

The MCE has been extended to support the standard DPI[1] protocol in order to enhance the interaction of mobile agents with SNMP agents [PWW00]. Thus, it is possible to use a DPI-based MCE for the automatic configuration of permanent virtual circuits in heterogeneous ATM networks.

Let us now examine typical scenarios introducing *netlets* and *deglets*.

A combination of *netlets* and *deglets* can be used to create and maintain a network model subject to dynamic changes [WPB99][WPB$^+$98].

By network model, we understand a representation of the different *Network Elements* (NEs) at a network management workstation with their available interfaces to management functions. The software components which provide mobile agents with interfaces are assumed to be vendor-dependent and

[1] Distributed Protocol Interface (DPI) is an extension of SNMP agents that permits the dynamic addition, deletion or replacement of management variables in the network component's SNMP MIB without requiring recompilation of the SNMP agent.

implemented into a local *Java* virtual machine; the protocol used for interaction with the resources may therefore be proprietary.

The *netlets* are injected into the network from a network management workstation and visit the network elements, using either a pre-configured itinerary or a default migration strategy based on auto-discovery mechanisms. For example, the agent is given an itinerary of devices to visit that has been generated by the action of a standard network discovery algorithm, or the agent simply follows the default migration path connecting mobile code daemons in the network, which is assumed to form a logically connected graph. Additional rules can be statically or dynamically implemented to influence the migration when necessary.

The *netlets* has the ability to spawn a "configuration" *deglet*. The network element state is then copied into the *deglet* which interacts with the local NE to get the information required for the NE's configuration. The "configuration" *deglet* in turn spawns a "model provisioning" agent which immediately moves towards the network management workstation in order to update the network model. The "configuration" *deglet* dies in the local NE.

Such an approach to the dynamic network configuration and monitoring problem has been also proposed, for example, with the *Distributed Network Management and Monitoring System* (DNMMS) architecture [Sha98]. Whereas an *Embassy* based on the *proxy* pattern is deployed in each network node and provides mobile agents with local interfaces, the *CountryBase* acts as a Web manager which is able to inject mobile agents within the network in order to perform specific tasks.

SWARM INTELLIGENCE

While the MCE enables the transfer of code from one component in the network to another and the principle of delegation provides a reason to use it, it does not provide for distributed problem solving groups or societies of agents. In this perspective, it is interesting to study an approach based on *Swarm Intelligence* [WP99] which has been adopted as an underlying mechanism for fault detection and localisation in networks. Swarm Intelligence is a property of systems composed of unintelligent agents of limited individual capabilities that collectively exhibit intelligent behaviour. This *emergent behaviour* is studied in Chapter 2.

In Swarm Intelligence, *netlets* act as "insects" which have the ability to deposit a chemical track in the environment that is, in the network node. Other agents are influenced during their migration in response to this chemical message, so that fault detection and location determination can actually arise as a result of the trail-laying behaviour of simple problem agents. The concentration of the chemical message is influenced by the change in value of the characteristic of the monitored service: the modification of the concentration depends on whether the change of this value is considered as beneficial to the service, in which case the chemical track will be "evaporated", or detrimental to the service, in which case the chemical track will be reinforced. A problem occurs when the chemical trail reaches certain limits. In this case, the agent can take appropriate action by informing the management workstation or by re-configuring the network.

Another approach towards fault detection consists in building an extended version of *netlets* – called *smartlets* – which includes reasoning mechanisms and therefore improves the analysis and filtering of alarms and associated data [EB99]. The *smartlet* interacts with the VMC to collect status data concerning the network component. A JESS[1] parser object is then created and used to parse the data collected and to apply the inference engine producing a set of decisions and conclusions about the status of this network and the origin of the faults. Results are then returned to the *smartlet*, by filtering and correlating the alarms, the *smarlet* localizes faults and isolates them. The output of this procedure is a set of network addresses of the network components generating the faults; the *smartlet* is then able to visit the defect components.

Fault diagnosis and network reconfiguration obviously constitute a central part of an NMS in which distributed *Artificial Intelligence* (AI) may significantly improve data processing, since it provides powerful reasoning techniques, particularly useful to process large amount of data, and the capacity to distribute processing over multiple systems. The quantity of information exchanged across the network may however lead to serious drawbacks and therefore constitutes a trade-off between efficiency and accuracy [Lec95]. In this context, we consider interactions between mobile agents and stationary "reasoning" agents as a necessary requirement for the elaboration of a flexible mobile agent based NMS.

MOBILE AGENTS IN INTELLIGENT NETWORKS AND MOBILE NETWORKS

Although *Intelligent Networks* (IN) and mobile networks do not constitute our major concern, we consider that it is necessary to expose the mobile agents based techniques used for their management from the perspective of generic agent design, so that we can benefit from these advantages in order to develop *Ecomobile*.

Intelligent[2] *Networks* (IN) constitutes an architectural framework for the rapid and uniform provisioning of advanced telecom services overstepping the Plain Old Telephone Service (POTS), such as call forwarding, private numbering plan, incoming call screening, etc. IN services are based on additional service logic and data, on top of different switched telecommunication networks. Centralized service nodes, called *Service Control Points* (SCPs), control the bearer switching nodes known as *Service Switching Points* (SSPs) which provide only the basic call processing capabilities. This control is achieved by means of the international *Signalling System No.7* (SS7) network. A dedicated outband signalling network is set up for this purpose. The IN Application Protocol (INAP) is implemented on top of the SS7 network and enables "real-time" connections between the switches and the service node.

In [BrM98][BBC$^+$98], a mobile agent based approach is proposed to provide IN networks with advanced services such as the call forwarding service provision and the a MA-based *Virtual Private Network* (VPN) service. The former allows the users to initiate an automatic routing of incoming calls to other destination devices, depending on the time of the day or on specific events, whereas the latter allows the customers to define a VPN on top of an IN infrastructure; the customer's lines, connected to different switches, constitute the VPN. In both cases, the mobile agent systems have been implemented with the *Grasshopper* agent platform (see Section 1.5 and consequently rely on MASIF. Several agencies are deployed, i.e. the Switch, End User, Customer and Provider agencies. Considering the reuse of the existing IN switching nodes, including the existing call models and IN interfaces, appropriate adaptation units have to be provided within the agencies, in order to handle the translation of INAP request coming from SSPs into object method invocations of the agents implementing the service features on a remote

[1] *Java Expert System Shell* (JESS) - http://herzberg.ca.sandia.gov/jess - see also section 1.5.2.
[2] In this case, the term "intelligent" refers to the capabilities of IN to process the call services in a fully automatic way; it is not related to the concept of "intelligent agent".

SCP, or even within an SSP/service node, and vice versa. IN networks can benefit from translation mechanisms similar to those found in CMIP/SNMP (see Section 1.1.1).

The mobile agents that have been developed for IN networks are similar to delegation agents or *deglets*. They are pre-configured by the customer, launched at the required place in the network and perform a configuration task in the different network devices by using specific gateways. Eventually, they can monitor the connections and react in case a problem appears or the switch needs to be re-configured. They do not exhibit any particular behaviour or cooperation ability.

Since mobile agents perform task locally, they are particularly useful when the connection between nodes is not permanently guaranteed as in mobile networks like GSM and UMTS. In this kind of network, the transmission quality changes during the connection and the transmission can temporarily disappear. When the connection is dropped, the agent can continue to execute its task in the mobile device and decide to migrate once the connection has been restored.

For example, mobile agents are proposed for managing the *Virtual Home Environment* (VHE) in UMTS networks. The VHE provides the user with a service environment which does not depend on his current location or on the home provider. A general approach of a mobile agent based VHE management using the MASIF architecture – agencies at the different business entities, with mobile agents migrating between end-user mobile devices and the provider - can be found in [HMW99]. In this context, mobile agents are mainly used as "configuration" agents and they do not exhibit any particular behaviour.

Future heterogeneous networks will support more and more multiple independent channels with respect to customer requirements, different multiplexing techniques will be implemented and the routing mechanism will have to rely on accurate knowledge of the network connectivity.

As mobile networks require dynamic updates of the network state regarding the connectivity and the modularity of mobile agents, the latter's ability to monitor the network continuously makes them very attractive as regards routing information management. We propose to examine a mobile agent based approach devoted to managing ad-hoc networks, called *MITAgent.*

THE MITAGENT PROJECT

An approach based on a population of cooperating mobile agents has been proposed at the MIT [MKM99][MKM98][KMM99]. Although this approach remains applicable to other networks, it suits mobile networks perfectly, and is particularly favourable in ad-hoc networks (see Section 1.4.2). Since this research was originally proposed at the MIT, we propose to refer to it as *MITAgents*.

The experiments were performed with discrete simulation and conducted on a network of nodes modelled as radio-frequency transceivers distributed on a two-dimensional space. Adjacent nodes could exchange routing information in the same way than IP routers. Three kinds of agents have been implemented: *random* agents, *conscientious* agents and *superconscientious* agents, according to the degree of collaboration required.

Random agents do not collaborate with one another and migrate randomly within the network, while conscientious agents exchange knowledge when they meet within a node; although they assimilate the data, they base their movement decisions entirely on their own first-hand experience. As for superconscientious agents, they use both first-hand and peer-obtained information to make movement decisions. Results have shown that the cooperation between conscientious agents significantly increases the system's performance. Superconscientious agents are still more efficient than conscientious agents but only in a small population: when they meet, they exchange the necessary knowledge used for subsequent migration decisions. After several meeting events and knowledge exchange – the number of meetings increases with the size of population -, superconscientious agents tend to choose identical paths to migrate because their internal knowledge is the same.

The number of agents living in the network is fixed at the beginning of the simulation and the communication between agents is instantaneous; no deployment on an agent platform has been considered.

DISCUSSION

In the approaches we have just described, mobile agent systems can be divided into two categories according to the agents' behaviour.

The first category relates to reactive agents which purely act as delegation agents (*deglets* or agents for IN services) and exhibit the following properties: ability to transport customer-profile information and to deal with SNMP/OSI agents, absence of interaction scheme and simple lifecycle (single activation and then disappearance from the system). In most cases, they visit nodes with a pre-planned itinerary.

The second category relates to reactive agents which populate the network and run continuously. The agents' population remains fixed so that, if an agent disappears from the network because of a network failure, for example, the population of agents is reduced, which might lead to dramatic performance issues. Because of their continuous presence within the network, these agents are more appropriate for inter-agent cooperation mechanisms.

In this thesis, we propose a particular mobile MAS architecture dealing with these two kinds of agent in a transparent way.

1.3.2 The Wave Technology

The *Wave* technology is an interesting approach tackling a wide range of distributed problems and using mobile processing techniques; it is particularly well suited to the simulation of mobile entities. *Wave* can be used for any kind of distributed problems beyond the field of network management.

The *Wave* technology relies on an interpreted programming language which is based on parallel and asynchronous spreading of a special recursive program code, known as "waves", in computer networks. The waves navigate in the existing networks or create new virtual *knowledge networks* (KNs) reflecting the structure and organization of the worlds to be modelled or controlled [S99]. Although conceived several years ago, only recently was the *Wave* technology actually recognized as a valuable emerging technology addressing distributed problems in an open environment. Although *Wave* is not actually a mobile agent technology, it implements several concepts which are similar to those introduced in our work, that is:

- An efficient framework for investigation, simulation and efficient control of distributed, open and self-organized systems,
- Mobility as a key concept for the support of distributed applications
- Mobility functions, such as cloning or meeting-based coordination

Wave basically requires a virtual network of interconnected nodes that is dynamically created by the program. Once the network has been created, the program can navigate freely and has access to the node environment.

Spatial variables are defined within a node to store information or within the wave itself to constitute its internal knowledge. Spatial variables belong to one of the two following categories: *task* variables and *environmental* variables. *Task* variables are defined inside the wave (*frontal* variables) or inside the node (*nodal* variables) as *shared* information used by algorithms, whereas *environmental* variables are used for information specific to the node environment. According to our previous reflections, we can propose the following analogy: frontal variables correspond to mobile agent variables, nodal variables correspond to variables stored by the agent platform[1], and environmental variables can be associated to the MIB.

Let us examine a Wave program corresponding to the implementation of a breadth-first parallel spread. This type of spread creates asynchronously and in parallel a breadth-first spanning tree covering all the network's nodes of the network. The pseudo-code is expressed as follows:

```
Start in some node as a current node
Repeat from all current nodes:
  If the current node is not marked, mark it
  Otherwise, halt this branch
Hop through all links (broadcast) to neighbouring nodes
  (excluding the predecessor node from which the current
   node has been reached)
Every node reached becomes a current node
```

A possible equivalent code in Wave – iterative version - is the following:

```
DIRECT #C.
REPEAT (
  INDIVISIBLE (Node_mark == NONE. Node_mark = 1).
  TERMINAL = CONTENT.
       ANY ## ANY
)
```

Finding the simple shortest path tree in a network can be expressed in Wave as follows:

```
@#a.F=0.RP(N~,F<N.N=F.N1=P.$.F+L)
```

As it appears in our examples, the Wave language relies on a cryptic notation with a set of keywords executed through an interpreter. The navigation mechanism is based on spread-based navigation patterns allowing exhaustive exploration of the knowledge network with variable depth degrees.

Among the research activities around the *Wave* technology and mobile agents, we can mention a recent contribution in the field of multipoint-to-point routing with QoS guarantees using mobile agents [GLV01].

In this section, we have mainly focused on mobile processing techniques tailored for network management; most approaches propose conceptual frameworks and have been simulated in a simple network configuration. We now propose to concentrate on the next generation of network infrastructures, which provide active components and are thus particularly relevant to the use and the deployment of mobile agent based solutions.

1.4 Active Network Management

Recent advances in active network technology as well as the emergence of new mobile devices including an execution environment based on a micro *Java* virtual machine, for example, encourage us to investigate towards a realistic implementation of *Ecomobile* into the network environment. In this thesis, active network management refers to a general approach devoted to network management systems based on active nodes and mobile agent technology; active nodes, in this context, can be considered as a

[1] We will see later on that such variables can be stored in a *blackboard* (section 2.2.2).

network node capable of hosting and executing client code. An active node can be composed of a passive node connected to a local proxy which encompass the execution environment.

Active Networks (AN) and ad-hoc networks, which are introduced in the next sections, can help us to achieve this goal.

1.4.1 Active Networks

The basic idea of *Active Networks* is to move the service code to the network's nodes instead of leaving the service logic outside the transport network. AN transform ordinary passive network nodes into programmable nodes and are able to support a variety of service models, so that there is no specification regarding service definition beyond the recently defined AN architectural framework.

The concept of AN strongly refers to mobile code in a simple way; the program is considered as small and is deployed in active nodes via a *capsule* which is sometimes called *active packet*.

The AN architecture model [Cal99] is quite general and consists of a set of *active* nodes connected by a variety of network technologies. Each active node is composed of the *Node Operating System* (NodeOS), the *Execution Environment* (EE) and the *Active Application* (AA), as depicted on Figure 1-10.

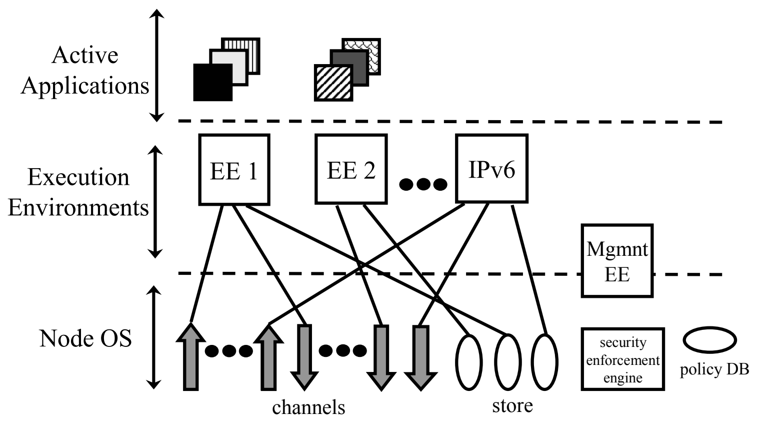


Figure 1-10. Active node infrastructure

Each EE exports a programming interface or virtual machine that can be programmed or controlled by packets directed towards it. Several EEs can be placed inside an active node. The *NodeOS* implements the core functionalities required to manage the resources provided by the network element; functionalities include transmission, computing and storage; in addition to that, each node has a *management* EE which supports control functions such as the maintenance of the node's security policy database, the support for the loading of new EEs for the updating and configuring of existing EEs, and the support for the instantiation of network management services arising from remote locations. Inter-EE communication is also supported via a standard loopback output channel.

Active Nodes therefore appear to be ideal candidates for the implementation of mobile agent based applications: they support the transfer of code via capsules, and ensure reliable communication channels as well as a secure access to the local database; they also support several execution environments, so that active applications can be divided into different categories according to their application domain.

The deployment of mobile agent systems into AN is still at an early stage and raises several issues [Kar00] such as security, performance, safety, garbage collection, platform independence, etc. In our context, a possible approach would consist in the implementation of an agent platform acting as an active application, so that agents have their own execution environment, an agent platform running in a specific EE; a specific communication channel between active nodes might be dedicated to the communication between the agents. Generally, this approach however implies dealing with small software components to reduce the local processing and to keep the memory in the active node at a reasonable size.

ANTS AND JANOS

ANTS and *Janos* are two *Java*-oriented operating systems for AN [THL01]. They both implement the NodeOS and EE layers of the active node model described above, and provide a resource-aware *Java* Virtual Machine called *JanosVM*. *Janos* is designed to prevent separate active applications from interfering with one another and to provide node administrators with strong control over active applications' resource usage. A critical challenge in the design of such an environment is to ensure that features are provided at the appropriate level and that there is no redundancy among the components. Furthermore, *Janos* must support the execution of untrusted code near the lowest level of packet receipt and dispatch; safety and security also constitute important issues in AN. Several similar OS are currently under development.

1.4.2 Ad-hoc Networks

Currently, ad-hoc networks are mostly based on *Wireless* LAN technology (WLAN) and rely on the IEEE standard (IEEE 802.11) in the *ad-hoc* mode. In this type of network, computers are brought together to form a network "on the fly". The network does not reveal any structure or any fixed points, and each node is usually able to communicate with every other node [ZP97]. The basic architecture is composed of a *Basic Service Set* (BSS) consisting of two or more wireless nodes, or stations (STAs), which are able to recognize one another and to establish communication. The set of stations belonging to a BSS communicate on a peer-to-peer level sharing a cell coverage area.

Consequently, the location of mobile devices in the network topology of ad-hoc networks changes continuously. Routing algorithms therefore face a new challenge in terms of optimization and implementation in the area of ad-hoc networks. In this context, the *MITAgent* (see Section 1.3.1) approach based on cooperative mobile agents seems very promising, provided the infrastructure required to execute mobile agents remains minimal and the agents themselves do not exceed a certain size.

1.4.3 The Terminodes

The *Terminode* project[1] is a long-term Swiss research project (2000-2010) which aims at studying and prototyping large-scale and self-organized mobile ad hoc networks. *Terminode* designates a mobile terminal which acts as a node and as a terminal at the same time. The project lays particular emphasis on the self-organization of a highly co-operative network of *terminodes* (mobile PC, walkie-talkies, PDA, mobile phones, etc.). As in ad-hoc networks, each *terminode* participates in a *virtual* network and is required to forward management information, such as geographical coordinates or routing information, to other nodes. In some cases, the *terminode* can act as a service provider for other *terminodes* [BBC$^+$01].

One of the major issues in the context of ad-hoc networks relates to the mobility management and the huge amount of information which has to be exchanged between *terminodes* for this purpose. A solution based on a *Virtual Home Region* (VHR) has been proposed [HLG$^+$01], so that a neighbourhood of

[1] http://www.terminodes.org

terminodes can register to a local VHR in order to retrieve information by means of a SNMP-like protocol; however, this approach leads to the introduction of a partial centralized management and could therefore result in scalability problems.

The business model emerging from a *terminode* network raises further difficulties: since the entire function set for resource control and service management is left to the responsibility of the *terminodes* themselves, the presence of an operator becomes superfluous, so that the realistic deployment of *terminodes* constitutes an important challenge from the business perspective.

Although the *terminodes* aim at building a virtual network in a highly self-organized and distributed environment, it appears that mobile agents have not been directly considered in this approach yet, to the best of our knowledge.

1.5 AGENT PLATFORMS

Agent platforms – or agent systems – provide intelligent and mobile agents with an execution environment, agent operations, security services and environmental facilities. Ideally, the agent platform should be able to deal with both *stationary* and *mobile* agents.

For several years, a considerable amount of agent platforms have been developed by academic and industrial organisations; most of these platforms resulted from research projects and therefore had a relatively short period of development. An excellent summary of agent platforms and corresponding features can be found in the *Agentlink* project[1].

There are basically two kinds of agent platforms: *mobile* agent platforms and *FIPA* agent platforms. Mobile agent platforms provide the agent with mobility services such as migration, localisation, cloning or place management, whereas FIPA agent platforms implement the FIPA agent reference model and related components. As FIPA does not support mobility, these agent platforms are not *per se* compatible with a mobile agent approach. A list of mobile agent platforms with comparisons between their respective performance levels can be found in [PCV99].

The implementation of the agent model is achieved though an intra-agent activity model which depends on the agent platform. For example, such a model can rely on a multi-thread mechanism making the agent able to process messages in a fully asynchronous way. Additionally, some specific methods can be invoked by the agent platform to inform the agent about external events. Such methods are called *callback* methods or simply *callbacks*.

Among the most complete and available *mobile* agent platforms, *Aglet*[2], an open source project of IBM, and *Grasshopper*[3], which was the first *Java*-based MASIF-compliant mobile agent platform, are worth mentioning. Both platforms fully support mobile agents through underlying *Java* mechanisms – serialization and transport over RMI-IIOP; they provide migration facilities and mechanisms operating in a full asynchronous message or event based communication model. Although the agent model does not refer to a specific architecture, the intra-agent activities make use of specific callbacks or *listeners* to control the agent behaviour (migration, cloning, location, etc.). Further development of *Aglet* and *Grasshopper* has been neglected, so that these platforms have not been adapted to the last release of FIPA specifications.

[1] http://www.agentlink.org
[2] http://aglets.sourceforge.net
[3] http://www.grasshopper.de

In this thesis, we aim at developing a mobile agent infrastructure which can be deployed in a FIPA-compliant environment with minimal requirements for mobility functions. In this perspective, we will try to consider only FIPA agent platforms in order to minimize the local resource consumption for agent facilities and to avoid interoperability issues between mobile and stationary agents.

FIPA-OS[1] and *Jade*[2] are two FIPA - but not MASIF - compliant agent platforms; they are frequently updated to support the last specification revisions and are currently widely used in the academic world as well as in the industry.

1.5.1 FIPA-OS

Originally developed by Nortel Networks, FIPA-OS was the first publicly open source project in agent technology. FIPA-OS is fully developed in *Java* and continuously improved over new releases. This agent platform is currently used in the scope of several collaborative research projects such as EU projects or industrial projects.

A FIPA-OS agent inherits from a root class which contains the underlying requirements for the inter-agent communication; the ACL message processing is achieved by means of dynamic invocation of *handle* callbacks. RMI is used for intra-platform communication and IIOP for inter-platform communication.

The intra-agent activity model relies on *tasks* based on a *conversation* and *task* model; a conversation defines a communication scheme between a pair of agents and consists of the collection of ACL messages, whose sequence is defined by the FIPA interaction protocols. A conversation manager can handle several tasks belonging to a single agent in a fully asynchronous way making an agent able to drive several conversations at the same time. Such an implementation makes FIPA-OS compliant with the FIPA reference model as far as the inter-agent asynchronous communication is concerned. However, the intra-agent activity model which is implemented by means of a multi-thread concurrency model complicates the development and the validation of co-operative approaches, so that the simulation of multi-agent systems intended to be deployed in a FIPA-OS platform is difficult to achieve (see Section 2.5).

Several add-ons have been developed around FIPA-OS such as a *spy* application, a remote agent starting application, a HTTP transport mechanism, etc. Moreover, a micro-FIPA-OS is being developed for PDAs and mobile device applications.

1.5.2 Jade

Jade is a freely downloadable *Java* agent platform and is fully compliant with the last revision of FIPA specifications. Unlike FIPA-OS, the intra-agent activity model defined in *Jade* is based upon a non-pre-emptive concurrency model. A *Jade* agent is implemented with a *Java* thread, which enables asynchronous inter-platform communication as specified by FIPA; it can implement one or several *behaviours*[3]: while intra-agent activities are synchronous, inter-agent communication relies on an asynchronous process. The behaviours are executed in a thread-per-agent concurrency model in which there is no stack to be saved; they are managed by an internal scheduler that implements a round-robin non-pre-emptive policy among all the behaviours available in the ready queue of an agent [BPR99]. The

[1] http://fipa-os.sourceforge.net
[2] http://jade.cselt.it
[3] In Jade, a process becomes a *behaviour.*

synchronous characteristic of cooperative processes makes *Jade* an attractive agent platform for the agent behaviour. More details about *Jade* behaviours are given in Section 2.1.1.

Jade uses the notion of *container* to locate the agents within the platform; the agents' mobility is supported between containers, but only within a single agent platform.

Jade furthermore integrates the *Java Expert System Shell* (JESS)[1], which is a rule engine and scripting environment written entirely in *Java* and originally inspired by the CLIPS[2] expert system shell. With JESS, the agent can *reason* on its internal knowledge, use declarative rules and make appropriate decision plans. Alarm correlation constitutes an interesting application of a rule-based language filtering alarms rapidly and making appropriate decisions.

As shown in Table 1-1, most platforms are open source, which brings considerable advantages in terms of code improvement and quality. The access to the code source also provides helpful facilities to examine in details the implementation which is a pre-requisite for allowing successful deployment in telecommunication networks [BDW01].

Agent Platform	**Mobility Functions**	**Intra-agent Activity Model**	**FIPA**	**MASIF**	**Source Available**
Aglet	Yes	Event-based Listeners	Partially	Partially	*Open Source*
Grasshopper	Yes	Callbacks IDL	Partially	Up-to-date	On demand
FIPA-OS	No	Conversation and task model	FIPA2000	No	*Open Source*
Jade	Intra-platform	Synchronous Behaviours	FIPA2000	No	*Open Source*

Table 1-1. Summary of agent platforms and properties

Since our system does not require a mobile agent platform, we propose to use *Jade* as agent platform and to design a FIPA agent acting as an *agency* and providing the agents with an execution environment and other mobility services. This architectural choice brings three major advantages; on the one hand, we avoid extra-overhead on the agent platform caused by the complexity of mobility functions; on the other hand, our FIPA agency fully benefits from FIPA ongoing activities; this agency is also able to communicate with other stationary agents.

Other development and research activities concerning agent platforms are the object of an European project called *Agentcities*[3]. This project aims at deploying an open world-wide network of FIPA-compliant agent platforms accessible through the public Internet. On top of these platforms, different agents are deployed and provide other agents with various services. The basic idea of *Agentcities* is to experiment agent technology and the composition of services with particular emphasis on the platform interoperability.

[1] http://herzberg.ca.sandia.gov/jess
[2] http://www.ghg.net/clips/CLIPS.html
[3] More information available at http://www.agentcities.org

1.6 Summary and Discussion

Nowadays, telecommunication networks are controlled by network management systems mainly based on TMN or SNMP. Both frameworks stem from the OSI model, rely on a client-server paradigm, and implement a manager-agent communication model. The agent, which controls a collection of managed objects describing every manageable resource in the network, from physical resources to client connections, interacts with a manager in which the entire service logic is executed.

The manager-agent model originally leads to the adoption of a centralized network management which unfortunately does not fit the inherent distributed environment of future transport networks in terms of scalability, flexibility and rapid time-to-market service deployment. The mobile agent technology allows the implementation of a decentralised management approach based on the principle of management by delegation. Considering mobile agents in telecommunication networks entails several advantages, among which the reduction of message processing between distributed entities limiting the bandwidth required for management purposes, the local processing enabling fast reaction to external changes, the design of distributed applications in which the management logic supposes the visit of several nodes, a significant improvement in terms of scalability and robustness, etc.

In our work, mobile agents are considered as a specific class of intelligent agents using advanced communication mechanisms and exhibiting properties such as autonomy, reactivity, pro-activity and social behaviour.

The mobility paradigm introduces new functions such as migration, cloning and various interaction schemes. Mobile and stationary agents are supposed to evolve in a common environment in which they interact in order to form a powerful multi-agent system. As they imply the transfer of code and data, mobile agents should be small enough to avoid an undesirable usage of bandwidth and they consequently match a reactive agent model rather than a deliberative model with reasoning mechanisms.

Different mobile agent based approaches for network management have been presented in this chapter: delegation agents (*deglets*) and *netlets* are two important classes of mobile agents belonging to a particular mobile code environment called MCE and implementing two kinds of agent behaviour; *deglets*, on the one hand, are used to perform a specific task in a network component which generally interacts with a resident OSI agent which is in charge of the manipulation of managed objects stored in the MIB and which can provide the mobile agent with necessary information; *netlets*, on the other hand, can form a population of mobile agents which continuously migrate within the network; they suit monitoring or topology discovery functions and can interact to exchange knowledge in order to improve their efficiency.

We have also presented *MITAgent* which is another approach purely based on direct communication between mobile agents travelling in the network to discover the topology and to build a network map. In this approach, mobile agents can exchange internal routing information in order to improve the system performance. *MITAgent* is particularly well suited to ad-hoc networks in which the network topology changes frequently because of the user's mobility.

In opposition to direct communication, an approach based on Swarm Intelligence resorts to a population of *netlets* in order to detect a problem in the network. In this behaviour scheme, mobile agents are considered as "small insects" which have the ability to deposit a chemical track in their environment giving information about the network state. The other members of the population then "read" the track and adapt their behaviour to the strength of chemical information.

Mobile multi-agent systems based on these agent behaviours have also been considered in various network technologies. Customer-profiled mobile agents have been developed to deploy new value-added services in Intelligent Networks, for example. In 3G networks (UMTS), mobile agents can manage the virtual home environment of mobile users in a proper way.

Finally we have laid particular emphasis on *Active Networks*, because their ability to transport code and to provide a rational execution environment qualifies them as an ideal environment for mobile agents.

In the telecommunication world, the deployment of agent platforms in heterogeneous environments strongly relies on standards. Although it has been stated that FIPA is the reference organisation for the elaboration of standards in agent technology, FIPA does not support agent mobility yet. In addition to FIPA, the OMG has released the MASIF specifications devoted to the interoperability of CORBA-based mobile agent platforms.

Mobile agents are executed in an environment provided by the agent platforms. A wide range of (mobile) agent platforms are presently available, but only a few are still being developed to implement the standardized architectures. *Aglet* from IBM and *Grasshopper* are two examples of mobile agent platforms freely downloadable from Internet; unfortunately, they are not upgraded to new standards and to the new *Java* development environment (JDK 1.3.x). *FIPA-OS* and *Jade* are two other popular agent platforms offering an excellent implementation of FIPA standards; but they do not support mobile agents. As we will show in the following chapters, a FIPA agent platform is sufficient to implement *Ecomobile*'s mobility requirements.

Despite the promising capabilities of mobile multi-agent systems, a lot of issues remain to be solved in order to achieve successful deployment of mobile agents in telecommunication networks. This problem is mainly due to the immaturity of the technology itself. For example, the different approaches presented in this chapter lead to different agent architectures and different agent systems, which makes their implementation in a standard-driven network infrastructure very difficult and leaves the specialists with a number of challenges in terms of scalability. The number of agents inside a population which has a strong impact on the system performance on the one hand and on the proliferation of mobile entities on the other hand, is also particularly difficult to control.

Since mobile agent technology is used in the scope of Internet applications, the intrusion of viruses constitutes a real threat: agents are made of code and data, so that a virus can be "inserted" in the population and activated in the network nodes, leading the entire network to misbehave or even collapse. However, security does not constitute a critical issue in the context of network management, as the agents themselves are supposed to evolve in an operator controlled environment and are not meant to be directly used by other Internet users. In *Ecomobile*, the infrastructure itself - for example active nodes in AN - and the agent platform deal with most security aspects by exploiting the underlying *Java* mechanisms.

The deployment of mobile agent systems also depends on their relying on standards such as FIPA or MASIF. Activities in that perspective are still emerging and need to be strengthened. In this context, the adoption of new business models by the network operators and the equipment vendors has to be clarified. Mobile agents inevitably lead to the dynamic installation of code in network components, so that the management and service logic is removed from the equipment vendors' full responsibility, although it can still be managed by the network operators themselves. In spite of numerous advantages provided by the installation of customized code in the devices from the point of view of the network operators, it is not certain that vendors are ready to accept such open environments.

Finally, we have observed that Active Networks, intelligent agents and mobile agents, which constitute three of the fundamental research areas characteristic of this thesis, gather three distinct researcher communities. Although the objectives pursued are sometimes similar, there is not, at the moment, any real attempt to establish collaborative research work towards the elaboration of standards, on the one hand, and the definition of a rational framework which could avoid the proliferation of protocols and data structures, on the other hand. In this thesis, we also aim at contributing to a unified view of the concepts of Active Networks, intelligent agents and mobile agents.

The next chapter of this thesis is devoted to the identification of three abstraction models for mobile agent systems, which will allow us to design a flexible infrastructure for active network management. We propose to consider three different approaches: *deglets* in MCE, *MITAgent* and *AntNet*, a similar powerful mobile agent approach based on Swarm Intelligence or *emergent behaviour*.

Chapter 2
Engineering Mobile Multi-agent Systems

In order to design and simulate an efficient mobile multi-agent system supporting intelligent tasks for network management, especially in the context of a highly dynamic and changing network environment, we will first have to identify its architectural components and to understand all the interactions taking place among them; in this perspective, the design of a scalable and flexible mobile multi-agent system leads us to apply a separation of concerns.

We believe that the success of future NMS will depend on their ability to *react* adequately to external changes in the environment, including modifications regarding the network environment (addition or removal of network devices in network topology, dynamic client connections, network failures, changes in the quality of services, etc.). The reactivity of mobile multi-agent systems appears optimal when most of the information managed by the mobile agents is issued from the environment defined by the agent system or by the network infrastructure which includes logical information, such as client connections or services description, and is stored in a local MIB, for example. In fact, we favour mobile MAS approaches promoting agent architectures in which the operational behaviour essentially requires environmental information.

In order to deal with the separation of concerns, it is necessary to elaborate a comprehensive methodology for the development of (mobile) MAS; the development of this methodology, however, is still at an early stage. Although ongoing research is oriented towards an *Agent-Oriented Software Engineering* (AOSE)[1] the mobility paradigm is not often considered in the design of intelligent agents: the agent working group[2] of OMG, for example, is currently working on an extension of the *Unified Modelling Language* (UML); it aims at supporting agent modelling and related interactions by introducing new formalisms. The result is the *Agent-UML* (AUML) language [OPB00]. In a similar perspective, the MESSAGE project [E907_01] focuses on the elaboration of a methodology centred on the agent-oriented realisation of telematic services and telecommunication applications.

The extension of the *Open Distributed Processing* (ODP) reference model [G851_96] defined by ITU-T with an adequate mapping of the MASIF concepts [MG01] constitutes an interesting approach towards the design of mobile MAS. For this purpose, the *Architecture Description Language* (ADL) has been introduced and defined as a UML profile called the MASIF-DESIGN. Recent work proposes attractive endeavours towards formalisms based on Petri nets [XD00]. CO-OPN/2 [HuB01], for example, is a formal component-oriented modelling language which can be used to model distributed applications involving mobile agent technology.

Although these formal methods tend to favour the design and testing of robust mobile agent applications, the complex behaviours of mobile MAS remain difficult to formalize. For this reason, we have chosen to emphasize a pragmatic approach based on experiments and using a reactive programming paradigm for a fine-grained control of asynchronous interactions inherent to the agent behaviours on the one hand and to provide the agents with an efficient cooperative environment on the other hand. This approach will be described in Chapter 4.

[1] http://www.jfipa.org/AgentOrientedSoftwareEngineering
[2] See http://www.objs.com/agent

In this chapter, we focus on the three fundamental abstractions governing any mobile MAS: the *task* - that is, the agent's specific goal, the *migration* – which refers to the agent's mobility, and the *interactions* – which designate the communication mechanisms between the mobile agents. We propose to define the three following abstraction models: the *computational* model (task design), the *navigation* model (migration design) and the *coordination* model (interaction design).

To illustrate the characteristics of each abstraction model, we will mainly refer to three different classes of mobile MAS architectures derived from the mobile agents based network management approaches which have been briefly introduced in Chapter 1: the *MCE deglet* approach, a mobile agent which must perform a specific task - the *AntNet*, an ant behaviour inspired system similar to a population of MCE *netlets* using Swarm Intelligence and communicating indirectly through their environment and the *MITAgent*, a population of cooperating agents exchanging routing information.

In this chapter, we shall also mention different design patterns proposed and supported by the *Aglet* mobile agent platform.

2.1 THE COMPUTATIONAL MODEL

We define the computational model as the part of the agent architecture that focuses on the mobile agent's operational behaviour, the agent task or *goal*. In the context of network management, typical tasks are configuration, monitoring, information retrieval from OSI agents, fault detection or routing tables updating. In order to achieve these objectives, the agents must be able to migrate, to communicate with other agents and with the environment, and to perform computation on internal data. The agents actually need to execute specific functions – or *methods* – to activate migration, to access the local environment and to contact other agents by using coordination mechanisms. The computational model therefore uses the methods provided by the agent system to determine how a specific task can be implemented.

The serialization mechanism, RMI, the facilities for transferring classes and the security framework presently make *Java* the ideal programming language on which the computational model can rely. Still, rule-based languages and rule engines can also be used to provide the computational model with a powerful reasoning framework which is particularly useful for alarm correlation for example[1].

Most mobile agent systems, such as *Aglet* or *Grasshopper*, provide agents with platform-specific methods such as *`moveTo(), clone()`*, *`sendMessage()`*. These methods can be called at any time during the agent execution. Other methods such as *`onActivation()`*, *`onArrival()`*, *`onDisposal()`*, etc. are automatically called by the agent platform; they are known as *callbacks*, and allow the agent to be informed of external changes, such as departure or arrival from/to somewhere. These methods generally cover all the aspects of the agent's behaviour.

The adoption of a design pattern [GHJ$^+$95] for mobile agents constitutes an interesting approach towards reducing the dependencies between the models during the mobile agents' design phase. Design patterns belong to a high-level abstraction of the problem description. Practical solutions based on design patterns require parameterization leading to a particular instantiation which makes behaviour changes during the runtime difficult.

We shall now introduce a few examples of task patterns characterizing different operational behaviours.

[1] Further information about reasoning systems and mobile agents at http://www.etcee.com/research/ki

2.1.1 Task Patterns

In the mobile agent's world, the *Master/Slave* design pattern, supported by *Aglet*, constitutes a fundamental task pattern [LO98] allowing a master mobile agent (MCE netlet-like for example) to delegate a task to a slave agent (MCE deglet-like for example).

Like other design patterns, the Master/Slave pattern resorts to the inheritance to abstract the behaviour-related methods (*`initializeTask() / doTask()`*). The slave agent moves to a destination host, performs the assigned task, and sends the task's result back. The master agent can delegate several slave agents in parallel, in which case the master agent is responsible for processing messages resulting from the slave agents.

A major drawback of the Master/Slave pattern is that the slave's behaviour is fixed at design time; because of the inheritance-based pattern, a mobile agent can not be transformed into a slave agent and can not easily be assigned new tasks to perform.

The *Plan* design pattern, which is an extension of the Master/Slave pattern, supports a workflow concept and organizes multiple tasks to be performed in sequence or in parallel by multiple agents; reusability of tasks, dynamic assignment of tasks to mobile agents and even task composition are promoted. The *Plan* pattern however requires a considerable amount of messages between slave and master agents and may lead to strong bandwidth consumption.

The *Jade* agent platform [BCT$^+$02] provides another approach towards a task design with generic behaviours. Based on message exchanges between agents[1], several behaviour schemes corresponding to various task types are defined and enable multiple interactions with other agents. The overall behaviours are depicted on Figure 2-1.

[1] Jade does not currently support physical mobility, since no standards for inter-platform mobility have been defined yet.

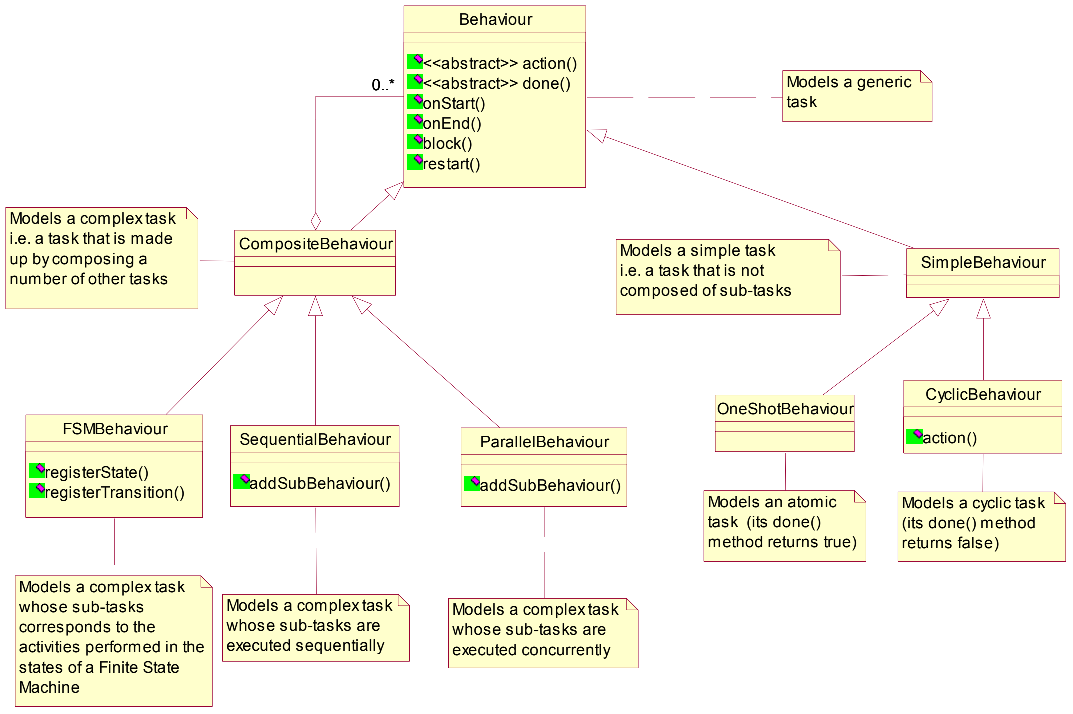

Behaviour
<<abstract>> action()
<<abstract>> done()
onStart()
onEnd()
block()
restart()
0..*
Models a generic task
Models a complex task i.e. a task that is made up by composing a number of other tasks
CompositeBehaviour
Models a simple task i.e. a task that is not composed of sub-tasks
SimpleBehaviour
FSMBehaviour
registerState()
registerTransition()
SequentialBehaviour
addSubBehaviour()
ParallelBehaviour
addSubBehaviour()
OneShotBehaviour
CyclicBehaviour
action()
Models an atomic task (its done() method returns true)
Models a cyclic task (its done() method returns false)
Models a complex task whose sub-tasks corresponds to the activities performed in the states of a Finite State Machine
Models a complex task whose sub-tasks are executed sequentially
Models a complex task whose sub-tasks are executed concurrently

Figure 2-1. UML Model of task behaviours in Jade

The behaviours are divided into two main categories: *simple* and *composite* behaviour. A simple behaviour consists in a task that is activated only once and cannot be blocked - *oneShotBehaviour* - or in a cyclically activated task. A composite behaviour is made up of several behaviours according to a parent-child relationship; it may consist of a sequential behaviour - *SequentialBehaviour* - which executes the sub-behaviours sequentially and terminates when all sub-behaviours have been executed. On the contrary, parallel behaviour - *ParallelBehaviour* - allows the developer to implement sub-behaviours which can be executed in a non-deterministic order. Finally, a behaviour can be described with a *finite state machine* (FSM); the parent behaviour controls the transitions between the FSM states and activates the behaviours corresponding to the current state.

2.1.2 Tightly and Loosely Coupled Task Model

According to the agent task model, dependencies between the operational behaviour on the one hand and the mobility and cooperation of the agent task on the other hand, appear at various levels, which makes the task more or less complex to design. In order to categorize the various task architectures with respect to these concerns, we propose to distinguish the *tightly coupled task* model from the *loosely coupled task* model: when the task model needs some direct dependencies with the other models so that the task itself has an influence on the agent migration or the agent-to-agent interactions, the task is tightly coupled with the navigation and coordination models. On the contrary, when the task model does not directly depend on the other models, the task is loosely coupled with the navigation and coordination models. The migration and the coordination scheme however can naturally have an impact on the operational behaviour. As the coordination mechanisms are controlled by the task itself (and the underlying inter-agent communication is managed by the agent system), the cooperation mechanisms between the agents often imply a tightly coupled task model.

According to this definition, we can assume that most location-aware tasks based on active migration are tightly coupled tasks which manage their migration decision internally, including the invocation of a *moveTo(destination)* method. Delegation agents such as MCE *deglets*, for example, implement their task by means of pre-computed paths; mobile agents implementing such a task model have to perform an operation at a specific node or at a collection of pre-defined nodes. It may however happen that the task does not need to migrate according to a pre-planned itinerary, but just navigates within the network topology with the assistance of an auto-discovery mechanism. As long as the navigation model supports an auto-discovery mechanism and fully controls the activation of the agent migration, the task remains loosely coupled and is simply activated at every node as a callback, independently from the location. An example of this behaviour appears in MCE *netlets*, a population of mobile agents living within the network and particularly well suited to monitoring functions.

As we will see in the next sections, the location of information necessary for migration and coordination plays a central role in loosely coupled task models.

2.2 THE COORDINATION MODEL

Coordination, which deals with the management of dependencies between activities, is a central problem in any dynamic system composed of interacting activities [Sch01]. Numerous research works have resulted in the definition of several coordination models and corresponding coordination languages, such as the *Encapsulation Coordination Model* (ECM) and its associated language STL++ [SCH99], for example, which mainly aims at separating coordination abstractions from the computation.

Coordination, which allows the mobile agents to cooperate and therefore to exchange knowledge, plays a central role in the scope of our research. Examples of mobile MAS exploiting coordination between agents are *MITAgent* (meeting-oriented coordination) and MCE *netlets* with Swarm Intelligence (indirect coordination) for example.

Although the interaction logic is generally embedded inside the mobile agent, code mobility introduces additional complexity in the coordination scheme, but indirect coordination provides a new interesting way to manage interactions between mobile agents.

In this section, we also introduce the software components required for the implementation of efficient coordination mechanisms.

2.2.1 Interaction Patterns

In order to address coordination problems in mobile agent applications, we propose to begin by considering general interaction schemes likely to intervene in these applications and which could provide an efficient way of decoupling the agent task from the coordination model. Although they rely on coordination mechanisms, interaction patterns are not limited to the strict context of coordination models: they propose cooperation schemes which can be directly mapped to a specific problem. Interaction patterns consequently tend to promote a decoupling between the coordination model and the computational model and therefore favour loosely coupled task models.

The following sample design patterns are supported by *Aglet*. The *Meeting* pattern provides a way for two or more mobile agents to initiate local interaction at a given host. The *Locker* pattern defines a private storage space for data left by an aglet before it is temporarily dispatched to another destination. The *Messenger* pattern defines a surrogate mobile agent carrying a remote message from one agent to another.

The *Finder* pattern defines a mobile agent that provides services for naming and locating agents with specific capabilities. The *Organized Group* pattern composes mobile agents into groups whose members all travel together.

2.2.2 Blackboard

The blackboard is an important environmental component used as a shared information space in which mobile agents can read or write information. In addition to mobile agents themselves, other agents or external users may access the blackboard through a GUI interface in order to write messages informing a specific agent or to influence the general behaviour of the whole agent population, so that the blackboard can be used for indirect interactions – or *indirect* coordination - between mobile agents.

According to this description, the blackboard is essentially a *passive* entity. Still, it can be extended with a *reactive blackboard* model enabling the implementation of programmable reactions activated in response to the agents-blackboard interactions [CLZ97]. Reactive blackboards entail several advantages: they can be used to implement specific local policies for the interactions between the agents and the node environment, to achieve better control and to prevent the local node from the intrusion of malicious agents; it can also be used to implement specific resources management policies to reach better efficiency in agent execution. An example of mobile agent environment experimenting reactive blackboards is MARS [CabLZ00].

Basically, the location of the blackboard is not subject to any specific limitation; however, in order to preserve communication resources and to improve scalability, an architectural choice may consist in associating each hosting environment with a single blackboard so that the mobile agent has access to the local blackboard only.

2.2.3 Mobility Oriented Coordination

The coordination models for mobile applications can be divided into four main categories: *direct* coordination, *meeting-oriented* coordination, *blackboard-based* coordination and *Linda-like* coordination [CaLZ00]. In addition to these categories, a taxonomy based on the degrees of spatial and temporal coupling induced by the coordination models is introduced: *spatially coupled* coordination models require the interacting entities to share a common name space; on the contrary, spatially uncoupled models enforce anonymous interactions. *Temporally coupled* coordination models imply synchronization of the entities involved in the communication, temporally uncoupled coordination models enforce asynchronous interactions.

DIRECT COORDINATION

In a direct coordination model, agent-to-agent interactions require the partners involved in the communication to be named explicitly and generally rely on a client-server communication model using *Java* RMI or CORBA, for example. This is the reason why direct coordination models imply both spatial and temporal coupling.

This approach allows a mobile agent to have full control of its environment constituted of distributed components: communications are issued directly and their semantics is not influenced by external entities such as the hosting execution environments. Moreover, and according to the direct coordination models, each agent has the capacity to link services dynamically to a server localized in a single execution environment, thus providing the flexibility and adaptability required in a heterogeneous and dynamic environment.

Direct coordination models however present several drawbacks as regards agent-to-agent interactions: the localization of agents, for example, may introduce complex routing schemes and location management may lead to residual information being left onto the nodes. Furthermore, in this kind of model, a high number of interactions require a stable network connection which makes communication highly dependent on network reliability and nullifies the advantages of using mobile agents [CabrLZ00].

In the context of *Ecomobile*, a direct coordination model is proposed for agent-to-environment interactions only.

MEETING-ORIENTED COORDINATION

Meeting-oriented coordination models define spatially uncoupled models. Mobile agents can communicate at local meeting points, as long as they reside in the same node. The synchronization between the participants may be realized by means of a *rendezvous* mechanism.

The agents are not required to name the interacting entities; however, they need to be informed of a common meeting name, in which case they can not maintain the anonymity of full spatial uncoupling. Since the schedule and the position of agents can generally not be predicted, these models also imply a high risk of missed interactions.

A typical application of meeting-oriented coordination appears in the *MITAgent* approach, for example: mobile agents continuously travel within the network to discover the connectivity and exchange their knowledge during occasional meetings.

As we will see in Chapter 3, *Ecomobile* realizes full spatially uncoupled meeting-oriented coordination for agent-to-agent interactions.

BLACKBOARD-BASED COORDINATION

In blackboard-based coordination models, agents interact via a blackboard. In this context, there is no restriction concerning the blackboard location. Messages can be left on blackboards, no matter where the corresponding receivers are or when they read the messages. Interactions are consequently fully temporally uncoupled, but remain spatially coupled, since agents must agree on a common message identifier to communicate and exchange data. According to this definition, blackboard-based coordination does not support anonymous messages. Apart from a common message identification between the emitter and the receiver, no other restriction is imposed regarding the message recipient, which may be an agent or a group of agents.

Agent-to-environment interactions can be facilitated by the use of a blackboard storing environment-specific information; for example, a stationary agent can be queried to prepare or possibly to pre-process information retrieved from the local environment and to place them in the blackboard, so that mobile agents can avoid time-consuming interactions with the stationary agent.

LINDA-LIKE AND FULLY INDIRECT COORDINATION

The inter-agent coordination model based on *Linda*[1] requires a particular shared data space called *tuple space*, in which the agents store and retrieve information by means of associative mechanisms. The information is described with *tuples*, a data structure adapted to pattern-matching algorithms. *Linda* defines several primitives to store and to retrieve information from the tuple space.

[1] http://www.cs.yale.edu/Linda/linda.html

Linda-like coordination models enforce full uncoupling requiring neither temporal nor spatial agreement. Unlike blackboard-based coordination, these models support anonymous messages. This kind of model suits large-scale mobile agent applications in which the hosting environment is sometimes difficult to manage, as in Internet applications, for example. However, Linda-like coordination increases the communication overhead in proportion to the number of interactions with the tuple space; access requests may imply several message exchanges between agent and tuple space.

The taxonomy of coordination models with their spatial and temporal coupling is summarized in Table 2-1. We have also highlighted the categories to which *Ecomobile* can be associated.

		Temporal	
		Coupled	*Uncoupled*
Spatial	*Coupled*	Direct, ***Ecomobile* (agent-to-environment)**	Blackboard-based
	Uncoupled	Meeting-oriented, ***Ecomobile* (agent-to-agent)**	Linda-like, fully indirect

Table 2-1. Taxonomy of coordination models with their spatial/temporal coupling

In order to deal with a mobility paradigm based on a pure local processing approach, we propose to extend the *Linda*-like coordination model to a fully indirect coordination model. In addition to spatial and temporal uncoupling, a blackboard is defined locally in each hosting environment and allows mobile agents to deposit and to retrieve information in a complete anonymous way that is, without any message identifier, agent identifier, blackboard or tuple space identifier. This approach focuses on the agent-to-environment interactions within the hosting environment (agent system) and avoids unnecessary overload of message-passing as well as requests between agents or a shared information space. The inter-node communication channel needed for mobile MAS purposes can be consequently used for agent migration only.

Fully indirect coordination favours a strong usage of the mobility paradigm and reduces the client-server communication to the hosting environment. Apart from migration functions, the agent task only needs methods to access the environment and tends to design loosely coupled tasks as far as the computational model is concerned. Still, fully indirect coordination only suits a particular kind of mobile MAS based on emergent behaviour (see Section 2.4).

2.3 THE NAVIGATION MODEL

The navigation model can be associated with the mobile agents' migration strategy, which most of the time depends on the application: the agent task requires a specific migration scheme and establishes dependencies between the navigation model and the computational model.

This dependency scheme implies severe limitations on the deployment of large-scale mobile agent systems. Implementing two network management functions such as monitoring and configuring, for example, leads distinct families of mobile agents to require different migration strategies. In the end, the mobile MAS exhibits several families of mobile agents, each with a specific *mobile behaviour*, resulting in a totally unorganized population of mobile agents, at least from the point of view of a macro-society, this proliferation of agents steadily causing considerable degradation of the network performance.

In *Ecomobile*, we deal with the issues related to multiple navigation models by proposing a reduction of dependencies between the agents' mobile behaviour and their tasks. We are now proceeding to introduce the location concept and the related mobility components; migration patterns will then be shortly analysed, before we examine two fundamentally different navigation models, namely the pre-planned and the stochastic navigation model.

2.3.1 Location Concept, Migration and Itinerary

Mobile agents migrate from one location to another. The location is defined as a static *place* and is considered as part of the hosting environment managed by the agent system. One or several places can co-exist in the same environment. According to the application, they can be statically created by the agent system or dynamically created by the mobile agents themselves when necessary; connection or customer profile related places may have an existence with different durations. Each place is identified by a unique address and may be combined with a node address, or a URL in case of Internet applications.

In *Grasshopper*, a place can be associated with specific functions which mobile agents resort to. In *Jade*, the place is considered as a simple agent container. Places can be defined by a semantics which generally depends on the application itself; they can represent different authorities or business entities, they can limit the agent-to-environment interactions in an environmental scope defining a place for each MIB, for example, or they can represent a virtual network of nodes mapping a group of places within each node; in the latter case, places are interconnected according to a virtual (or logical) topology that may reflect a physical topology. Figure 2-2 shows an example of interconnected places.

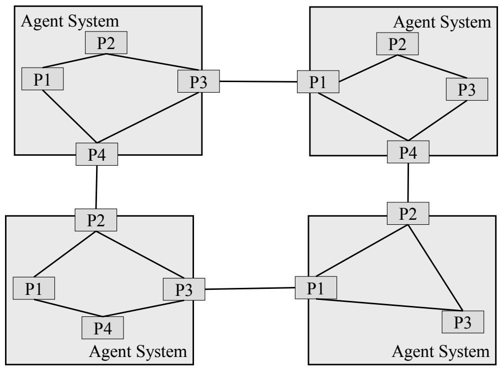


Figure 2-2. Places and location concept

When the migration is performed between places belonging to the same agent system, it can be associated with *virtual* or *logical* mobility but, when places belong to different agent systems, the migration represents a form of *physical* mobility [RSP00].

From the point of view of mobile code, two kinds of migrations are possible, i.e. *weak* and *strong* migration [CLZ00].

WEAK MIGRATION

The transfer of mobile agents generally uses weak migration and involves preserving the code and data state during migration. Usually, a migration method - *moveTo(destination)* for example - is invoked by the agent, and provokes the agent to be stopped, serialized, transferred and resumed in the destination node. The resuming operation requires to re-start the code or to invoke a specific callback method of the agent – `resume()`, for example. The invocation of this method is made by the agent system which is responsible for the code transfer.

STRONG MIGRATION

Strong migration appears to be a new concept fitting certain kind of network-unaware applications, such as load-balancing applications. When a mobile agent has to be moved to another execution environment during its execution on behalf of the agent system because its execution is not appropriate on the running environment any more, the agent code must be transferred with its execution state (program counter) so that the agent can be resumed exactly where it was suspended: strong migration allows this transfer. This technique is much more complicated than weak migration because it requires to extract and restore the computational state (in particular, the call stack and the program counter) whenever the migration request occurs anywhere in the code.

However, most network management tasks are performed locally and do not require strong migration, as the execution state can be preserved by using internal variables; the invocation of the `resume()` agent method will then perform the appropriate action based on the state variables. Weak migration is easily implemented with *Java* whereas strong migration requires adaptations of the *Java* virtual machine.

The agent system deals with the mobile agents' migration. Any migration request, however, can be initiated by the agent itself (*active* migration) or by the agent system (*passive* migration). Active migration is used in most mobile agent based applications, in which agents are autonomous entities initiating their own migration. In principle, passive migration is used when the agent system itself decides to move an agent without its knowing because the execution environment is not convenient any longer, for example, or because the agent is executing functions which are not appropriate to the local environment. When passive migration is synchronous, weak migration fulfils the agent transfer, whereas asynchronous passive migration calls for strong migration.

In the most simple case, the agent destination requires a single hop migration but, when the destination is not directly connected to the current place, it requires a multi-hop migration. In case of a multi-hop migration, the agent system may use routing algorithms to determine the best trip.

The *itinerary*, which is another component of the mobility paradigm, is simply a list of places that the agent can store as part of its internal knowledge; it can be a pre-computed path that the mobile agent will follow during its migration. A history of visited locations can be built by the agent in order to constitute a future itinerary back to the initial location (*boomerang* effect).

2.3.2 Migration Patterns

Migration design patterns are proposed to implement the mobile agents' migration strategy; they authorize a decoupling between the agent task and the navigation model. Both of the following patterns are supported by *Aglet*.

The *Itinerary* pattern provides the mobile agent with a travel plan: the *Itinerary* object is initialized with a list of destinations to be visited sequentially and a reference to the mobile agents; the itinerary can thus be shared by several agents. The *Itinerary* object must be completed before the agents start their travel.

The *Forwarding* pattern is a particular object residing in the agent system, which forwards newly arrived mobile agents automatically to another host.

2.3.3 Pre-planned Navigation

A mobile agent's itinerary can be pre-computed by the agent itself or by an external entity (agent or agent system). If the itinerary remains stable during the travel, the agent reads the list of locations sequentially and performs specific actions. The mobility of the agent therefore follows a *deterministic* navigation model. The itinerary may however be pro-actively refined during the agent trip according to a migration planning process located in the places visited by the agent. To illustrate this approach, we propose an adapted version of the shopping scenario [CCM+98]: suppose a user is looking for a specific available resource known to be offered by n+1 nodes. The probability p_i that node *i* has the resource available is known and independent for different nodes; still, it takes a given time t_i to perform queries within the node in order to determine whether the resource is available or not; going from node *i* to node *j* requires travel time l_{ij}. Given that information, and considering that the mobile agent starts and ends at node 0, what is the minimal expected time to find the item or to conclude that it is not available?

In order to deal with the agent migration-planning problem which is described in this scenario, mobile agents can interact with a local stationary agent to estimate the probabilities of success based on local knowledge: the node agent activates a decision-making module and returns a modified itinerary to the mobile agent. In this case, the agent's mobility follows a *non-deterministic* navigation model, which is also considered when the agent system has to compute a multi-hop routing scheme because the destination requested by the mobile agent is not immediately communicated to the agent system.

Both deterministic and non-deterministic pre-planned navigation models are strongly associated to the agent task, so that these navigation models are not compatible with loosely coupled tasks.

2.3.4 Stochastic Navigation

Stochastic navigation essentially implies a non-deterministic behaviour from the mobility point of view: mobile agents read information from the blackboard or query a stationary agent in order to retrieve local attributes; then, they perform the required action and update the local data. Finally, when the agent is ready to migrate, the environmental information may be sufficient to compute its destination without taking the operational behaviour into account. This model enables mobile agents to be more reactive to environmental changes.

This approach appears particularly interesting in the context of the computational model: since the navigation model purely depends on information managed by the hosting environment, and not by the mobile agent itself, the computational model supports a loosely coupled task model. The navigation model actually has full control on the migration strategy as well as on the migration activation: although the navigation model resorts to active migration, the mobile agent supports a synchronous passive migration as far as the task is concerned. Stochastic navigation models are therefore particularly fitted to support the implementation of mobile MAS which must reflect the current state of the network, such as monitoring agents systems (MCE *netlets*), discovery agents systems and all the approaches based on the *emergent behaviour* presented in the next section.

In spite of their highly reactive behaviour, stochastic navigation models are not adapted to delegation agents or destination-based agents. The probabilistic selection of destinations actually makes the mobile agent's trajectory difficult to predict and might drive the agent through a considerable amount of node visits before it reaches its final destination.

2.4 EMERGENT BEHAVIOUR

In this section, we deal with an innovative engineering approach relying on collective phenomena to implement mobile MAS and to achieve a global task by virtue of emergence and self-organization.

In Section 1.3.1, we have briefly described a mobile MAS based on the notion of *Swarm Intelligence*. In systems endowed with the property of Swarm Intelligence [BW89], unintelligent agents with limited individual capabilities collectively exhibit intelligent behaviour. This bio-inspired approach draws its model from evolutionary biology and from the study of ecosystems such as bacteria, ants or bees societies: it turns out that these societies exhibit social coherence although the individuals' behaviour is mainly stochastic. This kind of behaviour is known as *emergent behaviour*. Still, it has to be mentioned that, since the observation of the system is influenced by an important subjective component related to the observer's cognitive properties, the term of emergence remains controversial [CLC99].

In this area, significant research efforts have focused on ants societies. Ants are relatively simple insects that have a very limited amount of memory, are almost blind, and apparently exhibit random behaviour. A colony of ants is however able to perform complex tasks such as forming bridges, building and protecting their nest, regulating nest temperature, searching areas for food, finding the shortest routes to food and exploiting the richest available food source [SHMar99].

One of the questions raised by biologists was how ants can manage to establish the shortest paths from their colony to feeding sources and backwards. Observations have shown that the medium used to communicate information among ants and to influence their trajectory, consisted of special chemical trails called *pheromone*: ants mark the path with pheromone laid on the ground in varying quantities. When isolated ants encounter this pheromone trail, they follow it and thus reinforce it with their own pheromone [DMC96].

In the context of mobile MAS, mobile agents "mimic" the ants' behaviour. Ants and agents exhibit common characteristics [WhP99]: although no single agent has a global view of the world, a population of single agents evolving in the same environment give evidence of emergent behaviour leading to the fulfilment of the task. Moreover, direct agent-to-agent interaction is replaced by a communication paradigm relying on simple time-dependent environmental signals resembling *pheromone* trails in an ants colony; signals usually decay in time. Finally, the strength of chemical trails provides the driving force for migration patterns.

This communication model is also called *stigmergy*. Stigmergy can be defined as a form of indirect communication taking place among individuals by means of modifications induced in their environment. Two kinds of stigmergy have been observed: *sematectonic* stigmergy involves a change in the physical characteristics of the environment; this change can be compared to the modification of an ant nest each time an ant brings its ball of mud on top of it. According to *sign-based* stigmergy, on the contrary, the chemical substance deposited in the environment makes no direct contribution to the ultimate goal, but is used to influence the individuals' subsequent behaviour in order to reach the goal.

Since no single agent is entrusted with any critical mission, mobile MAS based on emergent behaviour can be considered as robust systems: if an agent should disappear from the population, the rest of the population would be able to pursue its work. As far as the abstraction models are concerned, mobile MAS based on stigmergetic communication rely on fully indirect coordination in a stochastic navigation model.

In the next sections, we describe two approaches resorting to emergent behaviour for distributed network management purposes: *AntNet* and *SynthECA*, which are particularly well-adapted to the development of a self-organized and self-tuned mobile agents population like *Ecomobile*.

2.4.1 AntNet

AntNet, which has been the object of numerous research works, is a mobile MAS aiming at the adaptive set-up of routing tables in communications networks [CD98]. It has to be noted that this approach refers to a more general optimisation technique called *Ant Colony Optimisation* (ACO)[1].

Routing in packet-switched networks (such as IP networks) is subject to various problems: the information, i.e. the routing tables, and the decision system are completely distributed; the session arrival and data generation process is generally non-stationary and stochastic; several conflicting performance measures are usually taken into account for the quality of service, such as throughput and average packet delay; finally, the constraints imposed by the underlying network technology are difficult to implement. A mobile MAS approach seems to be particularly adequate to deal with all these routing problems inherent to packet-switched networks.

AntNet resorts to ant-like mobile agents travelling in the network, using the same medium as the data traffic, and implementing distributed stigmergetic control. When they visit nodes, they "read" the environment and make a decision concerning their next destination; the environment can be associated to the ant's pheromone trail. Before leaving the node, the agent modifies the environment, and thus reinforces the decision, contributing to the migration decision of future visiting agents. Figure 2-3 shows the principle of the ants' navigation model. The ants decisions are based on link costs: the pheromone strength is actually influenced by two factors, the number of ants which travel on a link and the quantity of pheromone deposited by the ant. In this example, path length is given by the link costs: the shorter the path the stronger the pheromone trails.

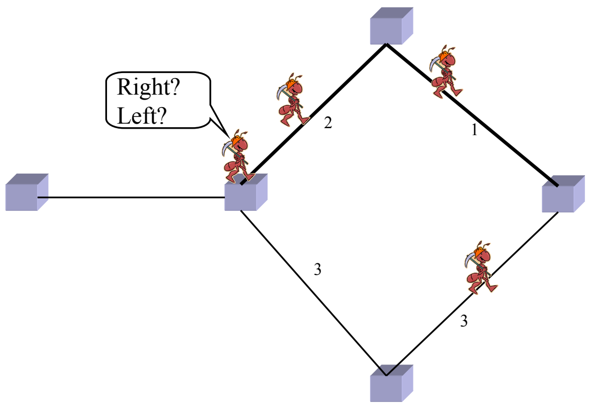


Figure 2-3. Ants making decisions based on the strength of the pheromone trail

We now propose to examine the different steps of a simplified version of the algorithm in order to identify the main characteristics related to the abstraction models.

AntNet defines two kinds of mobile agents: *forward* and *backward* ants. Each node includes a routing table T_k which contains probabilistic entries; T_k defines the probabilistic routing policy currently adopted at node k: for each possible destination d and for each neighbour node n, T_k stores a probability value P_{nd} expressing the benefit of choosing n as the next node when the destination node is d under the current network-wide routing policy.

[1] http://iridia.ulb.ac.be/~mdorigo/ACO/ACO.html

At regular intervals, a forward ant is launched from each network node towards a destination node to discover a feasible, low-cost path to that node and to investigate the load status of the network. Forward ants share their queues with data packets and experience the same traffic loads. Destinations are selected locally according to the data traffic patterns generated by the local workload. While they are travelling toward their destination nodes, the agents memorize their paths as well as the traffic conditions; they also store as internal knowledge the itinerary corresponding to the visited nodes and the time elapsed since the launching time.

At each node, every forward ant selects a node among all neighbouring nodes according to the probability computed as the normalized sum of the probabilistic entry in the routing table. When the destination node has been reached, the forward ant creates a backward ant, to which its memory is transferred, and dies. The backward ant follows the path discovered by the forward ant in the opposite direction. It is assumed that backward ants do not share the same queue as data traffic but use higher priority queues since the routing tables must be quickly updated. The backward ant updates the routing table of each visited node.

DISCUSSION

Further reflection on *AntNet* reveals that this model relies on stigmergetic communication in a fully indirect coordination model, since no agent-to-agent interaction is required and since an agent can interact only with the local blackboard associated to the agent place, in which the ants deposit and "read" the pheromone.

AntNet is ruled by a model combining stochastic and deterministic pre-planned navigation. *Forward* ants mainly use environmental information to decide on their next destination, so that the agents' objective is to follow a path until they have reached their final destination, the path being determined by data traffic conditions on the one hand and by probabilistic values computed from the pheromone quantity on the other hand. Although the migration function of *forward* ants relies only on traffic conditions and network availability at the beginning of the process, – so that the task could be separated from the migration function and thus become a loosely coupled task – the migration decision will however, after a certain time, depend on the pheromonal information appearing in each node. Since the migration decision is based upon predictive functions and strongly depends on the nature of the task to be performed, the computational model of *forwards* ants is actually based on tightly coupled task and *forwards* ants rely on a stochastic navigation model.

On the contrary, *backward* ants need to access their internal knowledge, namely the itinerary, in order to perform the migration from the destination to the source in a deterministic way (*boomerang* effect). *Backward* ants therefore rely on a pre-planned navigation model.

2.4.2 SynthECA

The *SynthECA* approach [Whi00] is based on sign-based stigmergy and can be considered as a generalization of *AntNet*. *SynthECA* is not only devoted to a routing process, but is also useful for fault location and planning. The chemical messages used for indirect communication between mobile agents have two attributes: a label and a concentration [WhP99]. The different possible labels lead to a classification of distinct chemical tracks according to the type of agents that must sense the signal; the three types of signals are: a routing chemical (*r-chemical*), a reliability chemical (*R-chemical*) and a quality of service chemical (*qos-chemical*).

Mobile agents sensing the r-chemical discover the connection path according to a logic which is similar to the logic revealed in *AntNet*. When a path has emerged, an *allocator* agent traverses the path and assigns resource to the connection; a quality-of-service (QoS) sensing agent then monitors the end-to-end path allocated to the connection and adapts the chemical intensity in accordance with the path quality. If a problem occurs along a specific connection at a specific node, the quality of service will be significantly deteriorated and *qos-agents* will drop *qos-chemical* so that the node responsible for the problem can be identified. It is assumed that the network resources are shared among several connections which participate in the reinforcement of the *qos-chemical* trails, provided the connection is affected with a quality reduction caused by the same problem. A *qos-location-agent* which senses the *qos-chemical* constantly migrates towards higher concentrations of *qos-chemical*. When the defect device has been identified, a *R-chemical* is dropped at its location in order to influence the routing and planning agents which will be in charge of diagnosing and solving the problem by re-configuring the fault device with an alternative connection path. *R-chemicals* can be used to drive the planning process along with an additional chemical resulting from network congestion (c-chemical). The *R-agent* is a planning agent supervising a reliability threshold for each network device and making adequate decisions regarding potential problems.

Mobile agents in *SynthECA* have a common architecture: they include emitters, receptors, chemistry, a migration decision function and a memory. Emitters and receptors allow the agent to sense and to change the chemical in the environment; the chemistry defines a set of chemical reactions which can be performed by the agent by means of internal information stored in a memory on the one hand, and of the sensed chemical on the other hand. The migration decision function, which implements the agent's migration strategy, allows the agent to hill climb in the direction of increasing concentrations of chemicals sensed either probabilistically or deterministically.

DISCUSSION

In *SynthECA* as in *AntNet*, there is no direct communication between mobile agents; the behaviour of the mobile MAS strictly depends on the environment characterized by the network parameters and by the chemical messages controlled by the agents themselves and reflecting the network state as regards the implemented functions; routing, for example, is concerned with availability and QoS, fault detection deals with operational state of devices, etc. Depending on the process, the migration function implements stochastic navigation with routing agents or pre-planned navigation, in which the description of the current connection is required, since *qos-agents* monitor the connections.

2.5 OUR CLASSIFICATION OF MOBILE MAS

According to the definition of abstraction models introduced in this chapter, we propose in Table 2-2 a classification of the mobile MAS which have been presented so far. It has to be noted that system architectures have been considered in the primary architecture and in the functionalities for which they have been originally designed. Most of them are perfectly extendable to support other functionalities, implying additional dependencies between abstraction models.

	Computational Model	**Coordination Model**	**Navigation Model**
MCE netlets	Loosely coupled task	Direct	Stochastic
MCE deglets	Tightly coupled task	No coordination	Pre-planned
AntNet	Tightly coupled task	Fully indirect	Stochastic/ Pre-planned
MITAgent	Loosely coupled task	Meeting-oriented	Stochastic
SynthECA	Tightly coupled task	Fully indirect	Stochastic

Table 2-2. Abstraction models of different mobile MAS

The classification of MAS with respect to their computational model is based on the following assumptions: *netlets* are able to interact with one another independently from their location, as long they can be identified, so that they implement a direct coordination model. While *netlets* travel along the links according to the network topology, and the tasks implemented in *netlets* generally look like monitoring functions and do not influence the migration, *deglets* are considered as delegation agents for general purposes and, in spite of the possible introduction of design patterns, implement tightly coupled tasks.

AntNet entails two kinds of agents: *forward* ants implement a stochastic navigation model, whereas *backward* ants, which are closer to delegation agents, follow a deterministic pre-planned navigation model. *AntNet* and *SynthECA* implement a fully indirect coordination model, using stigmergetic coordination and describing emergent behaviour. Since the migration decision is based on predictive functions depending on the agent objective, these two approaches however do not support a coupled task model.

In the *MITAgent* approach, cooperating agents exchange routing information according to a meeting-oriented coordination model; the agent task can be limited to knowledge transfer and internal knowledge updating; finally, the migration decision relies on local network information only and enables simple auto-discovery mechanisms. *MITAgent* consequently follows a loosely coupled task model.

Whereas direct coordination and meeting-oriented coordination enable agent-to-agent interactions, a (fully) indirect coordination model can also be considered in this context. Since there is no direct communication between the agents, knowledge exchange between agents however remains impossible unless it is transferred via the environment. This mechanism therefore requires the agents to be identified so that the information can be addressed to the agent requiring it.

2.6 SIMULATING MOBILE MAS

The implementation of abstraction models dedicated to mobile MAS for network management obviously requires an important effort with respect to simulation. Simulation plays a central role during the elaboration of mobile MAS and is motivated by the necessity to analyse the behaviour of mobile agents in response to particular stimulations. Simulation results are required to validate the architectural design, to make a performance evaluation compared to a reference model, and to receive a feedback from the MAS in order to detect pathological behaviours. The simulation results finally lead the system to be improved: the architecture can be adapted or the agent parameters can be tuned.

The simulation of a mobile MAS, which is composed of several autonomous mobile entities running in parallel and performing agent-to-agent and agent-to-environment interactions, remains a complicated task. Moreover, a behavioural simulation should not be affected by external parameters, such as CPU power, thread scheduling policy or memory. Different approaches have been proposed to meet this challenge [UK01]: JAMES [UTT00] - a *Java-based Agent Modelling Environment for Simulation* - is a simulation framework for multi-agent systems which provides the means to describe variable structure models and their distributed parallel execution. While the model design consists in a hierarchical compositional construction of models which can be either *atomic* or *coupled*, the coupled model makes the interaction between components possible. Although JAMES was originally designed for supporting deliberative agents, recent work has shown a possibility to use the JAMES formalism to simulate mobile agents by mapping the location concept onto coupled models [UKL02]; although the agent code does not move, the reconfiguration of the different models leads to migration simulation. A complete study of multi-agent systems simulation is beyond the scope of this work; however, we have to focus on different aspects which are obviously relevant to the simulation and deployment of mobile agent based network management system. The simulation framework - or *simulator* - must take into account the target environment of the mobile MAS, including the agent system's (agent platform's) components and relevant services.

Without going into details, three main scenarios are possible: in the first case, "everything" is simulated. This option is particularly useful during the first steps of the agent behaviour design; a minimum set of functions in the agent system is simulated. In the second case, the simulator does not simulate the agent system but interacts with a single instance of an agent platform running in the same environment as the simulator; in this case, physical mobility is replaced by logical mobility which is "simulated" within the same agent platform. Finally, the simulator can deal with several agent platforms which are physically distributed over a network; a similar approach is supported by JAMES, for example; according to this option, coordinators are necessary to keep the distributed simulations synchronized with the main simulator.

In order to capture the agent behaviour in response to the environmental changes, the network conditions also have to be taken into account and simulated[1]. Whereas IP network simulation is rather simple, optical networks or UMTS networks, for example, are much more difficult to simulate because of the complexity of the network itself in terms of capabilities, dynamic components and physical constraints. In order to simulate the network infrastructure, we have developed the *Generic Network Management Tool* (GNMT) (see Appendix A). More details about the simulation environment are given in Chapter 5.

Like the agent system, the network environment can be fully simulated on a central machine or distributed over several machines. A network environment combining a part of real network devices and a part of simulated devices may also lead to cost effective validation of the mobile MAS since, depending on the network technology which has to be simulated, hardware devices can be expensive or not available yet.

The transition from a simulated environment to a real implementation is a critical stage in the elaboration of mobile MAS. Although a mobile MAS can produce interesting results from the point of view of the simulation, it however remains uncertain whether the implementation of the MAS in real network conditions using agent platforms should lead to the same conclusions. The integration of mobile agents in a real agent system may also raise various architectural issues influencing their performance.

Our work is focused on a functional validation of the mobile MAS behaviour: real-time aspects are not considered. We propose to use the *Jade* agent platform in order to deploy our middleware, and the

[1] The term of simulation is sometimes understood as "partial emulation" of a network device or of an agent system, for example.

execution environment of the mobile agents will be controlled by a FIPA agent, called agency, which also implements the simulation environment. The high-level architecture of mobile MAS implies several intermediate components, such as a *Java* virtual machine, an agent platform or a *NodeOS*, which, we shall see, make the performance difficult to assess.

The problem of simulation and deployment of *Ecomobile* is mainly addressed in Chapter 4.

2.7 SUMMARY

In order to build an efficient NMS based on mobile MAS, we have introduced three abstraction models which contribute to a separation of concerns.

The computational model focuses on the agent task's operational behaviour. The definition of the tightly and loosely coupled task models classifies the agent task according to its dependencies with the other models: we have seen that a loosely coupled task model leads to a powerful abstraction of mobility functions and therefore facilitates the task design.

The coordination model focuses on the interactions taking place in the mobile MAS. By means of advanced communication mechanisms, mobile agents can improve their capacity to tackle complex problems by exchanging knowledge or regulating their population. The mobility paradigm introduces new possibilities in the coordination schemes, such as meeting-oriented or blackboard-based coordination. Although the separation between computation and coordination has been treated in a wide range of research works, the impact of the mobility paradigm on the coordination and the computational model remains an exciting challenge.

Finally, the navigation model focuses on the agent's migration strategy and the migration functions. The taxonomy of navigation models which has been proposed focuses on the difference between the pre-planned and the stochastic navigation model. While the former is dedicated to deterministic mobile behaviours, as with delegation agents, the latter resorts to environmental information for the migration and thus promotes loosely coupled task models.

Another interesting approach associated with the mobility paradigm has been presented in this chapter: according to the emergent behaviour, mobile agents can deposit specific chemical messages in the environment in order to influence the behaviour of other agents. Stigmergetic communication stems from biological observations of insects colonies and reveals a promising approach towards network management[1].

A behavioural study of a large-scale population of mobile agents requires a simulation framework able to manage the agent interactions on the one hand, and the environmental interactions on the other hand; the network environment in which the mobile MAS will be deployed must also be partially simulated so that all the aspects of the abstraction models can be validated. As we have seen, full simulation does not resort to an agent platform, although certain components of the agent platform must be considered in order to facilitate the deployment on the target environment. According to a second scenario, the simulator is combined with the target agent platform. Virtual locations can then be implemented to simulate the nodes geographically spread in the network and to use logical mobility for the agent migration.

[1] In this context, emergent behaviour is currently under investigation in the OPTIMA project; further details are given in section 7.1.

Designing mobile multi-agent systems is generally difficult because of the numerous agent-to-agent and agent-to-environment interactions taking place and the dependencies between the different system abstractions. According to the present state of the art, the design of mobile multi-agent systems is not supported by any clear methodological framework: developing a specific task often imposes a particular agent architecture with specific behaviours. The design of a mobile MAS implementing a loosely coupled task model for generic objectives remains an important issue.

Part II

Design, Implementation and Simulation

Chapter 3
The Conceptual Framework of *Ecomobile*

In this chapter, we wish to introduce the notion of artificial *ecosystem* as a framework composed of mobile agents evolving in a network infrastructure. We will focus on certain evolutionary aspects of biological systems and will try to apply them to our ecosystem. The notion of ecosystem stems from the science of ecology and can be defined as follows:

"*An ecosystem is a grouping of plants, animals, microbes, etc., living in an explicit unit of space and interacting with one another and with their environment.*" (Department of Environmental Science and Policy, University of California, Davis)[1]

An ecosystem as a whole has the ability to coordinate and has something in common with evolutionary systems whose evolution is purposeless and acts on populations of species, to gain performance despite individuals' activity. However, at the moment, artificial systems are separated from ecosystems in the sense that they are not coupled with the ecosystems; artificial systems by themselves do not have an ability to adapt. "*The real value of evolutionary systems is to create artificial systems that function as part of ecosystems*" [Num95].

Based on ecosystem principles, the *Decentralized Information Ecosystem Technology* (DIET) [M^+01] proposes to define lightweight agents called *infohabitants*, which only have the capability to communicate, and to develop a framework supporting a population of *infohabitants* as a basis for agent-based applications resorting to economic interactions and market-based computation. In [GF01], the notion of computational ecosystems is defined as a framework creating and maintaining value-adding chains of e-services with particular emphasis on the coordination and control of participating parties.

In Chapter 2, we have seen that mobile MAS exhibit a number of fundamental characteristics inherent to any society of living individuals: for example, they can move freely within a delimited environment and can interact with it as well as with each other. In fact, the mobility paradigm, which appears to illustrate the underlying behaviour of any well organized society, has several direct effects, such as the *dispersal* of the individuals in their physical environment or the fact that they exhibit different behaviours depending on their location. From the point of view of indirect effects, the mobility paradigm inherently contributes to the *self-organisation* of the environment, including *self-regulation* of the population. This property results from the moving agents' different geographic positions.

We have also introduced three abstraction models related to the design phase of mobile multi-agent systems: the *computational*, *navigation* and *coordination* models. This separation of concerns will allow us to define the conceptual approach of *Ecomobile*. In the present chapter, we propose to define *Ecomobile* as an artificial ecosystem in which a community of mobile agents living in the network infrastructure interact in an environment represented by the network nodes and links. *Ecomobile* acts as a mobile middleware for the dispersal and the activation of intelligent tasks in an active network infrastructure intended for fully decentralised network management. The regulation of the agent population and its self-adaptability to physical network constraints such as connectivity, are two essential properties which lead to an equilibrium in the system.

[1] http://www.des.ucdavis.edu/classes/ESP10/ecoservices.pdf

Our approach is mainly driven by the characteristics issued from the abstraction models; considering our objectives, we have adopted the minimal requirements of the mobility functions in order to avoid extra-overhead due to the support of obsolete services in the agent architecture and in the agent system. The architecture of *Ecomobile* has resulted in a particular model leading to the use of a common agent architecture for transport networks; our infrastructure also supports multiple operational behaviours based on different approaches like emergent behaviour.

In the beginning, we will introduce the *Ecomobile* model and its main components. Ecosystem principles leading to the population regulation will be then presented and we will try to identify elementary behaviours which may be observed in a society of mobile individuals living in a same environment. The combination of these simple ecological behaviours will determine the individuals' lifecycle, including their interactions with one another and with the physical environment. We will finally present samples of generic task objectives characterizing the operational behaviour of our mobile agents.

3.1 FUNDAMENTALS OF *ECOMOBILE*

Ecomobile is an ecosystem-inspired mobile agent middleware, that is an ecosystem housing a population of mobile entities continuously living within the network environment and respecting the network topology and physical limitations. Any changes occurring in the system must consequently be supported by the agent community.

It is assumed that each network component is able to *host* an agent system on top of a virtual machine or operating system like the active nodes in *Active Networks*, for example. The agent system can also be located close to the node in a separate machine, in which case it should be able to access the local environment through a proxy.

Such a population of mobile agents is obviously useless unless they can be programmed and lead to the implementation of specific tasks. The infrastructure is called a middleware because it constitutes an intermediate layer between the physical resources and the applications themselves. The tasks are deposited in the ecosystem environment by an external entity, either manually or automatically. The population of mobile agents is then responsible for loading the tasks, disseminating them within the network and activating them periodically. These mechanisms will be explained in details in the following sections.

A Threefold Architecture

In *Ecomobile*, the mobile MAS is composed of three distinct active components: the agency acting as the mobile agent system, the mobile agent, and the specific task or operational behaviour. Since the task is physically separated from the mobile agent itself, *Ecomobile* can be regarded as a *threefold architecture*. According to the abstraction models, the mobile agent itself implements both the navigation and the coordination model to form the *Mobile Behaviour Scheme* (MBS), whereas the *Task Objective* (TO) refers to the operational behaviour and therefore to the computational model. MBS and TO constitute a particular instantiation of the abstraction models. It can also be said that the population of mobile agents is the substratum implementing the ecosystem-related concepts of the MBS. The *Ecomobile* model is depicted on Figure 3-1.

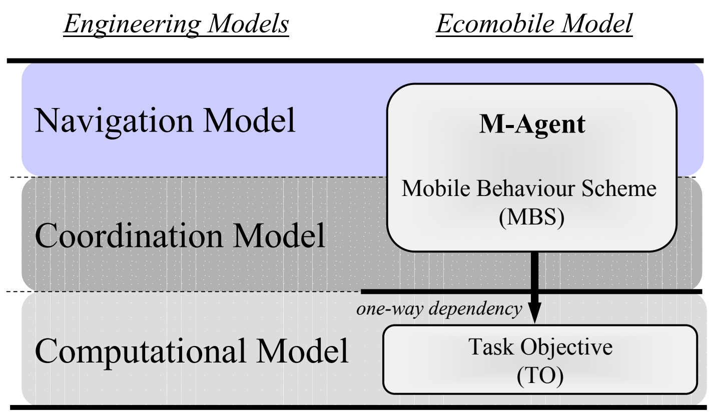


Figure 3-1. *Ecomobile*: an instantiation of abstraction models

As can be seen on this figure, the task in our model is clearly separated from the agent itself, so that *Ecomobile* appears to constitute an innovative approach: although the task patterns presented in the previous chapter allow the designer to make a distinction between the operational behaviour and the rest of the agent activities, the architecture finally results in the implementation of a single entity, the mobile agent. In *Ecomobile*, the ecosystem is activated even though no task has been implemented; the middleware is both *mobile* and *active*. The task objective, which corresponds to the operational behaviour or the application itself, is inserted into the ecosystem environment and becomes part of it; the place where the task has been deposited can be visited by any agent of the population: when an agent discovers the task objective, it loads the task into its *context* and activates it. In order to guarantee this functionality, it is necessary to make sure that the environment is regularly visited by the agent society so that the agent behaviour influences the dissemination of the task within the network and activates it at different locations. The density of the population and the population size therefore play an important role in the successful realization of this approach; the more often the nodes are visited, the more often the tasks are executed. The task objectives have no direct influence on the agent population.

From that description, it appears that the agent is not aware of the task description and that, reciprocally, the task has no effect on the agent behaviour, as far as the migration and coordination mechanisms are concerned. The task is loaded and activated by the MBS so that there is a dependency relationship only from the MBS to the TO. The agent task consequently relies on a loosely coupled task model (see Section 2.1.2). The impact of the dependency between the MBS and the TO will be shown further down in this chapter.

This type of architecture also leads to the mobile agent and the task objective having distinct lifecycles: the task objective does not depend on the existence of one mobile agent in particular but can be transported by different mobile agents during its lifecycle. Whereas the task can be loaded by any individual in the ecosystem, it can also be *offloaded* into the environment on behalf of the mobile agent or of the task itself. Once it has returned to the environment, the task is suspended until another agent of the population loads it again and pursues its execution over the network.

Consequently, the mobile agents of the ecosystem are used in order to transport and to activate the task objectives. The MBS also controls the task dissemination within the network by means of a cloning mechanism and enables the cooperation between the task objectives themselves. The *Ecomobile* approach has a profound effect on the structure of the computational model: since the task has no influence on the mobile agents' destination, the decisions concerning the TO's current location belong to the task objective itself. If the next destination given by the MBS is not satisfactory, the task can "leave" the agent, i.e. decide to be offloaded at its current location. The mobile agents' statistical visits, which emerge from the self-regulatory population property of the ecosystem, will allow the task to be reloaded and propagated to the expected destination.

The threefold architecture of *Ecomobile* consequently defines two logical navigation planes: the navigation of mobile agents and the navigation of task objectives; mobile agents and task objectives can have different trajectories; while the mobile agent can implement a stochastic navigation model with active migration, any navigation model can be implemented at the task objective level; the task objective will however resort to a synchronous passive migration.

Ecomobile has been implemented in *Java*[1], which is currently the most efficient object-oriented programming language in software agent technology; *Java* is platform-independent and provides efficient mechanisms such as serialization, RMI communication, security framework and dynamic class loading mechanisms; the most popular agent platforms are therefore implemented in *Java*.

3.2 NODE ENVIRONMENT

The presentation of *Ecomobile* principles first requires to introduce the nodal environment, which is mainly composed of several model-specific components, such as the *agency*, the *place* and the *blackboard*. Mobile agents in *Ecomobile* are called *M-agents*; a description of their architecture will then allow us to examine in details the *Ecomobile* model and the ecosystem-inspired approach.

The organizational architecture within an active node[2] is depicted on Figure 3-2. The MIB contains the component's managed objects accessible to the mobile agents through a specific OSI agent, for example, or any control unit. The *execution environment* is generally a *Java* virtual machine; a FIPA agent platform runs on top of it. Further details concerning this part of the system are outlined in the section devoted to the issue of deployment (4.4).

[1] http://www.sun.com/java

[2] In this context, *active node* refers to a node capable of hosting external code and providing an execution environment, such as active node in *Active Networks*; a proxy can deal with legacy *passive* nodes (see Section 1.4).

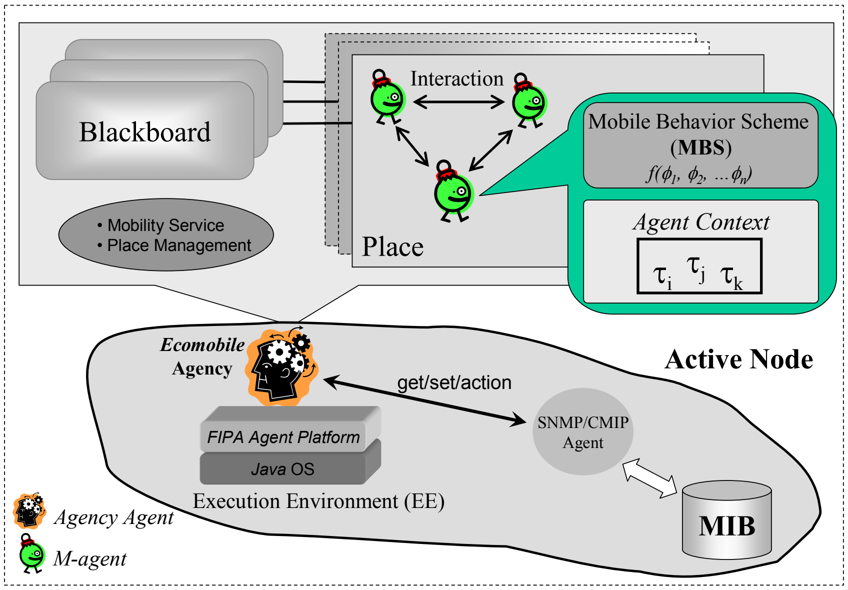


Figure 3-2. Active node environment with respect to *Ecomobile*

The main *Ecomobile* components are the *Agency*, the *Blackboards* and the *Places*. They are described in the next sections.

3.2.1 The Agency

The *Agency* is a FIPA agent which implements the *Ecomobile* components; it may co-exist with other stationary agents. The agency provides *Ecomobile* with an adequate mobile agent environment different from the active node's execution environment. The agency is responsible for managing the *Ecomobile* environment containing the blackboards and the places; it governs the M-agent activities and manages the agent-to-agent and agent-to-environment interactions, so that security policies and access control can be implemented into the agency. Communication between the agency and the OSI agents can be provided by JIDM gateways, for example (see Section 1.1.1). The agency also provides the mobility services necessary for the M-agent migration. The migration itself is achieved between the *Ecomobile* agencies via a specific ACL message and protocol.

Details concerning the execution environment for the M-agents and the deployment of the FIPA agency are given in Chapter 4.

3.2.2 The Place

The place, in *Ecomobile*, has two main functions: at first, the place is used as a basic *location concept*; M-agents migrate from place to place. Secondly, the place is used as a *coordination space* between M-agents. Let us now examine in details the two notions.

Places are interconnected to form the intra-agency and the inter-agency connectivity. The intra-agency connectivity allows M-agents to migrate virtually within an agency, whereas the inter-agency connectivity allows them to migrate physically from one node to another. M-agents use place identifiers called *Universal Place Identifiers* (UPI), which are defined as follows:

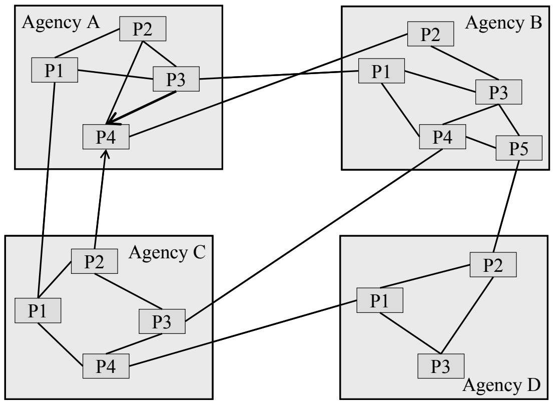
UPI ::= (NodeAddr, PlaceID) where *NodeAddr* is the node's physical address (e.g. MAC or IP address)

The *PlaceID* has to be unique within a same platform so that *UPI* results in a unique identifier for the whole network. The agency maintains a connectivity matrix describing the place topology. According to the *UPI*, the agency determines whether the agent has to be migrated physically or not. Inter-agency connectivity can be retrieved from any place via the agency.

The connectivity matrix depicted on Figure 3-3, which is a subset of the full matrix, is simply defined as follows:

$$f(\mathrm{UPI}_a, \mathrm{UPI}_b) = 1 \text{ if there is a link between } \mathrm{UPI}_a \text{ and } \mathrm{UPI}_b$$

In this matrix, the asymmetry therefore corresponds to unidirectional links.



	$UPI(A,P3)$	$UPI(A,P4)$	$UPI(B,P1)$	$UPI(C,P2)$
$UPI(A,P3)$	0	1	1	0
$UPI(A,P4)$	0	0	0	0
$UPI(B,P1)$	1	0	0	0
$UPI(C,P2)$	0	1	0	0

Figure 3-3. Places intra-/inter-agency connectivity and partial connectivity matrix

In *Ecomobile*, priority is given to the mobility paradigm, and M-agents are not allowed to communicate unless they reside in the same location. In the context of our work, the place semantics is a mapping leading to the reflection of the physical and logical connectivity of the network and implying that, in the most simple case, each node is associated to a place, so that the place connectivity corresponds to the node connectivity.

In Section 7.2.2, we propose to map a place semantics matching the wavelength planes; a place per wavelength is defined and the intra-agency connectivity matches the wavelength switching matrix.

3.2.3 The Blackboard

Each place is associated to a passive blackboard which is part of the ecosystem environment and allows M-agents to deposit information intended for other agents. Each M-agent only has visibility on the blackboard associated to the place in which it resides. The blackboard is also used as a repository for the task objectives.

3.3 The M-Agent

An M-agent is a mobile agent considered as an individual in the ecosystem. The general architecture of an M-agent, which is depicted on Figure 3-4, reveals two distinct parts: the MBS and the Agent Context. The MBS implements the navigation and coordination models and is defined at the design time; subsequent dynamic modifications in the behaviour are possible by means of self-adaptive functions, which are currently not implemented in *Ecomobile*: we propose to examine the ecosystem behaviour with a fixed implementation of MBS.

Unlike the MBS, the agent context constitutes the dynamic part of an M-agent. The tasks which are deposited into a blackboard are loaded dynamically into the agent context; since the agent population does not depend on the TO, the existence of the ecosystem does not depend on any task[1].

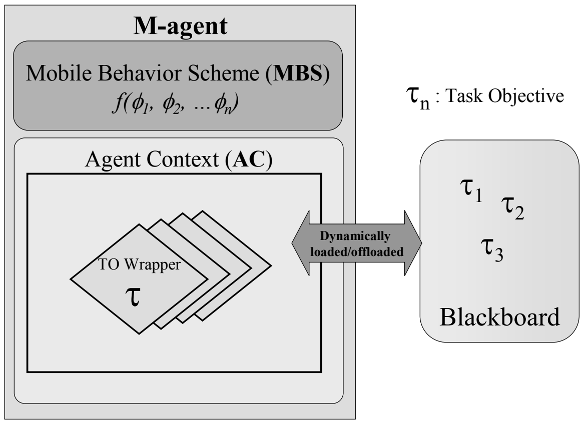

M-agent
Mobile Behavior Scheme (MBS)
$f(\phi_1, \phi_2, \ldots \phi_n)$
Agent Context (AC)
TO Wrapper
τ
Dynamically loaded/offloaded
τ_n : Task Objective
τ_1
τ_2
τ_3
Blackboard

Figure 3-4. M-agent Architecture

Figure 3-5 reveals a simplified version of the UML class diagram representing the M-agent's major components. For the sake of readability, only the important operations are depicted in the model, and their arguments have not been included.

The central component is the *`M-agent`* class, which represents the mobile agent itself, and actually implements the *Mobile Behaviour Scheme* (MBS). The M-agent class inherits from a class called *`LambdaAgent`*[2], which refers to *optical* agents that is, a mobile agent able to travel along a wavelength; the agent's transport occurs through the multiplexing of a specific communication channel in the overhead structure of the optical frame [RS99]. *`LambdaAgent`* implements common functionalities for M-agents, such as the execution of multiple MBS, for example, or the interaction with the environment.

[1] This characteristic allows us to designate the ecosystem as a *middleware*, on top of which the task objective will be *programmed* with an appropriate computational model.
[2] This root class is related to the initial motivation of our work namely, the development of an intelligent optical transport network.

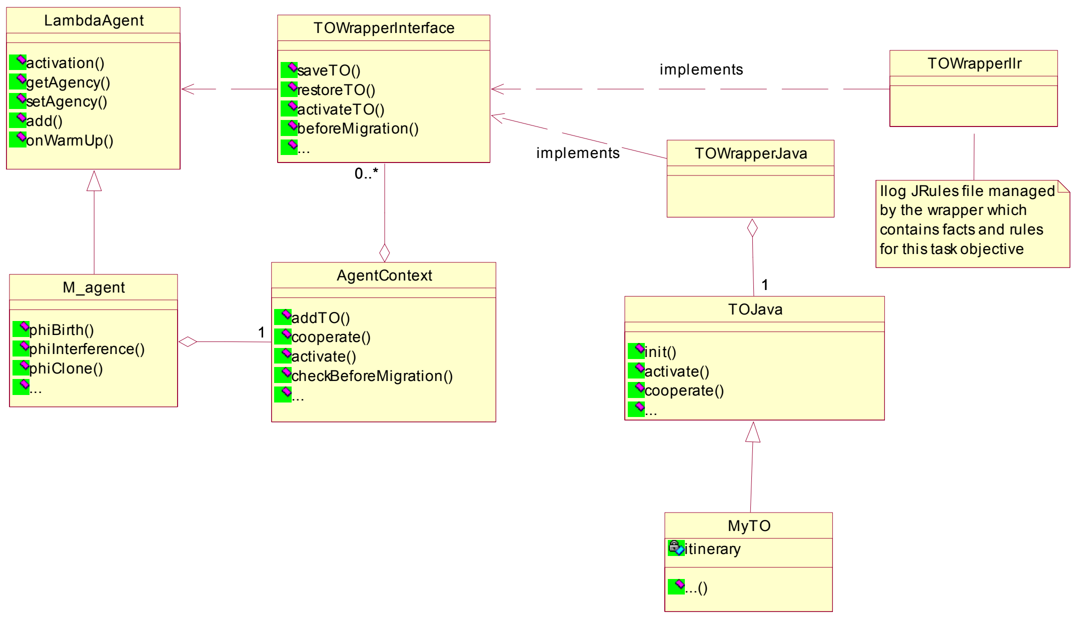


Figure 3-5. UML Diagram of M-agent components in *Ecomobile*

The M-agent also contains an agent context (class `AgentContext`), which is defined as the container dedicated to task objectives. Initially, the agent context is empty and will be filled with the task objectives found in the blackboard associated to the current M-agent location. The agent context defines the methods applied to the whole container. The role of these methods will be clarified along the subsequent sections. The task objective wrapper (class `TOWrapperInterface`) provides the agent context with an interface to the task objective which can be expressed in *Java* or in a rule-based language such as JESS for example. The wrapper is discussed in 3.5.3.

The objects, or variables, moving with the M-agent, are called *frontal* objects, the objects in the agency which are specific to *Ecomobile* are known as *nodal* objects and the objects specific to the local node as *environmental* objects; these denominations are based on the *Wave* terminology introduced in Section 1.3.2.

A *Java* task objective extends a base class used for *Java*-based tasks, called `TOJava`, which defines the callbacks invoked by the MBS. These methods are used to express the dependency between the MBS and the TO. `TOJava` includes a special method in order to serialize itself and to be deposited in the blackboard. Other kinds of TO can be formulated in other languages such as rule-based languages; for example, the class `TOWrapperIlr` provides an interface to *Ilog JRules* language (see Section 3.5.2).

We now proceed to examine in details the MBS and the task objectives.

3.4 THE MOBILE BEHAVIOUR SCHEME

According to our mobile middleware, the mobility paradigm reveals a "living" society of software agents navigating constantly within the network. In *Ecomobile*, the agents' birth and death leads to an emergent self-regulatory phenomenon; our particular architecture model leads to the design of particular bio-inspired behaviours accommodating the dissemination and activation of complex network management tasks within the network infrastructure.

The MBS describes the M-agent behaviour from the point of view of navigation and agent-to-agent interaction; it defines eight *Reactive Behaviours* (RB or Φ-behaviours) characterizing the activities of the ecosystem individuals, such as migration, self-reproduction, action, etc., on the one hand, and activities implying interactions with other individuals, such as communication or competition, on the other hand. The term *reactive* will be explained in Section 4.3.

Ecology enlightens the impact of the individuals' behaviour on the ecosystem population: the competition between individuals in a natural ecosystem, in particular, plays a fundamental role in the self-regulation of the population. Competition can appear between individuals belonging to the same species, in which case it is known as *intraspecific* competition, or belonging to different species, in which case it is called *interspecific* competition [BHT90]. We aim at exploiting a particular kind of competition called *territoriality*, in order to address the problem of the density control and of the size of mobile agents' population size. Territoriality is defined as an *interference*[1]-based intraspecific competition occurring between members of the same species for the control of territories. The significance of territoriality lies in the fact that individuals of a territorial species that fail to obtain a territory often make no contribution whatsoever to future generations. Consequently, the density-dependent birth and mortality rate leads the territoriality to have a particularly powerful regulatory influence on the populations concerned[2].

This highly interesting density-dependent and self-regulatory property leads us to propose the design of a mobile behaviour scheme approximating the territoriality paradigm issued from the ecosystem theory, and based on active interference between individuals. Such a design requires a judicious combination of Φ-behaviours. In our context, the territory refers to the network node.

3.4.1 Notations

A number of definitions are now introduced in order to formalize the description of Φ-behaviours. Most of the following functions are time-dependent; for simplification reasons, we have omitted the time in their arguments.

Let us assume the following definitions:

$\boldsymbol{\lambda_i}$	M-agent *i*
P ::= { $\boldsymbol{\lambda_1, \lambda_2, \lambda_3, \ldots}$ **}**	Population of M-agents
$\boldsymbol{\Phi_i}$	Reactive behaviour *i*
$\boldsymbol{\tau_i}$	Task objective *i*
$\boldsymbol{\tau_i(\lambda_j)}$**{*callback*}**	Invocation of a TO callback by the MBS on τ_i on M-agent *j* (see 3.5.1)
AC($\boldsymbol{\lambda_i}$**):: = {**$\boldsymbol{\tau_1, \tau_2, \ldots \tau_n}$**}**	Agent context of M-agent *i*
$\boldsymbol{\theta(\lambda_i)}$	Gives the behavioural function currently executed by M-agent *i*
Pl($\boldsymbol{\lambda_i}$**)**	Gives the actual location for M-agent *i* by means of the *Universal Place Identifier* (UPI) for M-agent *i*
Ref($\boldsymbol{\lambda_i}$**)**	Gives the reference to another M-agent which is stored in M-agent *i*
Dest($\boldsymbol{\lambda_i}$**)**	Calculate the next destination (UPI) of M-agent *i* by means of the connectivity matrix
Bl(UPI$\boldsymbol{_a}$**):: = {**$\boldsymbol{\tau_1, \tau_2, \ldots \tau_n}$**}**	Designates the TOs in the blackboard attached to place UPI_a

[1] Interference is a kind of competition between individuals interacting directly.
[2] In ecology, territoriality is a particularly important and widespread asymmetric intraspecific competition.

An M-agent can execute only one Φ-behaviour at the same time. All the Φ-behaviours can be parameterized and changed dynamically.

In the following section, we use the above definitions in order to express the influence of Φ-behaviours on the M-agent architecture and thus to define a semi-formal description of the reactive behaviours.

3.4.2 Reactive Behaviours

The reactive behaviours (Φ-behaviours) represent the primary *ecological* behavioural functions that an M-agent can perform in the ecosystem; there are eight Φ-behaviours:

$\phi_{\mathbf{birth}}$	Birth of an M-agent	(1)
$\phi_{\mathbf{migration}}$	Migration of an M-agent	(2)
$\phi_{\mathbf{action}}$	Activation of task objectives (an M-agent performing Φ_{action} can not be "seen" by other agents)	(3)
$\phi_{\mathbf{interference}}$	Competition with another M-agent using direct interaction	(4)
$\phi_{\mathbf{absorption}}$	Absorption of all resources (knowledge) of another M-agent	(5)
$\phi_{\mathbf{clone}}$	Cloning of an M-agent	(6)
$\phi_{\mathbf{dwelling}}$	Dwelling at the current location; no special action is performed (The agent is able to sense another interfering M-agent)	(7)
$\phi_{\mathbf{death}}$	Death of an M-agent.	(8)

It has to be noted that we have renamed certain Φ-behaviours previously presented in [RSH01] and [RS02]. The following Φ-behaviours are concerned by these changes: $\phi_{\mathbf{migration}}$ (previously ϕ_{move}), $\phi_{\mathbf{action}}$ (previously ϕ_{exec}), $\phi_{\mathbf{interference}}$ (previously $\phi_{contact}$), $\phi_{\mathbf{absorption}}$ (previously ϕ_{merge}) and $\phi_{\mathbf{dwelling}}$ (previously ϕ_{wait}).

The following Φ-behaviour descriptions reflect the behaviours implemented into *Ecomobile*; we also discuss possible variations in their implementation. The main actions are given within square brackets and separated by a semi-colon.

(1) $\phi_{\mathbf{birth}}$

The Φ_{birth} behaviour corresponds to the birth of an M-agent: an M-agent is introduced into the ecosystem at a specific place.

Pre-conditions: < « > (A new agent is created in the location UPI_a)

$\phi_{birth}(\lambda_i)$ ::= [P := P » { λ_i } ; AC(λ_i) = « ; Pl(λ_i) = UPI_a]

Since the M-agent appears as a new individual in the ecosystem, a monitoring function is triggered during the execution of this behaviour, so that we can keep track of the population size during the simulation.

(2) $\phi_{\mathbf{migration}}$

The $\Phi_{migration}$ behaviour allows M-agents to move from one place to another; it implies active migration with a single-hop destination given by UPI_b.

This behaviour, which implements the M-agents' migration strategy, plays a central role in the ecosystem behaviour. The migration function calculates the M-agent's next destination by means of the connectivity matrix defined in the agency. In *Ecomobile*, the mapping semantics involves synchronization between the connectivity matrix and the underlying network infrastructure; the removal of a network link leads to the removal of the corresponding link in the connectivity matrix.

According to the computational model defined in *Ecomobile*, migration is not influenced by the task objectives in any way.

Pre-conditions: $< Pl(\lambda_i) \int UPI_b >$

$\phi_{migration}(\lambda_i) ::= [\ Dest(\lambda_i) = UPI_b\ ;\ \forall\tau_j \in AC(\lambda_i) => \tau_j(\lambda_i)\{beforeMigration\}\ ;\ Pl(\lambda_i) = UPI_b\ ]$

The above expression shows that the task objectives are informed of the next destination before migrating, so that they remain able to perform any specific action; if the destination is not convenient, for example, the TO can offload itself in the blackboard.

The migration function retrieves the possible destinations from the agency and makes a random decision based on these destinations; the destination function *Dest()* determines whether the destination has already been pre-computed in another Φ-behaviour like Φ_{clone}.

According to a different migration strategy, the destination could also be selected by means of a "round-robin" mechanism operated on the list of available output ports; this approach allows the network to be explored in a more efficient manner.

The migration time is computed randomly. In case of intra-platform migration (virtual migration), the simulation time is set to 0; more information concerning the time reference is given in Section 4.1.

(3) ϕ_{action}

This behavioural function performs loading and activation of task objectives: the agent first looks for eventual TOs in the blackboard, loads them into its context and activates them. All TOs present in the agent context are then activated via the TO wrapper.

Pre-conditions: $< Pl(\lambda_i) = UPI_a >$

$\phi_{action}(\lambda_i) ::= [\ AC(\lambda_i) := AC(\lambda_i) \gg Bl(UPI_a)\ ;\ \forall\tau_j \in AC(\lambda_i) => \tau_j(\lambda_i)\{cooperate, activate\}\ ]$

Once the activation of these task objectives has been initiated by the M-agent, it can not be interrupted any more. At each TO activation, two callbacks are invoked: the first one enables an inter-TO cooperation (see 3.5.3) while the second one is used to activate the TO.

Several approaches are possible in order to implement this behaviour. Depending on the local processing power, we can imagine starting a thread per each TO in order to improve the performance. In this case, the agent context has to wait until all TOs threads are terminated. In our implementation, however, the TOs are executed atomically and sequentially. With regards to the computational model, the order of TO activation is not specified that is, the TOs can not make assumptions on their execution order.

An execution time can be also introduced for simulation purposes; the time spent to perform the tasks may therefore temporarily and locally influence the ecosystem behaviour. In our simulation however, we assume fast TO execution and do not consider the execution time since it may be included in the migration time.

(4) $\phi_{interference}$

This behaviour allows an M-agent to interfere actively and directly with another co-residing target M-agent: the first agent enquires about the presence of any other M-agent at the same place; the only condition for the success of the interference is that the target M-agent is performing the $\phi_{dwelling}$ behavioural function.

Pre-conditions: $< Pl(\lambda_i) = Pl(\lambda_j)\ ;\ \theta(\lambda_j) = \phi_{dwelling} >$

$\phi_{interference}(\lambda_i) ::= [\ Ref(\lambda_i) := \lambda_j\ ]$

According to the above expression, the interference mechanism occurs through direct coupling between two M-agents resorting to a meeting-oriented coordination paradigm (see 2.2.3). The interfering agent receives the reference to the sensed agent. If several M-agents are present in the same place, the interfering agent makes a random selection. The concept of territoriality leads interactions to occur between a pair of individuals.

(5) $\phi_{absorption}$

This behaviour allows an M-agent to "absorb" the resource that is, the knowledge belonging to another M-agent: in other words, this operation consists in transferring the entire agent context from one M-agent to the other.

Pre-conditions: $< Pl(\lambda_i) = Pl(\lambda_j)\ ;\ Ref(\lambda_i) = \lambda_j >$

$\phi_{absorption}(\lambda_i) ::= [\ AC(\lambda_i) := AC(\lambda_i) \gg AC(\lambda_j)\ ;\ \forall\tau_k \in AC(\lambda_i) => \tau_k(\lambda_i)\{cooperate\}\ ]$

The insertion of new task objectives into the current agent context of the surviving M-agent (λ_i) will trigger the inter-TO cooperation mechanism (see Section 3.5.3).

(6) ϕ_{clone}

The cloning operation allows an M-agent to create a replication of itself; the cloned M-agent (offspring) contains the same knowledge as the parent. The child's next destination may however differ from the parent's one; although the destination UPI_b is assigned to the clone, the M-agent does not perform any migration in this behaviour.

Pre-conditions: $< Pl(\lambda_i) = UPI_a >$

$\phi_{clone}(\lambda_i) ::= [\lambda_j ::= \lambda_i;\ P := P \gg \{\ \lambda_j\ \};\ AC(\lambda_j) = AC(\lambda_i);\ Dest(\lambda_j) := UPI_b\ ]$

The cloning operation is usually related to the concept of migration because it generally becomes necessary for the agents to clone themselves in order to improve the exploration process within the network; load-balancing between several machines, for example, can be achieved simply with successive cloning operations [SSC+98]. In *Ecomobile*, the cloning operation naturally matches the individuals' reproduction mechanism inherent to ecosystem principles; according to this mechanism, the offspring is used to disseminate replica of task objectives within the network and to provide a birth contribution in the self-regulated M-agent population.

The number of clones itself strongly depends on the migration strategy: each new cloned M-agent is assigned a destination and must consequently not be re-computed in the migration behaviour ($\phi_{migration}$).

As in $\Phi_{migration}$, different cloning strategies are possible. It could for example be decided that the M-agent is cloned to the link by which the parent has arrived; moreover, a clone could also be generated in the direction which will subsequently be taken by the parent agent. Finally, the parent destination needs to be established during this operation so that the next call to $\phi_{\text{migration}}$ will be in line with the cloning strategy.

According to the strategy which has been chosen, the population size can be subject to serious variations which could eventually influence the propagation rate of task objectives; in *Ecomobile*, the MBS has been conceived to generate clones for all the available output links, including the parent's arrival link; however, no clone to the parent's destination is generated. In other words, the number of clones equals the nodal degree.

(7) ϕ_{dwelling}

This behaviour makes the agent able to sense the environment and therefore to react to the presence of another agent. From an ecological behaviour point of view, the agent is simply spending some time in its environment doing nothing particular.

Pre-conditions: $< \mathrm{Pl}(\lambda_i) = UPI_a >$
$\phi_{\text{dwelling}}(\lambda_i)$ ::= [«]

The waiting time, which actually gives the M-agent an opportunity to meet other M-agents, has been proposed in co-evolutionary agent systems [SS99], for example, developed for multiple node and span failure restoration. A mechanism called *Early-Route Completion* (ERC) allows two mobile agents to meet at an intermediate node and to exchange route information. This strategy resembles the *MITAgent* approach. The meeting opportunity can also be improved by means of a probabilistic waiting time.

In *Ecomobile*, the waiting time can be either fixed or calculated dynamically. Effects of the waiting time on the ecosystem behaviour are discussed in Section 5.3.3

(8) ϕ_{death}

The Φ_{death} behaviour corresponds to the death of an M-agent that is, its removal from the ecosystem; the population size is updated by the monitoring function.

Pre-conditions: < « >
$\phi_{\text{death}}(\lambda_i)$ ::=[$\mathrm{Pl}(\lambda_i)$ = « ; $\mathrm{AC}(\lambda_i)$:= «; $P := P \setminus \{ \lambda_i \}$]

The M-agent disappears from its current place, including all TOs present in the agent context at the death time.

It appears from the previous definitions that each Φ-behaviour has its own set of parameters making the agents able to accommodate various conditions during the ecosystem lifetime. Density control, for example, can be more finely tuned by adjustment of the waiting time for the Φ_{dwelling} behaviour or the number of generated clones. The number of ecosystem parameters obviously makes an exhaustive behavioural analysis impossible. In order to limit our investigations to a reasonable number of behaviours, we have chosen a particular subset of parameter values which will be discussed in Chapter 5.

We now describe two different mobile behaviour schemes approximating the territoriality paradigm and therefore leading to a density-dependent self-regulatory population of M-agents.

3.4.3 Low Diffusion

Mobile behaviour schemes can assume a variety of forms; we have chosen to present two MBS which exhibit interesting results from the point of view of simulation (see Chapter 5). We shall begin with a MBS describing a simple M-agent behaviour based on the territoriality paradigm.

In this MBS, the agent starts migration directly after its birth (Φ_{birth}). When it has reached a new place, it activates the Φ_{action} behaviour, which consists in loading the possible task objectives present in the associated blackboard into its context. The agent, which "defends" its territory (its place), enquires if another M-agent is currently residing in the same place: it senses a visible M-agent, as defined by $\Phi_{interference}$, provided the sensed agent is performing a $\Phi_{dwelling}$ behaviour.

The M-agent can then perform two different actions depending on whether another M-agent has been sensed or not. In case of success, the sensing agent performs a $\Phi_{absorption}$ which implies a transfer of all TOs from the sensed agent into its agent context, irrespective of the task nature. In the proposed MBS, the sensed agent dies (Φ_{die}) whereas the agent enriched with the new knowledge leaves the current place towards another place ($\Phi_{migration}$) in order to "conquer" other nodes[1]. If the M-agent fails to sense another agent, it performs a Φ_{clone} behaviour and continues its migration over the network.

Since the diffusion of the M-agents within the network is relatively "slow", this kind of mobile behaviour scheme is called *low diffusion* - or *MBS-low*. This *MBS-low* is depicted on Figure 3-6.

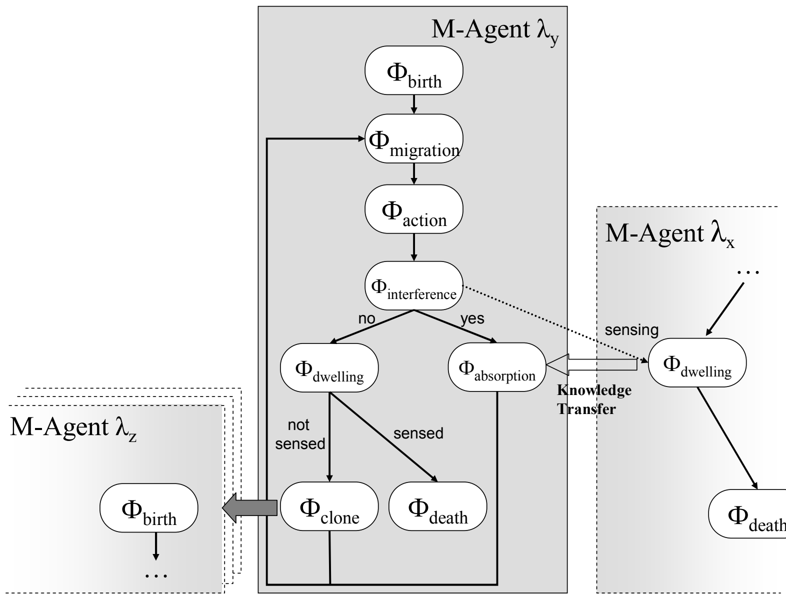


Figure 3-6. MBS Low Diffusion - M-agent λ_y interacting with λ_x

As we have seen, the above described MBS leads to M-agent cloning, unless a co-residing agent can be contacted. According to the meeting opportunities, the network exploration can occur at various speeds:

[1] In this context, "conquering" the network simply means exploring the nodes; no specific mark is deposited by the M-agent in its environment.

when a contacted M-agent has died, the surviving M-agent follows one of the possible output links, leaving the other links unexplored for the time being[1].

3.4.4 High Diffusion

The second MBS we introduce is rather similar to the first one, with the exception of two modifications: in order to ensure that no other agent is sharing its location place during a certain time (defined by $\Phi_{dwelling}$), the M-agent iterates the interference-absorption scheme as shown on Figure 3-7.

According to the size of the agent context, the quantity of knowledge or task objectives transferred is therefore higher than in the previous MBS.

This behaviour may result in the consecutive elimination of an important amount of agents in a short time; in order to compensate for the high mortality rate of this iterative process, the M-agent activates a cloning process before leaving the node so that all the available output links are investigated by the M-agents at the migration time. This behaviour, which is similar to a breadth-first parallel search, leads to a faster diffusion of M-agents and allows exhaustive exploration of the network. This is why this mobile behaviour scheme is referred to as *high diffusion* - or *MBS-high*.

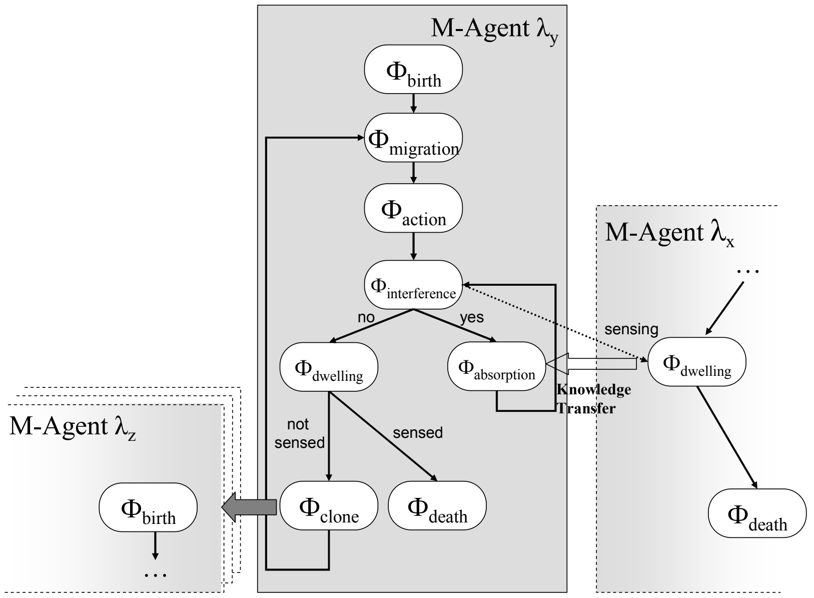


Figure 3-7. MBS High Diffusion - M-agent λ_y interacting with λ_x

The interesting diffusion property of this MBS, however, does not guarantee superior system performance. The simulation has shown that the *MBS-high* leads to a reasonable increase in population size. Still, the variation of the birth and mortality rates are important, so that more processing time is required to achieve the creation and removal of M-agents.

[1] As explained in the definition of $\Phi_{migration}$ (see Section 3.4.2), the migration strategy can contribute to a better exploration of the network.

The interference-absorption loop causes another side effect: according to the network topology, a node's high connectivity degree can lead an M-agent to waste an indefinite period of time in this loop because of the high number of M-agent visits. The task objectives loaded in the M-agent are compelled to stay in the current place because of this immobility. In order to avoid this problem, we propose to implement a heuristic-based waiting time function into the Φ_{dwelling} behaviour; the heuristics, which is based on the number of M-agents arriving at a node, compared to the number of agents leaving the node, is explained in Section 5.3.2.

3.5 Task Objectives

In the previous sections, we have discussed the agent behaviour from the navigation and coordination point of view; we have defined two mobile behaviour schemes characterizing a particular ecosystem behaviour called territoriality. We now propose to examine in details the *Task Objective* (TO), which describes the operational behaviour; in other words we are going to concentrate on the characteristics of the computational model.

A TO describes a particular task, a network management function in our context, which is defined by means of a programming language such as *Java*, a rule-based language or any interpreted language like *Wave*, for example; each task actually represents a specific class with respect to object-oriented conventions. A TO can therefore have frontal objects constituting the task's knowledge and moving with the agent in a persistent way. A TO can be multiply instantiated by the M-agents within the network, as the ecosystem is able to handle multiple instances of task objectives, irrespective of the tasks' nature. The TOs themselves have to be informed whether interactions can take place between TO instances of the same class or even between different classes. An interaction mechanism is activated by the MBS at a certain moment of the agent lifecycle and allows the TOs to *intelligently* cooperate[1] in order to accomplish the overall goal.

TOs are initially deposited into blackboards; they are loaded dynamically by the ecosystem via M-agents and are spread in the network automatically through the invocation of the standard *Java* `clone()` method. Decisions on the TO lifecycle and its own trajectory within the network only depend on the TO itself; the TO can leave the M-agent and be put back into the blackboard by means of an offloading mechanism. If the TO claims to be persistent, during its initialization, the M-agent can decide to offload the TO, as it may happen that no sufficient place is available to put the TO in the agent context during the absorption phase for example. When the TO is not persistent, the M-agent simply ignores it.

According to the computation model, the TO design relies on a loosely coupled task model and imposes particular key concepts, such as the current location, for example, which is transferred by the MBS; furthermore, the passive migration of a TO may also require specific decisions about its destination.

The cooperation, which is asynchronously activated by the MBS, must be designed so that it does not depend on the time. Finally, the TO is informed of the agent's destination right before the migration, and it may decide to leave the M-agent if the destination is not appropriate. Although the TO does not invoke the cloning mechanism, it has to duplicate its frontal objects appropriately.

The next sections outline the characteristics of the TO such as its lifecycle, programming language and interactions between the MBS and the TO.

3.5.1 Lifecycle and Callbacks

In the *Ecomobile* model, the mobile behaviour scheme leads to a self-regulated population of M-agents, so that the number of agents does not depend on the task objectives. This approach however results in a particular trade-off between the number of agents and their size: the task objectives are loaded into the agent context of the M-agent and therefore contribute to its size[2].

A possible approach towards the control of the TO lifecycle consists in resorting to an energy concept [Bau99] developed for mobile agents to control their garbage collection; the algorithms can be implemented at a TO level. In the beginning, the TO has a certain amount of energy; every time the agent accedes local resources, it consumes some of this energy. Once the TO has run out of energy, it becomes an "orphan" and can be removed by a "garbage collection" task objective. According to this approach, the automatic dispersal of TOs and their cooperation mechanisms allow them to gain control over a particular class of TOs.

Each TO can implement different lifecycle strategies. *Callbacks* allow the TOs to manage their lifecycle. The TOs inform their wrapper whether they are still alive, or if they have to be discarded. In *Ecomobile*, each TO owns an internal state associated to the agent behaviour; this state is managed by the wrapper in accordance with the MBS.

The different states of a task objective are depicted on Figure 3-8.

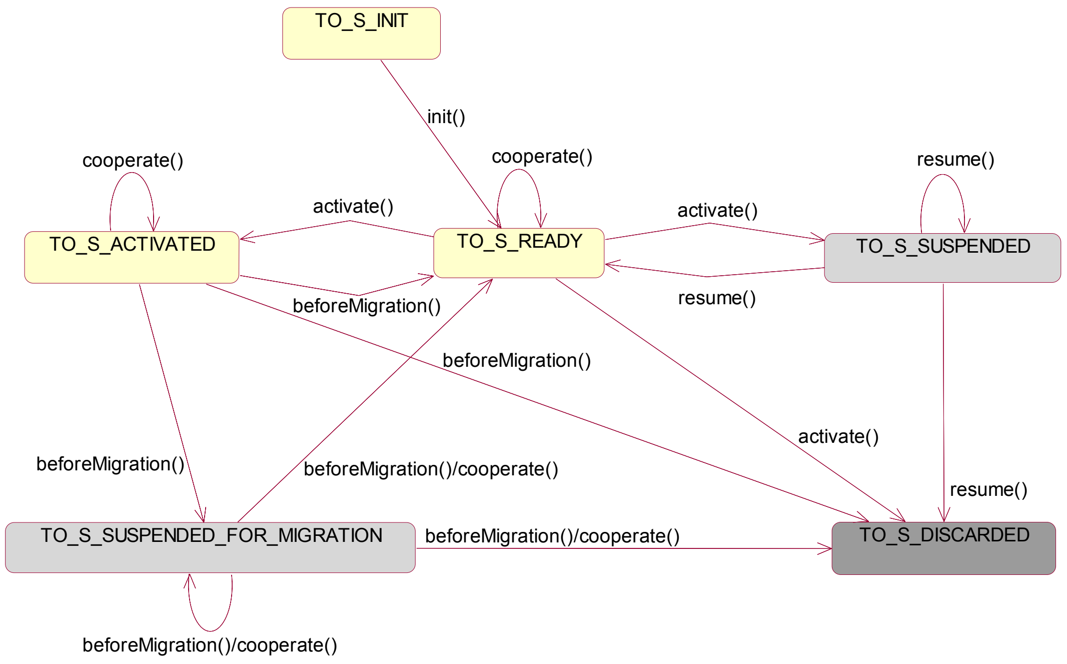


Figure 3-8. State and Callbacks-based Transition Diagram in TO Lifecycle

Transitions from one state to another are performed by the callbacks implementing the TO behaviour. The states *TO_S_SUSPENDED* and *TO_S_SUSPENDED_FOR_MIGRATION* are associated to serialized

[1] In this context, cooperation refers to a general interaction scheme and does not exclude "competition" between TOs.
[2] The agent context can manage a number of task objectives up to a certain limit.

instances of the TOs that have been offloaded in the blackboard. State *TO_S_DISCARDED* refers to a task objective which is not alive any more. In the following, TO states and callbacks are subject to a detailed description. TO callbacks are introduced in Table 3-1 and are defined in the root class `TOJava`.

```
// Callbacks
public Boolean init(TOWrapperInterface wrapper);
public Boolean activate(TOWrapperInterface wrapper);
public Boolean cooperate(TOWrapperInterface wrapper, TOJava otherTO);
public Boolean resume(TOWrapperInterface wrapper);
public Boolean beforeMigration(TOWrapperInterface wrapper);

// Called when the task objective must be cloned
public Object clone();

// TO user methods
public void setPersistent(Boolean state);
public void cleanEnv(TOWrapperInterface wrapper);
public void offload(TOWrapperInterface wrapper);
public void setPriority(int priority);
public void discard();
```

Table 3-1. Task objectives callbacks and user methods

In all the callbacks, the task objective wrapper is passed as an argument in order to inform the TO of the current location and, as far as `beforeMigration()` is concerned, the next destination; the wrapper includes a reference to the agency so that information about the local environment can be retrieved. All the callbacks return a `boolean` telling the wrapper if the TO is discarded; when the TO wishes to be discarded, the wrapper informs the agent context that the task objective must be removed.

In *Ecomobile*, the inter-TO cooperation provides an efficient mechanism in order to perform knowledge exchange between different instances of task objectives and in order to avoid knowledge redundancy; it therefore plays a central role in the regulation of the agent context size. The cooperation mechanism relies on a *master/slave* paradigm between two TOs; the callback `cooperate()` provides the master TO with a reference to a slave TO. Master and slave TOs can belong to different TO classes; the master is responsible for checking the class in order to perform the cooperation task. The master can process an information exchange or simply influence the behaviour of the slave; it can, for example, use the `discard()` method to "kill" the slave; the master can finally decide to remove itself from the system so that the slave can continue to live.

The current implementation of *Ecomobile* imposes the following assumption:

$$\boldsymbol{\tau_{\text{master}}\{\text{cooperate}(\tau_{\text{slave}})\} = \tau_{\text{slave}}\{\text{cooperate}(\tau_{\text{master}})\}}$$

This expression indicates that the cooperation methods of the master/slave TOs should be symmetric. However, this limitation is not too severe, since most cooperation methods aim at exchanging, merging or synchronizing activities between TOs belonging to the same family.

The cooperation mechanism is activated by the MBS; as can be seen on the state diagram, the `cooperate()` callback can be invoked when the TO is in the following states: *TO_S_READY*, *TO_S_ACTIVATED* and *TO_S_SUSPENDED_FOR_MIGRATION*.

The cloning behaviour - Φ_{clone} - invokes the method `clone()` of the TO. According to this method, all the frontal objects which are not "cloneable" by the *Java* `clone()`[1] method must be explicitly duplicated.

The different states are now examined in detail by means of *Specification and Description Language* (SDL) diagrams. For the sake of readability, and since the TO lifecycle depends on task-dependent implementation choice, we have used a simplified notation within the SDL diagrams, which is not always compliant with standard SDL semantics. A comprehensive overview of SDL can be found in [Hog89]. We now introduce three SDL macros (*Activate, BeforeMigration* and *Cooperate*) which will be used in the subsequent diagrams.

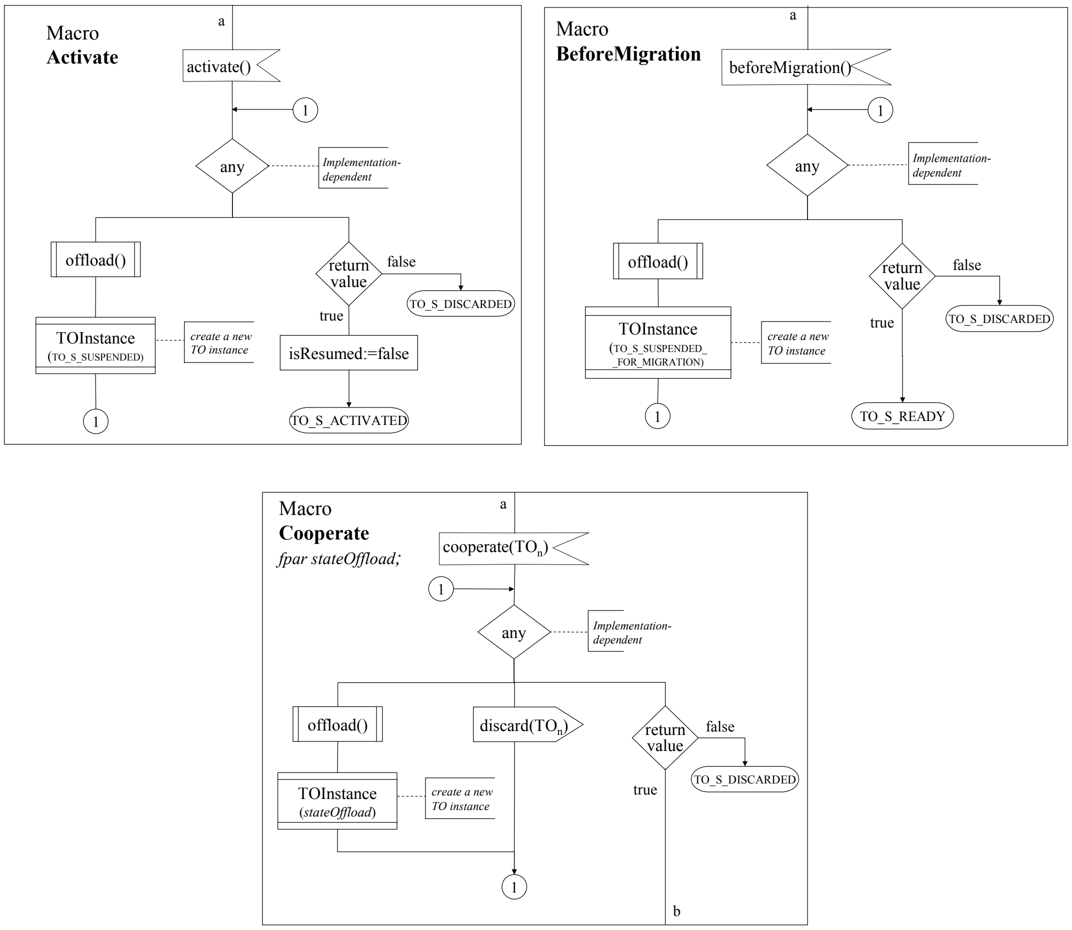


Figure 3-9. Macros definition in SDL

Each callback is followed by a task-dependent implementation choice. The selection of possible variants is represented within the diagram by means of the diamond shape; it is not part of the implementation. All activities following the callback entry point are performed during the callback execution. The method `offload()`, for example, can be used at any time during the callback execution.

[1] See the *Object* class in the Java API documentation (http://java.sun.com/j2se/1.3/docs/api/index.html).

TO_S_INIT

The initial state of a newly created TO instance actually depends on the originator; when the *user* manually deposits a TO in the blackboard, the TO enters state *TO_S_INIT*. When a TO instance is offloaded by the M-agent, a new instance of the TO is created in the state *TO_S_SUSPENDED* or *TO_S_SUSPENDED_FOR_MIGRATION* accordingly.

On the SDL diagram depicted on Figure 3-10, the formal parameter refers to the initial state of the TO.

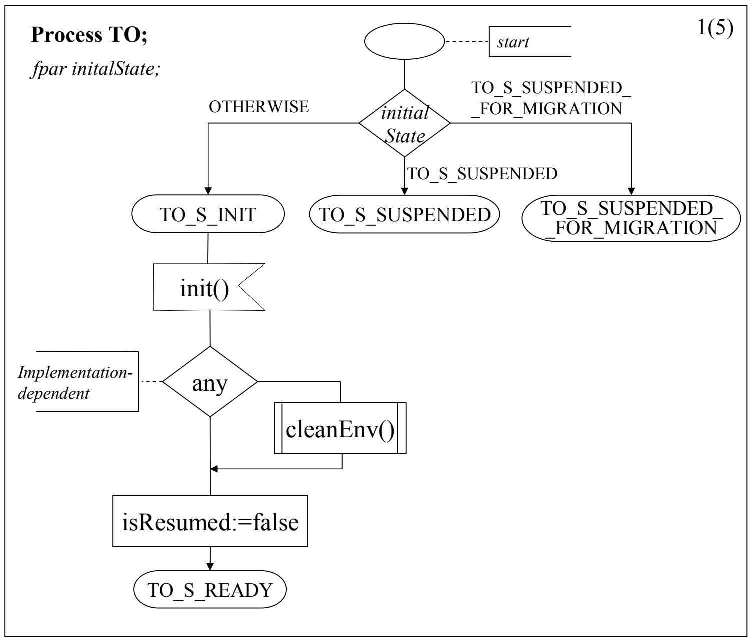


Figure 3-10. SDL Diagram from the state *TO_S_INIT*

When the TO has been loaded by a M-agent and instantiated in the agent context, the callback `init()` is called by the MBS and the TO enters state *TO_S_READY*.

In the `init()` method, the TO first decides via a `cleanEnv()` method whether the TO must be removed from the environment or not; if it must be removed, the TO is loaded only once (one shot loading) so that other M-agents subsequently visiting the place do not come across it any more; the TO dispersal within the network can also be improved by keeping the TO in its blackboard so that other M-agents can load it and disseminate it.

The TO persistence - enabled by the `setPersistent()` method - indicates if the task objective must temporarily be saved, in case the M-agent has no place in its context any more. Since the number of TO instances within the ecosystem may be sufficient to guarantee the TO's further existence, whether persistence is required or not depends on the task implementation.

In addition to these methods, the TO can set a priority - `setPriority()` - and perform initialization of frontal objects. During TO transfer phases, when the agent context size has reached its maximal limit, low-priority TOs will be temporarily offloaded first, while high-priority TOs will be preserved as much as possible in the agent context.

Since `setPersistent()` and `setPriority()` are optional and have to be considered at the same semantic level as `cleanEnv()`, these methods are not depicted on the SDL diagram.

TO_S_READY

TO_S_READY (Figure 3-11) indicates that the TO is ready to be activated by the Φ_{action} behaviour. The TO activation is performed at each node visit, via the method `activate()` (SDL macro *Activate*) and can not be interrupted by another M-agent.

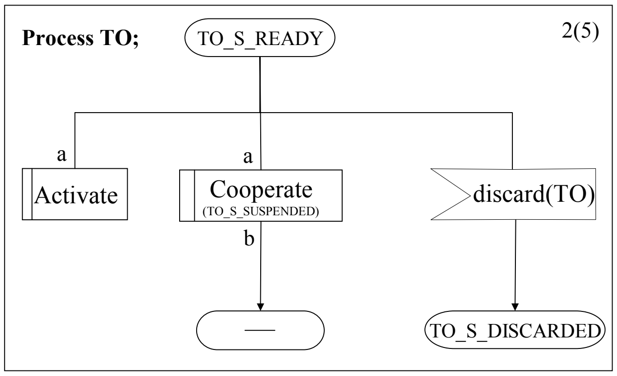


Figure 3-11. SDL Diagram from the state *TO_S_READY*

During the execution, a TO instance can decide to deposit a copy of itself into the environment by means of the user method `offload()`. The TO instance residing in the blackboard is then suspended until the next activation while the running instance remains active in the agent context and continues its work.

Once the TO execution is finished, it can decide to leave the M-agent by simply returning `false` at the end of the callback, in which case the M-agent removes the TO from its context; dissemination and cloning of this TO instance will consequently be stopped.

In this state, the TO can also be triggered by the `cooperate()` callback (SDL macro *Cooperate*); the master TO involved in the inter-TO cooperation mechanism can also decide to discard the TO.

TO_S_ACTIVATED

Once the TO has been executed, it enters *TO_S_ACTIVATED* (Figure 3-12) and is temporary suspended until the MBS induces a migration.

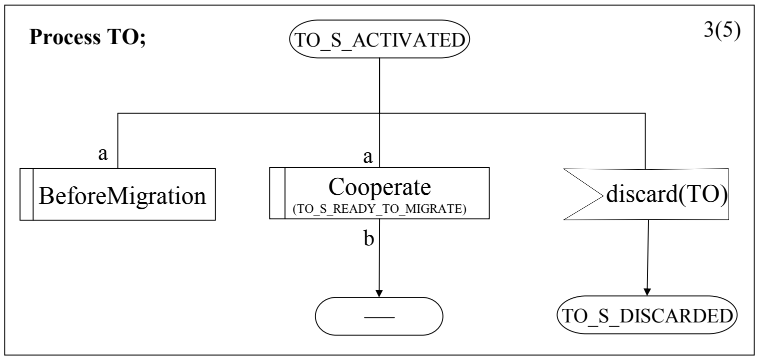


Figure 3-12. SDL Diagram from the state *TO_S_ACTIVATED*

The `beforeMigration()` callback allows the TO to decide whether it can remain alive in the M-agent and thus migrate with the agent, whether it has to be offloaded, or if it terminates. When the TO has migrated, it will immediately re-enter *TO_S_READY* for further activations.

As in *TO_S_READY*, the TO can be triggered by `cooperate()` or be discarded during the cooperation mechanism.

TO_S_SUSPENDED_FOR_MIGRATION

This state (Figure 3-13) is used to designate a task objective which has been offloaded just before migration: the TO is then loaded by another M-agent and can determine once more, via the `beforeMigration()` callback (SDL macro *BeforeMigration*), whether the next destination is appropriate or not.

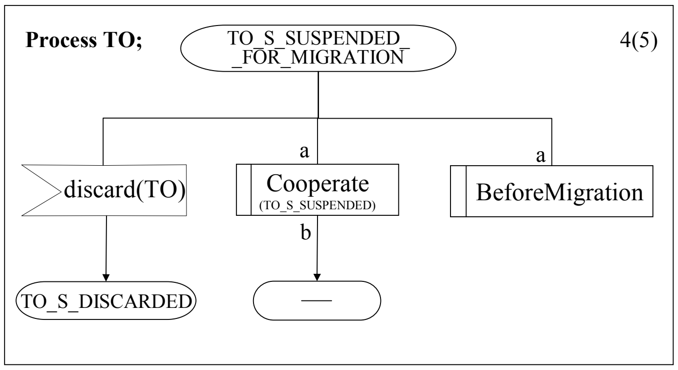


Figure 3-13. SDL Diagram from the state *TO_S_SUSPENDED_FOR_MIGRATION*

A TO can be loaded and offloaded cyclically until the destination matches its requirements.

The TO can also be triggered by `cooperate()` or be discarded during the inter-TO cooperation mechanism.

TO_S_SUSPENDED

A task objective is in the state *TO_S_SUSPENDED* (Figure 3-14) if and only if the TO itself has invoked the `offload()` callback.

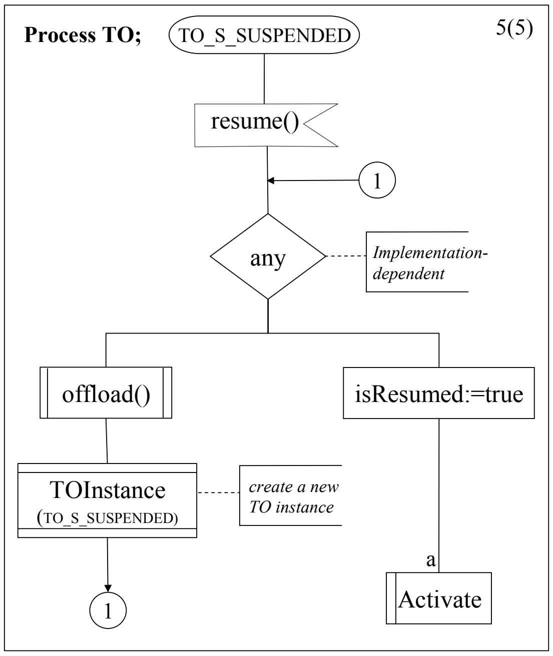


Figure 3-14. SDL Diagram from the state *TO_S_SUSPENDED*

The TO will be reactivated through the `resume()` callback in which the TO can decide to offload once more or to continue its execution.

TO_S_DISCARDED

When the TO is removed from the agent context and not offloaded, it enters the state *TO_S_DISCARDED*. The MBS then proceeds to its disposal. A TO is discarded when it terminates or it is discarded by another TO.

3.5.2 Rule-based Task Objectives

As we have seen, task objectives can easily be developed in *Java*. However, a object-oriented language does not necessarily suit all kinds of reasoning-based applications. A rule-based language would be more appropriate in order to achieve an alarm filtering, for example. When facts and rules are used in a common framework, the inter-TO cooperation makes a real "fusion" of the knowledge easier; internal mechanisms of the rule engine automatically lead to an adequate merging of facts and rules.

In [RSH01], we have experienced the development of TOs with a particular rule engine supplied by *Ilog*[1] and known as *JRules*, which provides an efficient implementation of the *Rete* algorithm [FC82] and facilities for manipulating *Java* objects directly in the rules, so that the objects of the agent context can be accessed without any particular problem. The corresponding rule-based language enables interactions with *Java* objects and supports most features of well-known rule engines such as CLIPS or JESS[2].

[1] http://www.ilog.com - has also provided a graphical library for the development of the *Generic Network Management Tool* (see Appendix A)
[2] http://www.ghg.net/clips and http://herzberg.ca.sandia.gov/jess

The interface between the *Java* based MBS and such a rule engine is achieved through the TO wrapper that maps the agent context operation onto the *JRules* concepts. The objects related to this rule engine are now briefly defined.

The *working memory*, in which the *facts* are stored, and the *agenda,* in which the valid rules are instantiated, are two important components used by the rules engine in *JRules*. The facts correspond to *Java* objects and are simply deposited into the working memory when the `assert` operation has been invoked. Once the inference engine has been activated, all the *valid* rules which are stored in the agenda are *fired*. In other words, when the conditions of a rule become true, an instance of the corresponding rule is left in the agenda and the rule is considered as valid; the `action` part of the rule will be executed at the activation time.

The general description of a rule can be expressed by the following statement:

rule ***ruleName*** { *when* { ... conditions ... } *then* { ... actions .... } }

Without going into details, we propose to describe a short example (Table 3-2) of a *JRule* TO storing the itinerary and terminating when a cycle has been detected. A more sophisticated TO for routing purposes in optical networks has been proposed in [RoRS01].

```
rule BuildItinerary {
  when {
    Activation();
  } then {
    assert NodeTuple(?context.wrapper.getAgency().getLocalNode().getAddr(),
                     ?context.prevAddr);
  }
};

rule DetectCycle {
  when {
    collect NodeTuple(addr equals
                     ?context.wrapper.getAgency().getLocalNode().getAddr())
    where (size() > 1);
  } then {
    ?context.wrapper.discard();
  }
};
```

Table 3-2. An example of a task objective in Ilog JRules

According to the first rule, the itinerary is stored in the working memory via the `assert` statement. The fact *`Activation()`* becomes true whenever the activation of the task objective is performed. The object called *`NodeTuple`* contains a node address and a reference to the previous visited node address so that it remains possible to keep track of the itinerary. The context referred by *`?context`* allows facts and rules to access to a space of variables associated to the working memory.

The second rule shows how a cycle can be detected when the itinerary has been built; it is activated right after the creation of new *`NodeTuple`* facts. In case of a cycle, the *`wrapper.discard()`* method is called so that the task objective including facts and rules disappears from the agent context.

3.5.3 Wrapper and Interactions with Task Objectives

The wrapper enables interactions between the mobile behaviour scheme and the TOs; it also provides the TOs with a reference to the local agency giving access to nodal objects, such as a rule engine or a blackboard, as well as to the local node environment. In addition, and when it is possible, the wrapper stores the next destination retrieved by the TO.

The wrapper is linked to a specific TO and therefore migrates with the M-agent. The wrapper functionalities consist in mapping the agent context operations onto a specific language and its related mechanisms, in order to manage the internal state of the task objective and to handle the storage mechanism by means of the blackboard; the storage mechanism allows a task objective to read or save a task objective properly, for example.

Figure 3-15 reveals two types of wrapper; the *Java* wrapper simply invokes corresponding callbacks within the TO, whereas the *Ilr* (*Ilog JRules*) wrapper contains specific frontal objects, such as a working memory used to store facts and rules related to the TO; this memory is also responsible for the activation of the external rule engine which has to be instantiated in the agency. A rule engine usually requires an important code size and should not be considered as mobile.

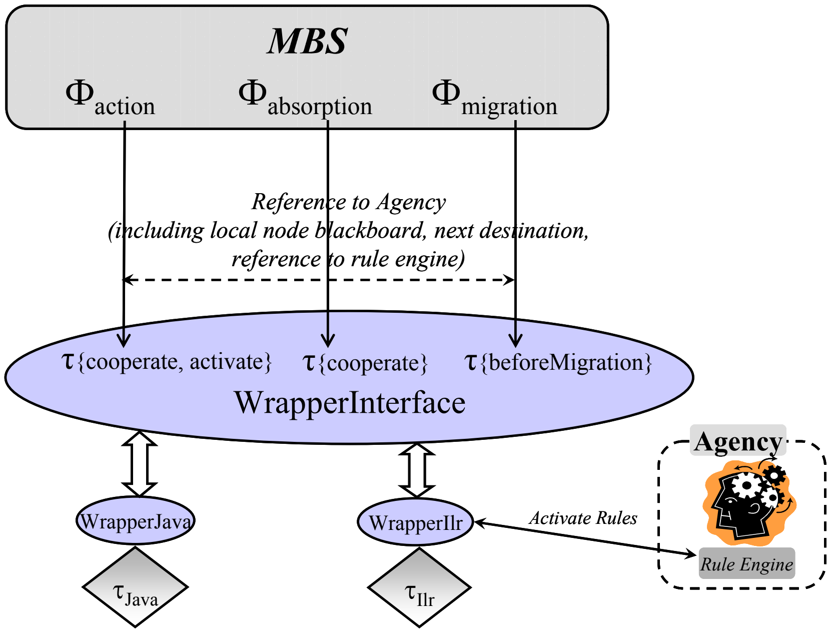


Figure 3-15. Interactions between MBS, task objective wrappers

According to the Figure 3-15, the mobile behaviour scheme interacts with the TO within the three Φ-behaviours (Φ_{action}, $\Phi_{absorption}$, $\Phi_{migration}$). The callbacks are activated by these behaviours and give the TO a reference to the wrapper containing the above-mentioned information.

Moreover, the cloning process requires still another interaction which consists in the original `clone()` method; in the wrapper and the TO, this operation does not correspond to any particular behaviour and must only contain statements for frontal object initialization necessary to achieve the cloning. We have actually experienced some difficulties designing a TO and controlling its propagation when considering conditional TO cloning based on internal decisions. Only the MBS should therefore drive the cloning of TOs. This is the reason why the `clone()` method is not considered as a TO callback.

As far as the `cooperate()` callback is concerned, the master/slave coupling may be performed by means of different TOs using different languages; a *Java* TO, for example, can be coupled to a *JRules* TO. In this case, an intermediate slave TO has to be created by the wrapper in the corresponding language of the master TO, so that the cooperation can be performed in a common language; this technique has not been implemented into *Ecomobile* yet.

3.6 SAMPLES OF GENERIC TASK OBJECTIVES

In this section, we present a couple of basic TOs which can be considered as TO models for network management tasks. These simple TOs have been designed according to the computational model of *Ecomobile* and make compositional building of complex task objectives possible.

These TO samples will show how it is possible to implement different operational behaviours mainly corresponding to the underlying functions of the different mobile MAS approaches exposed in Chapter 1 and Chapter 2; they can be used for delegation agents, *MITAgent* or in approaches based on emergent behaviour based approaches.

For simplification reasons, we assume that, in the task objectives we present, a single place is associated to a single node and that the *Universal Place Identifier* (UPI) only consists in the node address.

3.6.1 Travelling in a network

The basic task objective *TO_Travel* described in Table 3-3 builds a cycle-free itinerary in a continuous process; the TO lifecycle relies on the cooperation mechanism. When two task objectives meet, their itinerary is compared; the TO whose itinerary is contained in the other is simply discarded.

```
 1 public class TO_Travel extends TOJava {
 2
 3   private Vector itinerary;    // Itinerary as frontal object
 4
 5   public TO_Travel() { itinerary = new Vector(); }
 6   public Object clone() { … tells how to clone (copy of itinerary) }
 7
 8   public boolean activate(TOWrapperInterface wrapper) {
 9     if (itinerary.contains(wrapper.getAgency().getLocalNode().getAddr()))
10       itinerary.clear();
11
12     itinerary.add(wrapper.getAgency().getLocalNode().getAddr());
13     return true;
14   }
15
16   public boolean cooperate(TOWrapperInterface wrapper, TOJava otherTO) {
17     if (otherTO instanceof TO_Travel) {
18       TO_Travel _other = (TO_Travel) otherTO;
19
20       if (itinerary.contains(_other.itinerary))
21         _other.discard();
22       else if (_other.itinerary.contains(itinerary))
23         return false;
24     }
25   }
26
27   public void init(TOWrapperInterface wrapper) {
28     cleanEnv(wrapper);    // One shot loading
29     setPriority(10);      // Maximum priority
30     setPersistent(true);  // In case of overloaded AC, TO is offloaded in env.
31   }
32 }
```

Table 3-3. A simple TO model to travel in the network

All the frontal objects are placed in the declarative part of the class. In *TO_Travel*, only the itinerary (3) has to be maintained by the TO itself. In `init()` (27), the task objective removes itself from the environment by means of the `cleanEnv()` method (28), so that the TO is loaded only once. Maximum priority (29) is not relevant in this context, but persistence (30) indicates that the TO must be saved in the blackboard when the agent context is full.

The `cooperate()` callback (16) eliminates the TOs which have the same knowledge (17-24) that is, the same itinerary; this strategy avoids exponential growth of the agent context. The cooperation mechanism still plays an important role in the self-regulation of the average agent context size.

This task objective relies on a stochastic navigation model since its trajectory only depends on the M-agent's movement and therefore on the network environment; in the TO's callbacks, there is no migration-related decision.

A behavioural simulation of this task objective is presented in Section 5.4.1.

From now on, only the relevant callbacks are described; the constructor and `clone()` methods are not depicted any more.

3.6.2 Monitoring

The monitoring function, which handles fault and performance management, naturally constitutes a major function in network management. The task objective presented in Table 3-4 is dedicated in particular to monitoring tasks examining the node periodically; it does not have to be plugged into the device.

Each monitoring task must be defined as a different *`TO_Monitor`* class.

```
 1 public class TO_Monitor extends TOJava {
 2
 3  public boolean activate(TOWrapperInterface wrapper) {
 4     // Monitoring function
 5     // …
 6     return true;
 7   }
 8
 9   public boolean cooperate(TOWrapperInterface wrapper, TOJava otherTO) {
10     if (otherTO instanceof TO_Monitor)
11       otherTO.discard();
12
13     return true;
14   }
15 }
```

Table 3-4. A generic monitoring TO model

In this task objective, the `cooperate()` method kills the slave TO regardless of its internal knowledge (10-11). The particular monitoring function must be implemented in `activate()`.

3.6.3 The Node Inspector

The node inspector task objective consists in implementing a particular task, or *service*, into the node component; the service is activated during M-agent visits. Unlike *TO_Monitor*, this task objective is offloaded at each place-related blackboard; therefore, the task does not travel along the links but continues to reside within the node, so that it can maintain the knowledge specific to each node.

When a node fails, the links with the neighbouring nodes consequently fail and the place connectivity matrix is updated; once the node has been re-installed and the *Ecomobile* agency has been informed about the inter-place connectivity, the task objective of connected places will be re-installed automatically.

This task objective can be used in the context of TOs' operational management, for example, such as garbage collection functions or TO class management (see Section 4.4.1). Table 3-5 shows the details of *TO_NodeInspector*.

```
 1 public class TO_NodeInspector extends TOJava {
 2
 3   // Frontal objects
 4   private boolean resident = false;
 5
 6   public boolean activate(TOWrapperInterface wrapper) {
 7     wrapper.getAgency().getBlackboard().theContent().put("TO_Inspector", "ok");
 8     resident = true;
 9
10     // Execute specific task ...
11
12     offload(wrapper);  // Save the resident TO - can fall asleep again :-)
13     resident = false;  // For subsequent diffusion
14     return true;       // Keep living for investigation
15   }
16
17   // Ensure only one TO in the agent context
18   public boolean cooperate(TOWrapperInterface wrapper, TOJava otherTO) {
19     if (otherTO instanceof TO_NodeInspector) {
20       if (((TO_NodeInspector) otherTO).resident)
21         return false;
22
23       otherTO.discard();
24     }
25     return true;  // Still alive
26   }
27
28    // Will check if the output links must be investigated
29   public boolean beforeMigration(TOWrapperInterface wrapper) {
30     if (wrapper.getAgency().getBlackboard().content().containsKey(wrapper.destination())
31       return false;
32
33     wrapper.getAgency().getBlackboard().content().put(wrapper.destination()(), "ok");
34     // Re-init internal knowledge if necessary …
35     return true;
36   }
37
38   public void resume(TOWrapperInterface wrapper) {
39     cleanEnv(wrapper);  // Ensure the presence of only one TO
40   }
41 }
```

Table 3-5. A node inspector TO model

In `activate()` (6), the task objective is offloaded (12) after its specific processing; in order to avoid having multiple instances of the TO in the blackboard, the `resume()` callback (38), which is invoked when the TO has been suspended and re-loaded by another M-agent, performs a `cleanEnv()` (39) in order to remove the TO from the blackboard. For each place, there is a single instance of *TO_NodeInspector*. The cooperation mechanism (19-25) avoids the redundancy between several TOs; the `beforeMigration()` callback controls the propagation of the task objective towards the neighbouring nodes (30-35).

In order to determine whether each place has been investigated or not, a particular nodal object (34) is deposited into the blackboard; this indicator allows the task objective to be installed pro-actively when a new connection has been set up.

3.6.4 Path Selection

While the previous task objectives rely on a stochastic navigation model implemented by the mobile behaviour scheme, the TO model which we are about to examine must have a pre-planned itinerary. In this particular model, the path corresponds to a list of nodes initialized in the TO without intermediate nodes. Operational behaviours which are similar to delegation agents can be implemented by means of this TO model.

```
 1 public class TO_PathSelect extends TOJava {
 2
 3   // Frontal objects
 4   private Vector itinerary;          // Internal trajectory
 5   private Vector path;               // Path to follow
 6
 7   public boolean activate(TOWrapperInterface wrapper) {
 8     GenericNode currentNode = wrapper.getAgency().getLocalNode();
 9
10     // Check if the node belongs to the path
11     if (currentNode.getAddr().equals((String) path.firstElement())) {
12
13       path.removeElementAt(0);        // Prepare the next node to search
14
15       if (path.isEmpty()) {
16         // Do something useful with the path nodes …
17         return false;   // Finished
18       }
19     }
20
21     itinerary.add(currentNode.getAddr());  // Store the current location
22     offload(wrapper);                      // Offload for subsequent destinations
23     return true;                           // Keep alive
24   }
25
26   public boolean beforeMigration(TOWrapperInterface wrapper) {
27     // Disappear only if the next destination is not appropriate
28     if (!wrapper.destination().equals((String) path.firstElement()))
29       return false;
30
31     cleanEnv(wrapper);    // We continue our trip; it is not necessary to stay here
32     return true;
33   }
34
35   public boolean cooperate(TOWrapperInterface wrapper, TOJava otherTO) {
36     // Check for only one instance, such as in TO_Monitor …
37   }
38
39   public void init(TOWrapperInterfaceInterface wrapper) {
40     // …
41     // Here is the pre-defined path
42     path.add("Geneve"); path.add("Lausanne"); path.add("Bulle");
43     path.add("Fribourg"); path.add("Bern"); path.add("Basel");
44     // …
45   }
46 }
```

Figure 3-16. A pre-planned navigation TO model

TO_PathSelect resorts to the following principle: the TO is offloaded (22) and remains resident, whenever its destination does not correspond to the one expected (28-29) according to the path description (42-43) defined in *`init()`*; this simple mechanism allows the TO to progress node by node towards its destination; the progression is controlled by the *`beforeMigration()`* callback (26). The visit frequency and the migration strategy obviously have an impact on the performance attained in this approach; a behavioural simulation is presented in Section 5.4.2.

3.6.5 The Exhaustive Path Finder

The exhaustive path finder task objective, which is a bit more complex than the above-studied TOs, enables an exhaustive search for all cycle-free paths in a network from one source node to a destination node. This TO gives the opportunity to look for alternative paths leading to better Quality-of-Service; this task objective, however, may induce considerable execution delay due to the exponential growth of the paths number related to the size of the network topology; an adequate problem-related heuristics should be implemented into the cooperation mechanism so that the number of paths can be reduced; such an approach will be presented in the context of *TO_Routing* (see Section 3.6.6).

```
public class TO_ExhaustivePathFinder extends TOJava {

  // Frontal objects
  private Vector itinerary;          // Internal trajectory
  private Vector path;               // Path to follow
  private String src, dest;          // Source and destination
  private String _id;                // Allows the TO instance to be identified

  public boolean activate(TOWrapperInterface wrapper) {
    GenericNode currentNode = wrapper.getAgency().getLocalNode();

    if (isResumed()) return true;

    if (currentNode.getAddr().equals(dest)) {
      itinerary.add(currentNode.getAddr());

      // Do something useful with the discovered path (we have reached the dest.)
      return false;
    }

    itinerary.add(currentNode.getAddr());

    Vector _outs = getAgency().getPlacesOut();  // Get the UPIs of connected places
    Vector toExplore = new Vector();

    for (Iterator _i = _outs.iterator(); _i.hasNext(); ) {
      String addr = ((Port) _i.next()).getPeerNode().getAddr();

      if (!itinerary.contains(addr)) toExplore.add(addr); // Detect a cycle
    }

    if (toExplore.isEmpty()) return false;  // No interest to go further

    _id = this.toString();  // _id will be the same for future clones
    wrapper.getAgency().getBlackboard().theContent().put(_id, toExplore);
    offload(wrapper);

    return true;
  }

  public boolean beforeMigration(TOWrapperInterface wrapper) {
    // This is a particular nodal object used by this TO
    Vector toExplore = (Vector) wrapper.getAgency().getBlackboard().theContent().get(_id);

    // Still some destination to explore?
    if (toExplore == null) {  cleanEnv(wrapper);  return false; }

    // Next destination is not visited yet?
    if (!toExplore.contains(wrapper.getNextDest().getDestination()))
      return false; // This destination is not interesting for me.

    // Update the vector of possible destination
    toExplore.remove(wrapper.getNextDest().getDestination());
    if (toExplore.isEmpty()) {
      wrapper.getAgency().getBlackboard().theContent().remove(_id);
      cleanEnv(wrapper);
    }
    return true;
  }

  public void init(TOWrapperInterface wrapper) {
    // …
    src = "Geneve"; dest = "Lugano";
  }
}
```

Table 3-6. TO model for an exhaustive path finder in a network

This task objective aims at investigating all the output links of each node; the corresponding investigation algorithm resorts to a specific nodal object `toExplore` (24, 43) containing a list of node addresses still remaining to be explored and identified by a TO instance identifier `_id` (34): once the TO decides to select a specific link and continues its exploration, the corresponding peer node address is removed from its list (55).

Each task objective must keep track of its itinerary in order to extract all the paths; as suggested in Table 3-6, this TO does not resort to any cooperation mechanism: each new instance of the TO generated towards a particular direction must survive until the destination has been reached.

In Chapter 2, we have seen that the migration strategy in the stochastic navigation model relies on environmental information, which facilitates the implementation of loosely coupled tasks. The task objective *TO_ExhaustivePathFinder* perfectly illustrates this model; the destinations to be investigated have to be stored outside the task objective and therefore outside the agent. If this information were stored as frontal object, the MBS cloning mechanism would imply a duplication of the destination list. As the task objective can not influence the agent migration, it would be difficult to synchronize the knowledge of all the TO instances in order to avoid multiple investigations of a same link. As we have already mentioned, however, the TOs must have distinct trajectories. In order to control the migration information concerning the task objective, we store a reference to the TO as a nodal object associated to its list of remaining destinations. This data structure is built in the *`activate()`* callback and updated in the *`beforeMigration()`* callback. The structure is removed from the blackboard when all the destinations have been investigated. A behavioural analysis of this task objective is presented in Section 5.4.3.

3.6.6 On-line Routing

On-line routing in transport networks is a basic function which is used for a wide range of management functions: allocating a path, restoring a service via an alternative path, pre-computing protection paths, monitoring connections while trying to optimize the QoS, etc.

This task objective is devoted to a routing function updating the routing tables continuously according to the TO exploration. As we have seen in the *MITAgent* approach in Section 1.3.1, resorting to the cooperation mechanism in order to exchange routing information considerably improves the performance of this function. The routing mechanism implemented into the task objective described in Table 3-7 consists in maintaining a routing table describing all the destinations which can be reached from the current node. In the TO model, link cost and distance are not taken into account. Additional information related to routing cost can be added as frontal object; it can then be compared to the local information present in the routing table. Such a TO may be used to implement wavelength assignment algorithms, for example (see Chapter 7); wavelength connectivity is intrinsically guaranteed by the task objectives' migration over the wavelength within the network.

```
 1 public class TO_Routing extends TOJava {
 2
 3   // Internal knowledge
 4   private Vector itinerary;
 5   private int maxItiLength;
 6
 7   public boolean activate(TOWrapperInterface wrapper) {
 8     GenericNode currentNode = wrapper.getAgency().getLocalNode();
 9
10     // Retrieve the port (top-level) by which the TO has entered the node
11     Port issuingPort = (Port) wrapper.medium();
12
13      // If the itinerary is not empty, do update
14      if (itinerary.size() > 0) {
15
16        // First, process backwarding nodes if bi-directional links by
17        // updating with visited nodes
18        for (int i = itinerary.size()-1; i >= 0; i--) {
19
20          String addr = (String) itinerary.elementAt(i);   // Node address
21          if (!(currentNode.getAddr().equals(addr)))
22            // Update the internal table associated to the port issuingPort
23        }
24
25        // Then, process forwarding nodes, if any
26        boolean cont = true;
27        for (int i = itinerary.size()-1; (cont && (i >= 0)); i--)
28
29          if (((String) itinerary.elementAt(i)).equals(currentNode.getAddr())) {
30
31            // Find the right port
32            Port forwardingPort = currentNode.getPort((String) itinerary.elementAt(i+1));
33
34            for (int j = i+1; j < itinerary.size(); j++) {
35              String addr = (String) itinerary.elementAt(j);
36
37              // Update the internal table associated to this port …
38            }
39            cont = false;
40          }
41      }
42
43      // Check the size of the itinerary - a way to date the TO
44      if (itinerary.size() > maxItiLength) itinerary.clear();
45
46      itinerary.add(currentNode.getAddr());         // Store the current location
47      return true;
48    }
49
50    // Cooperation between Task Objectives
51    public boolean cooperate(TOWrapperInterface wrapper, TOJava otherTO) {
52      if (otherTO instanceof TO_Routing) {
53        TO_Routing _other = (TO_Routing) otherTO;
54
55        // Compare the itinerary knowledge
56        if (itinerary.containsAll(_other.itinerary))
57          other.discard();
58        else if (_other.itinerary.containsAll(itinerary))
59          return false;
60      }
61      return true;
62    }
63
64    public void init(TOWrapperInterface wrapper) {
65      // …
66      maxItiLength = wrapper.getAgency().getLocalNode().numberNodes() / 2;
67    }
68  }
```

Table 3-7. On-line routing TO model

At each node, there are two ways (16, 25) to update the routing table as it is depicted in the *`activate()`* callback. Since the TO uses the physical links to migrate, the itinerary stored in the TO directly reflects the reverse node connectivity when the links are bi-directional; when the links are uni-directional, updating a routing table can be achieved only in case of a cycle, as described in the second part of *`activate()`*.

The cooperation between the TOs consists in absorbing the knowledge of the task objective owning the smaller quantity of node addresses (56-59). In order to limit the itinerary length, the itinerary is reset when a certain number of addresses have been collected (44); the number of network nodes divided by two gives an empiric estimation of the maximal itinerary size (66).

3.7 OVERVIEW OF THE INTERACTIONS IN *ECOMOBILE*

Figure 3-17 presents a summary of all the interactions involved in *Ecomobile*. The components considered in this context are the agency (FIPA Agency), the blackboard, the M-agent and the task objectives.

AGENT-TO-AGENT INTERACTION

This kind of interaction between two M-agents is managed by the MBS, which involves the $\Phi_{intereference}$, $\Phi_{dwelling}$ and $\Phi_{absorption}$ behaviours. Since mobile agents share an identical execution environment controlled by the agency, the two entities communicate via object referencing. The underlying synchronization mechanism relies on cooperative synchronous processes and will be explained in detail in Chapter 4. Agent-to-agent interaction implies interaction between the task objectives.

TO-TO-TO INTERACTION

Interactions between task objectives are managed in the agent context of the M-agent and involve the coupling of TOs according to a master/slave paradigm. The master TO obtains a reference to the slave TO and handles the knowledge transfer, after which the slave TO can be discarded.

These interactions occur when new task objectives have been loaded into the agent context or, in case of successful interaction, when the knowledge has been transferred, the knowledge here referring to the whole set of task objectives.

INTRA-AGENT INTERACTION

Two components of the M-agent are involved in this interaction: the agent context on the one hand and the task objective wrapper on the other hand. The former invokes the methods of the wrapper which, in turn, activates the TO callbacks. The M-agent also maintains a reference to itself within the TO wrapper so that the TO can access the information relative to its current location as well as the agency, in order to deal with the local environment.

AGENCY-TO-AGENT INTERACTION

A reference to the agency is maintained within the M-agent so that the MBS can interact with the agency via specific methods

The task objective interacts with the agency through its wrapper, which, as we have already shown, contains a reference to the M-agent.

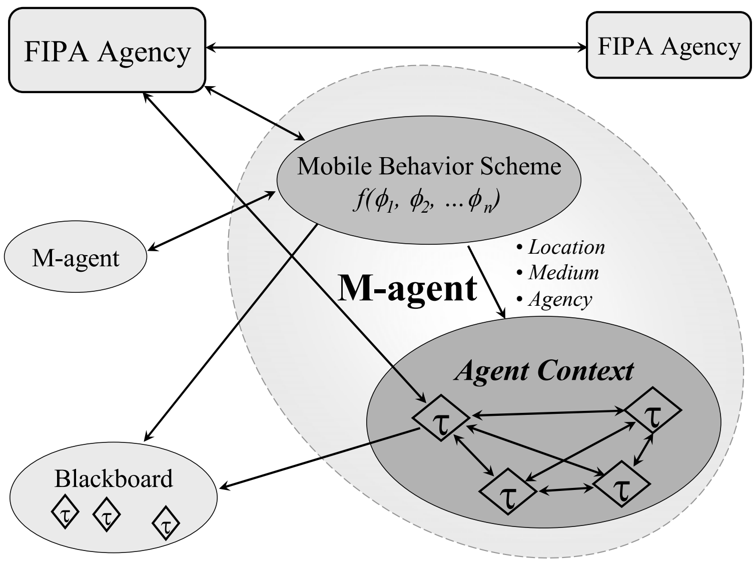


Figure 3-17. Interactions in *Ecomobile*

AGENT-TO-BLACKBOARD INTERACTION

The MBS loads or offloads the task objectives into the blackboard associated to the current place; a task objective itself has no control over serialized TOs present in the blackboard. In order to reach its goal, the TO can however access the blackboard via the agency to manage nodal objects like those required to manage information related to a chemical trail, for example, needed for emergent behaviour based algorithms (see Section 2.4).

AGENCY-TO-AGENCY INTERACTION

This kind of interaction has not been discussed yet because it is related to the specific implementation of *Ecomobile* which will be detailed in Section 4.4. Agency-to-agency interactions allow the M-agents to move between the network nodes. The agency, which provides the M-agents' execution environment, is a FIPA-compliant agent, so that mobility is achieved through the exchange of ACL messages between agencies.

3.8 SUMMARY

In this chapter, we have presented the conceptual framework of *Ecomobile*. The identification of the three abstraction models presented in Chapter 2 have allowed us to propose an agent architecture based on the *Mobile Behaviour Scheme* (MBS) including the navigation and the coordination models, and the *Task Objective* (TO) related to the computational model. According to this model, mobile agents and task objectives have distinct lifecycles and different trajectories in the network.

The implementation model of *Ecomobile* corresponds to a threefold architecture made up of the following active components: agency, mobile agent and task objective.

In *Ecomobile*, the place is used as a location concept and a coordination space for the M-agents. Several places can be defined and interconnected within an agency in order to form a virtual network. Intra- and inter-agency connectivity enable both virtual and physical mobility. The information related to the connectivity is kept in the agency via the *Universal Place Identifier* (UPI) and the connectivity matrix which, in this context, reflects the physical network topology. Finally, a passive blackboard is associated to each place and provides the M-agents with a shared repository for TO-related information and for the task objectives themselves.

The mobile agent society is composed of mobile entities called M-agents which act as ecological individuals exhibiting particular behaviours called Φ-behaviours. The territoriality paradigm, which refers to a density-dependent intra-specific competition based on active interference between ecological individuals, plays a central role in the self-regulation of the M-agent population and allows us to implement a "living" ecosystem-inspired mobile agent middleware into the network infrastructure. Two different MBS have been proposed: *MBS-low* implements a simple interference-absorption scheme between two M-agents meeting at a place, and leads to occasional cloning; the diffusion of mobile agents is considered as relatively "slow". On the contrary, *MBS-high* implements an interference-absorption loop favouring a greater number of meeting opportunities, and followed by systematic M-agent cloning; the diffusion of M-agents is therefore considered as relatively "fast".

While the MBS defines the M-agent's lifecycle within the network infrastructure, so that the ecosystem can adapt itself to network characteristics such as network topology, availability, quality of service, etc., and maintains a density-dependent self-regulated population size, the M-agent's operational behaviour is defined by intelligent tasks relying on cooperation mechanisms achieved by the task objectives. The TOs are deposited into a blackboard before they are dynamically loaded into the ecosystem by the M-agents. The task objective wrapper makes task objectives flexible enough to be described by means of different programming approaches, such as *Java*, or a rule-based language like *Ilog JRules* or JESS, or even *Wave*[1]. The particular computational model characterizing *Ecomobile* leads to a task design based on specific callbacks for the TO activation, migration and cooperation, on the associated current location and the next destination. These callbacks are regularly activated by the MBS.

Generic operational behaviours have finally been described by means of task objective models; they correspond to basic functions defined for the management of transport networks and can be used for the compositional construction of more advanced network management functions.

In order to take a further step in the development of *Ecomobile*, we have to implement the concepts presented in this chapter into an reactive execution environment, so that M-agents and task objectives can evolve and interact properly, with respect to the ecosystem behaviour defined in this chapter. The reactive programming paradigm introduced in Chapter 4 will help us to reach our goal.

[1] Since to TOs are loaded dynamically, any interpreted language is perfectly suitable.

Chapter 4
Implementation with Reactive Programming and Deployment

The *Ecomobile* model developed in the previous chapter requires an adequate implementation so that the ecosystem behaviour can be finely simulated and analyzed. The response of our system to the dynamic insertion of task objectives will allow us to evaluate the system performance; it will also provide indicators related to the efficiency of the inter-TO cooperation, for example, or to the rapidity of their dissemination within the network.

In an initial approach, *Ecomobile* has been implemented by means of *Java* threads. Although a mapping between the two concepts, threads as concurrent processes and mobile agents as autonomous entities, appeared to be natural, this approach induces a number of problems: the framework for the management of cooperative threads provided by the *Java Virtual Machine* (JVM) is not complete enough; components such as *semaphore* or *rendezvous* are not supported and require a third-party library for concurrent programming[1] [Lea00]. *Java* threads moreover rely on the JVM scheduling policy, which is not standardized, so that in this context the *hotspot* JVM of the *Java Development Kit* (JDK) resorts to the scheduling policy of the underlying operating system.

The considerable overhead required by the JVM to manage the threads and the pre-emptive scheduling policy would introduce undesirable dependencies on external factors, such as the type of the operating system, the CPU power of the machine running the simulation, or the memory size, and thus lead to a huge number of asynchronous interactions and non-deterministic effects. Therefore, although *Java* threads would allow a relative straightforward implementation of mobile agents with the help of an adequate third-party library, this approach is not very suitable for the complex analysis of mobile agent behaviour.

In order to deal with these problems, we have opted for an approach based on a cooperative process-based discrete-event implementation which actually seems more promising. Since *Ecomobile* can be considered as a reactive system, we propose to adopt a reactive programming paradigm to implement the M-agent behaviour. In this chapter, we will discover how reactive programming and an associated framework called *Junior* can lead to the efficient implementation of the mobile behaviour scheme of *Ecomobile*.

As we have already pointed out in Chapter 3, the introduction of our simulated ecosystem into a transport network environment should lead to a realistic deployment of the *Ecomobile* components with particular attention to the non-deterministic effects due to multiple asynchronous interactions; still, the parallel behaviour which is inherent to the ecosystem, and in particular the asynchronous agent migration, should be maintained; in order to deal with these issues, we propose on the one hand to limit the asynchronous interactions in *Ecomobile* to the inter-agency communication, and to implement the Φ-behaviours of the different M-agents by means of a particular parallel construct on the other hand.

In the present chapter, we wish to show that the agency services can be delegated to a FIPA agent in the *Jade* environment. The efficiency and scalability issues related to this approach will also be discussed.

[1] See also http://gee.cs.oswego.edu/dl/cpj

4.1 Reactive Programming

The implementation of reactive systems [HP85][Bo00] by means of the reactive programming paradigm requires the introduction of new abstractions and new programming language components in order to be implemented. Reactive systems combine two essential characteristics: interacting permanently with their environment, they exhibit a cyclic behaviour and never terminate, but they should also be fast enough to timely react to changes induced by the environment. After being activated by the environment itself, a reactive system should therefore produce a corresponding reaction possibly modifying the environment, and then wait for subsequent activations. It appears from this definition that a reactive system can be decomposed into distinct steps, so that an implementation based on a synchronous model should be possible.

Two fundamental notions defined in the scope of the reactive approach constitute the underlying mechanism addressing the synchronous concurrency: *instant* and *reaction*. An *instant* is defined as a logical instant of execution during which all the parallel components of a program perform one execution step that is, a *reaction* corresponding to that instant. The duration of an instant may vary from one reaction to the other; still, the environment must remain in a coherent state during the reaction, which means that an event considered as present during the reaction must be kept present until the end of the reaction.

The succession of instants constitutes an efficient mechanism leading to the implementation of a time reference model which primary behaviours such as Φ_{dwelling}, for example, can resort to; corresponding to a fixed number of logical instants, the waiting time can remain identical even though the duration of a reaction may vary over the time. Such a time reference is therefore particularly useful while elaborating and debugging complex multi-agent system behaviour.

As shown on Figure 4-1, a new instant begins at each activation time.

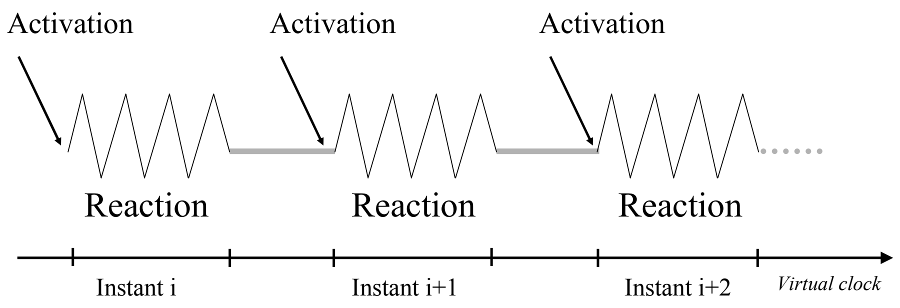


Figure 4-1. Instant and reaction in the reactive model

A reactive program is decomposed into reactive instructions which can be executed within an instant. The concurrency model leads several reactive instructions belonging to different programs to share the same execution instant, so that instructions are interleaved and executed in a determined (sequential) order or an undefined (parallel) order. The *events* which are simply managed by reactive instructions enable inter-process communication. An event constitutes non persistent data; it can be generated within an instant but is automatically reset at the beginning of the next instant.

The definition of instants and events is bound to a general problem related to the parallel execution of reactive instructions in a synchronous concurrency model known as the *causality problem* [Hal93]. A causality problem may arise when parallel instructions are executed synchronously within the same instant: an instruction instantaneously reacts to the *absence* of an event which is actually being generated in the same instant by another instruction; within the same instant, reactive instructions may consequently consider an event to be present or not present so that the system is placed in an incoherent state and favours non-deterministic behaviour. Meanwhile, reaction to the presence of an event does not raise any ambiguity.

The causality problem can be overcome by means of *delayed reaction to absence* [BHS01]; the instruction testing for the absence of an event is postponed to the next instant so that it can make sure that the event is really absent; the strong coherence property, stating that during one instant, the same event cannot be tested as present by one component and as absent by another component, thus remains guaranteed.

Several languages based on the reactive model are currently available, such as *Esterel*[1], *SL* or *Reactive-C*[2], for example. We now propose to describe the *Junior* framework which provides an excellent lightweight *Java*-based API for reactive programming.

4.2 THE JUNIOR FRAMEWORK

Junior (*Jr*), which has been developed in the scope of the MIMOSA project[3] at *INRIA*[4], provides a *Java* API based on formal semantics for reactive programming; its reasonable code size makes it a micro-kernel, on top of which several extensions have been elaborated, such as the *SugarCubes* [BS02], a *Java* library providing extensions for the experiments on various reactive formalisms. An implementation of *Junior* called *Senior* has been developed for the *Scheme* programming language.

Junior implements a *reactive machine* managing the execution of event-based reactive programs; reactive instructions use events to synchronize their execution in a cooperative way. The basic assumption in *Junior* is that the execution during an instant always converges on a stable state from which the next instant can start, so that no reactive instruction should enter an infinite loop[5].

In order to guarantee the strong coherence property described in the previous section and thus to solve the causality problem, *Junior* implements instantaneous reaction to presence and delayed reaction to absence; each event is instantaneously broadcast within an instant, so that processes concerned with this event can react immediately.

Several implementations of *Junior* have been developed[6]: *Rewrite*, which is the first implementation of the Junior semantics [HSB99], is not as efficient as *Replace*, which avoids program re-buildings; *Replace* is close to *SugarCubes*.

Let us now introduce the reactive machine and the reactive instructions, which are the basic components of *Junior*.

[1] http://www-sop.inria.fr/meije/esterel/esterel-eng.html
[2] Information about SL and Reactive-C available at http://www-sop.inria.fr/mimosa/rp/ReactiveC
[3] More information about MIMOSA (*Migration & Mobility: Semantics & Applications*) available at http://www-sop.inria.fr/mimosa/rp
[4] *Institut National de Recherche en Informatique et en Automatique*, Sofia Antipolis, France
[5] In the present implementation of Junior, there is no automatic detection of possible ill-formed reactive instructions.
[6] In *Ecomobile*, we have considered *Replace*, version 2.1b1 (experimental).

4.2.1 The Reactive Machine

The reactive machine provides the program with an execution environment and defines the global instants during which reactive instructions are executed; it is also responsible for broadcasting the events when they have been generated. A program is added in the reactive machine via the instruction `add()`, and the reaction is activated externally with the `react()` method, as depicted on Figure 4-2. Reactive instructions can be added once at the beginning of the program execution, or they can be added dynamically during the execution; they can also generate new instructions to be added into the reactive machine, for example.

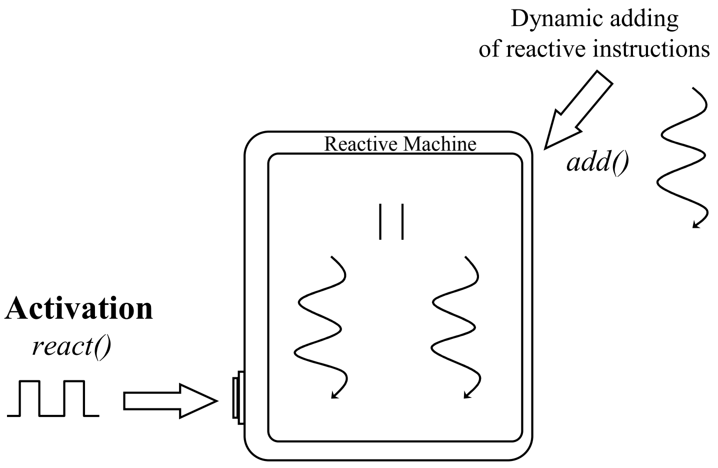


Figure 4-2. Reactive machine in Junior

Reactive instructions are not re-entrant; their state is embedded and fully controlled by the reactive machine. In *Junior*, two kinds of reactive machines have been developed: *Machine* and *SafeMachine*. Whereas the former is intended to be run in a single thread, the latter can be used when several threads are involved in the execution of the reactive machine; a *Java applet*, for example, can manage the graphical interactions by means of several threads and therefore provoke uncontrolled interferences with the program execution.

```java
public boolean react() {

  performAddings();                // Check for new added instructions
  if (terminated == false) {       // No instruction available
    byte res = instant.rewrite(); // Perform instructions
    env.newInstant();              // Prepare the next instant

    if (res == TERM)
      instant.body = new Instant(Jr.Nothing()); // Reset the program

    terminated = (TERM == res);    // Program finished?
  }
  return terminated;
}
```

Table 4-1. The method `react()` of the reactive machine (*MachineImpl.java*) activating a reaction

The method `react()` detailed in Table 4-1 consists in performing the activation of all reactive instructions which have been prepared for the present instant.

In the current version of *Junior Replace* v2.1, the reactive instructions attached to the object *instant* are unfortunately not removed from the stack when they have been completed; instructions are added in *Junior* according to a recursive scheme. In *Ecomobile*, for example, we have observed that the dynamic addition of instructions raises a stack overflow problem after only a few instants. This is the reason why we have added two lines in the `react()` method (lines 8 and 9) in Table 4-1 - in order to "clean" the instant object containing the program once all the reactive instructions belonging to an instant have been executed; in this case, the value returned by `instant.rewrite()` is *`TERM`*. It has to be noted that if a reactive instruction requires several instants, in particular when implementing a delay, the instant can not be cleaned and overflow problems can still happen. The authors of *Junior*, however, are developing a new version of *Junior*, called *Storm*, which will improve the processing of reactive instructions so that a huge amount of events and instructions will be supported.

4.2.2 The Reactive Instructions

In this section, we are trying to examine whether the *Junior* reactive instructions are fulfilling the requirements for the implementation of *Ecomobile* according to the reactive programming paradigm. In the *Junior* framework, all the instructions are defined statically in a class named `Jre`. The instructions themselves are defined by means of *Java* classes. Since a *Junior* program can be regarded as a reactive instruction itself, all the instructions inherit from the class `Program`. In order to improve readability, we have omitted to prefix each instruction with `Jre` in the next code fragments.

(1) *`Seq(Program first, Program second)`*

This instruction defines two reactive instructions which are executed sequentially and always in the same order (first, second). It has to be noted that each instruction may require several instants to accomplish its execution.

(2) *`Par(Program first, Program second)`*

Unlike the previous instruction (1), the *Par()* introduces the notion of parallelism between two reactive instructions; the execution order, which is not fixed, leads to an absence of determinism which is typical of the parallelism paradigm. Although the execution can choose non-deterministically two non-suspended instructions, the current implementation of *Junior* actually implements a deterministic order (left instruction, then right). However, if one of the two instructions is suspended, the non-suspended instruction will be executed before the end of the instant, which is not the case with the `Seq()` (1) instruction. The `Par()` instruction is also referred to as *parallel construct*.

(3) *`Atom(Action action)`*

This instruction allows the reactive instruction to interact with *Java* objects; the instruction is executed in an atomic way by the method `execute()`, as defined in the interface `Action`.

(4) *`Generate(String event, Object value)`*

The communication between reactive instructions can be performed via this instruction, which generates an event described by a specific `String` and is instantaneously broadcast by the reactive machine to other reactive instructions belonging to the same instant. A value object can optionally be associated to an event so that the instructions waiting for a specific event can retrieve event-related information.

In *Ecomobile*, the reaction to the presence of an M-agent is achieved by means of an event generation.

(5) `Until(String event, Program body)`

This instruction allows any reactive instruction to wait for a specific event; the program `body` is performed at each instant during which the event is absent; the program is also completed during the instant in which the event is present, but it then disappears and will not be executed in future instants.

(6) `Repeat(long count, Program body)`

This instruction implements finite loops; the program `body` is executed during a number of instants corresponding to the value of `count`.

(7) `Stop()`

This instruction allows the execution of a reactive instruction to be delayed to the next instant. For example, the following instructions can be used so that a particular event is expected during a certain amount of instants:

```
Seq(Until("open", Repeat(10, Stop())), Atom(javaClassOpen))
```

It waits for the event `open` during a maximum of 10 instants; if no event occurs within this period, the reactive instruction is completed and removed from the program. The next instruction `Atom()` will be executed in any case; particular action can be performed according to an optional event-related value (see `Generate()` (4)).

The dynamic insertion of reactive instructions is realized by means of the method `add()` issued from the reactive machine, which simply executes a `Par()` instruction performing the addition to the current instant-related program.

The current version of the reactive machine does not allow instructions to be added in sequence; still, replacing the `Par()` (2) with the `Seq()` (1) instruction in the `add()` method makes a sequential insertion possible.

4.2.3 Fair Threads

As we have seen, the reactive programming paradigm and its synchronous cooperative model can be considered as an interesting alternative to *Java* threads. In *Java*, the lack of a clear thread semantics makes a thread-based concurrency model providing both pre-emptive and cooperative frameworks difficult to implement. Moreover, the semantics strongly depends on the underlying execution environment. In order to improve the *Java* thread mechanism, the authors of *Junior* have developed the concept of *fair thread* on top of the *Junior* kernel [Bou01], leading to the elaboration of a particular library called *FairThread*.

According to this concept, a fair scheduler defines the execution phases during which there is an equal probability that threads will be executed, according to a strict round-robin algorithm; each thread must cooperate via the `cooperate()` method which suspends the thread execution and allows the scheduler to process other threads; the pre-emption can not occur in an uncontrolled way so that debugging is facilitated. Like reactive instructions, threads communicate via events. The fair scheduler ensures that each event generated within a phase is broadcast to all threads started in the scheduler, so that each thread can "see" the events in exactly the same way.

In the context of reactive programming, an execution phase corresponds to an instant and the *`cooperate()`* method corresponds to the *`Stop()`* reactive instruction; the semantics of the event is the same in both frameworks. However, from the efficiency point of view, reactive programming, which does not require context switching, is much more efficient; the instructions are simply interleaved, as shown on Figure 4-3, and events are broadcast to each instruction.

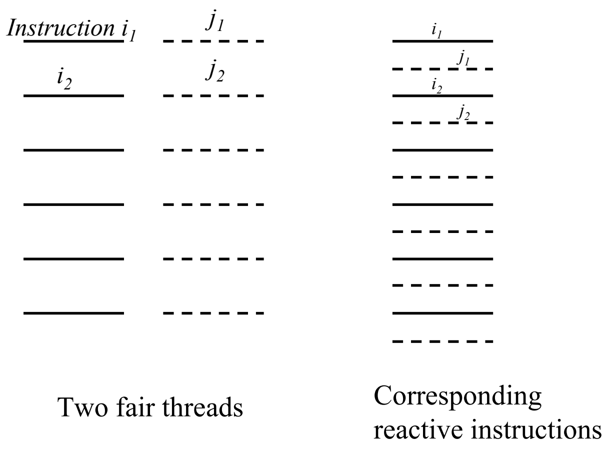


Figure 4-3. Interleaving of Reactive Instructions

The reactive programming paradigm is also preferred to an approach based on fair threads because it allows for a mapping between the Φ-behaviours and the reactive instructions. Besides, the implementation of *Ecomobile* with fair threads has revealed several drawbacks as far as scalability and performance are concerned, since the behavioural decomposition of the MBS leads to frequent context switching. Finally, memory overflow has led fair thread to fail after a few scheduling phases even with a reasonable number of network nodes.

4.2.4 Towards a Reactive Operating System

The *Reactive Operating System* (ROS) [Bo01] is a distributed operating system based upon the reactive model, which has been developed with *SugarCubes*. One of the major objectives of ROS is the support of mobile agents within a synchronous migration model. In this context, migration relies on a special reactive instruction called *`Freeze()`*, as well as on the RMI communication model.

According to the ROS, mobile agents are composed of reactive instructions. The migration is initiated with a specific instruction (*transfer*), which can be part of the mobile agent program or which can be inserted dynamically. During the migration, the reactive instructions which are being executed are *frozen* as long as they are declared as *freezable*. Instructions which are not *freezable* are simply not authorized to migrate. The frozen instructions are then transferred via a RMI call to the ROS server. This approach leads to a selective migration of reactive instructions.

Although it is not based on a FIPA-compliant environment, the ROS functionality resembles the *Ecomobile* agency. However, since the migration in *Ecomobile* is initiated by a specific behaviour[1] ($\Phi_{migration}$) and resorts to ACL-based communication, the M-agent does not require a transfer of reactive instructions and, therefore, the instructions do not need to be *freezable*. Still, the agent restoration is performed by the agency, which involves the dynamic adding of a reactive instruction. Furthermore, whereas the ROS migration relies on synchronous transfer, the *Ecomobile* agency resorts to the FIPA asynchronous communication model.

4.3 MAPPING OF THE MBS ON REACTIVE INSTRUCTIONS

We are now ready to introduce the implementation of *Ecomobile* into the *Junior* framework.

According to the previous sections, the *reactive* behaviours (Φ-behaviour) introduced in *Ecomobile* have important similarities with reactive instructions. As we have seen in Section 3.4.1, the Φ-behaviours are mainly influenced by the environment and have to be executed atomically by the M-agents. The synchronous concurrency model implemented in *Junior*, which is used to manage cooperative instructions, allows us to map each Φ-behaviour onto an atomic reactive instruction. The parallel construct `Par()` allows several M-agents to be processed in a non-deterministic way at the same time and to reflect an appropriate behaviour at the ecosystem level.

In *Ecomobile*, however, there is no reaction to absence; the active interference between M-agents implies that one M-agent tests the presence of another M-agent according to the $\Phi_{intereference}$-$\Phi_{dwelling}$ scheme.

The Φ-behaviours are activated via the `execute()` method (9) of the `Action` interface, according to the model shown on Table 4-2. The invocation of these behaviours is performed by the `Atom()` reactive instruction.

```
 1 public class PhiBehaviour implements Action {
 2
 3   LambdaAgent _agent;
 4
 5   public PhiAction(LambdaAgent agent) {
 6     _agent = agent;
 7   }
 8
 9   public void execute(Environment env) {
10     _agent.phiAction();  // Perform Φ-behaviour
11   }
12 }
```

Table 4-2. Implementation sample of a Φ-behaviour with a reactive instruction

The Φ-behaviours generate new reactive instructions dynamically, according to the MBS; further details concerning this matter are given in Section 4.3.2.

4.3.1 A Causality Problem in the $\Phi_{interference}$- $\Phi_{dwelling}$ Scheme

A simple mapping of Φ-behaviours onto reactive instructions may lead to a causality problem identical to the problem generated by the presence of events between several reactive instructions (see Section 4.1); in this context, however, the causality problem appears at the semantics level of reactive behaviours defined by the mobile behaviour scheme. The test for the presence of other M-agents may raise this high-

[1] The M-agent can not perform more than one reactive behaviour at a time.

level problem. As depicted on Figure 4-4, several agents can enter the same behaviour at a same instant and pursue their lifecycle in a similar way, assuming that the reactive behaviours are configured identically. It has to be noted that, during any reaction, several instructions are executed in an undetermined order "simulating" the parallelism effect. The Φ_{action} behaviour is not shown on the figure, since an M-agent performing this behavioural function can not be sensed by other M-agents.

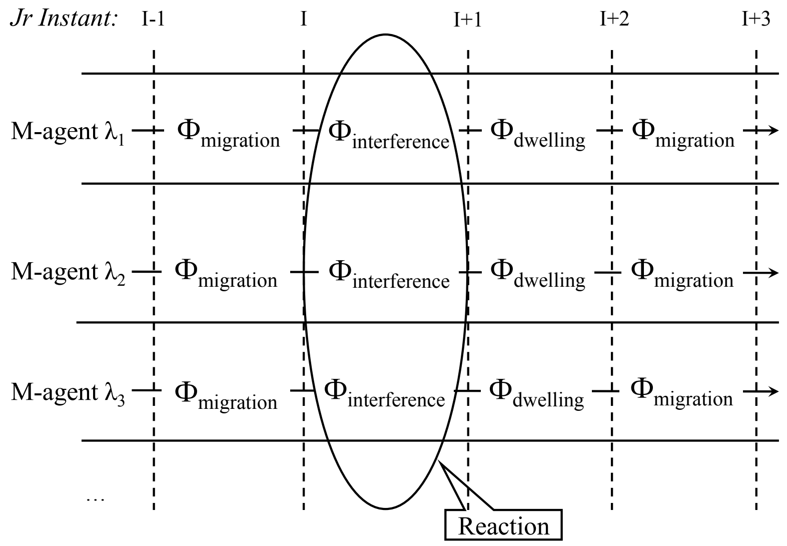

Jr Instant: I-1 I I+1 I+2 I+3
M-agent λ_1 $\Phi_{migration}$ $\Phi_{interference}$ $\Phi_{dwelling}$ $\Phi_{migration}$
M-agent λ_2 $\Phi_{migration}$ $\Phi_{interference}$ $\Phi_{dwelling}$ $\Phi_{migration}$
M-agent λ_3 $\Phi_{migration}$ $\Phi_{interference}$ $\Phi_{dwelling}$ $\Phi_{migration}$
…
Reaction

Figure 4-4. A causality problem in reactive behaviours

This synchronized behaviour introduces a causality problem: M-agents arriving at the same time ignore each other as if they were "blind"; according to the figure example, at instant *I*, the three M-agents λ_1, λ_2 and λ_3 arrive at the same place, perform the $\Phi_{interference}$ behaviour and then, believing that no agent is present because no agent is performing a $\Phi_{dwelling}$ at this instant, the three agents perform a $\Phi_{dwelling}$ at the next instant according to the MBS and therefore continue to co-reside. This misleading behaviour is in contradiction with the territoriality paradigm, which the M-agents behaviour are supposed to exhibit according to the MBS (see Section 3.4): the territorial behaviour should actually lead any M-agent to react to the presence of another co-residing M-agent. The M-agent's improper reaction hampers the self-regulation of the agent population and this, as simulation has shown, leads to an exponential growth of the population.

The problem can be solved by the contraction of the two reactive behaviours $\Phi_{interference}$ and $\Phi_{dwelling}$ into a unique reactive instruction, so that the two behaviours are performed in the same instant, as shown on Figure 4-5. Each M-agent first tests the presence of another agent; if there is no agent, it directly activates the next behaviour, which consists in waiting, and is therefore considered by the other M-agents as present. λ_1 performs the two Φ-behaviours in the same instant.

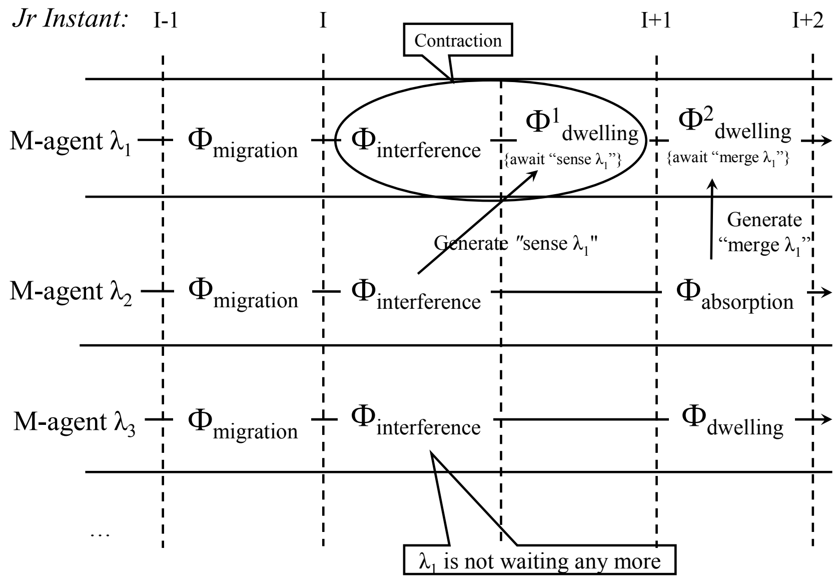


Figure 4-5. Contraction of the interference-dwelling scheme into a unique reaction

It is important to highlight that, during a reaction, the execution order of identical Φ-behaviours may however lead to variations due to environmental changes induced by a single Φ-behaviour execution. In this context, the term reactive is fully justified.

In this example, M-agent λ_1 performs $\Phi_{inteference}$, immediately followed by $\Phi_{dwelling}$, at instant *I*. During the same instant, M-agent λ_2 also performs $\Phi_{inteference}$, and is able to react to the presence of M-agent λ_1 by generating an appropriate *sense* event including a reference to the sensed M-agent λ_1. Finally, M-agent λ_3 performs $\Phi_{inteference}$ and do not detect any M-agent since no M-agent is dwelling any more.

The absorption phase requires M-agent λ_1 to remain synchronized with M-agent λ_2. This synchronization is actually achieved in the $\Phi_{dwelling}$ behaviour by expecting a *merge* event from M-agent λ_2. The event will be generated once the absorption is finished.

According to the MBS semantics, any coupling of reactive behaviours involving an event generation and a reaction to presence, as it is the case with $\Phi_{inteference}$ and $\Phi_{dwelling}$, may lead to a causality problem which must be tackled by a Φ-behaviour contraction.

4.3.2 The MBS-low and the MBS-high as Reactive Programs

The mapping of the mobile behaviour scheme *MBS-low/high* is presented in Table 4-3. The reactive instructions begin with a capital letter (*PhiBirth()*, for example) and are defined according to Table 4-2. The method *add()* performs the dynamic insertion of a reactive instruction into the reactive machine. This method actually prepares for the next instant.

```
public class M_agent extends LambdaAgent {

  M_agent sensedAgent = null, sensingAgent = null; // References to another M-agent
  boolean dwelling = false;

  public void phiBirth() { add(Jre.Atom(new PhiMigration(this))); }

  public void phiMigration(boolean afterMove) {
    // Pre/post-migration processing …
    add(Jre.Seq( Jre.Repeat(moveTime, Jre.Stop()), Jre.Atom(new PhiAction(this)) ));
11   }
12 
13   public void phiAction() { add(Jre.Atom(new PhiInterference(this))); }
14 
15   public void phiInterference() {
16     // sensedAgent = reference to a co-residing M-agent
17 
18     if (sensedAgent.isSensible()) {           // Is the agent sensible (dwelling) ?
19       sensedAgent.sensingAgent = this;        // Realize the coupling between agents
20 
21       add(Jre.Seq(Jre.Generate("sense:"+otherAgent.getID()),
22           Jre.Atom(new PhiAbsorption(this))));
23       return ;                                // Instruction terminated
24     }
25     phiDwelling(); // Contraction to avoid causality problem (it is not a react. inst.)
26   }
27 
28   public void phiAbsorption() {
29     // … perform knowledge transfer
30 
31     switch (MBSSelected) {                    // Next instruction depending on MBS
32       case MBSHighDiffusion:
33         add(Jre.Seq( Jre.Generate("merge:"+sensedAgent.getID()),
34                      Jre.Atom(new PhiInterference(this))) );
35         break;
36 
37       case MBSLowDiffusion:
38         add(Jre.Seq( Jre.Generate("merge:"+sensedAgent.getID()),
39                      Jre.Atom(new PhiMigration(this))));
40         break;
41     }
42   }
43 
44   public void phiDwelling() {
45     if (dwelling) {
46       dwelling = false;
47 
48       if (sensingAgent != null)       // Any interfering M-agent?
49         add(Jre.Seq(Jre.Await("merge:"+getID()), Jre.Atom(new PhiDeath(this))));
50       else {
51         add(Jre.Atom(new PhiClone(this)));
52         break;
53       }
54     } else {
55       dwelling = true; sensingAgent = null;
56 
57       add(Jre.Seq(Jre.Until("sense:"+getID(),
58                   Jre.Repeat(_waitTime, Jre.Stop())), Jre.Atom(new PhiDwelling(this))));
59     }
60   }
61   public void phiClone() { add(Jre.Atom(new PhiMigration(this))); }
62   public void phiDeath() { Jre.Atom(new Terminate(this)); }
63 }
```

Table 4-3. Description of MBS using the dynamic insertion of reactive instructions

The appropriate type of MBS is defined by `MBSSelected`, which corresponds either to `MBSHighDiffusion` or to `MBSLowDiffusion`. The variable `moveTime` of the `phiMigration()` method (10) simulates the duration of the migration. Once the migration has been achieved, this method is called again in order to perform post-migration initialization and to add the next reactive instruction. In the method `phiInterference()`, the M-agent is querying the agency for the presence of a waiting agent performing the Φ_{dwelling} behaviour (16-18). When a sensing agent has been found, the M-agent generates a `sense` event containing the reference to the waiting agent (21). The `phiAbsorption()` method informs the sensed agent of the end of the knowledge transfer by generating a `merge` event (33, 38). According to the MBS type, the sensing M-agent will reiterate the interference-absorption scheme (34) or leave the node and pursue its route (39), whereas the sensed M-agent will die (49).

The next section is devoted to the deployment of *Ecomobile* in the *Jade* agent platform; we will proceed, in particular, to the analysis of the interactions between the reactive machine and the agency, and we will examine how the reactions are triggered. This part of the implementation is also concerned with the co-existence of mobile agents and stationary FIPA agents.

4.4 DEPLOYMENT WITH JADE

The deployment of *Ecomobile* components within the network infrastructure is achieved by means of a FIPA-compliant agent platform which is assumed to be installed in each network device. The agents' migration occurs via migration services provided by an agency which is itself a FIPA agent, so that the M-agents do not depend on any specific mobile agent platform: the agent platform provides our agents deployment with the necessary communication infrastructure as well as with an adequate security framework.

This approach is characterized by asynchronous transfer of mobile agents between the network nodes and synchronous activation of the reactive machine instance which is present in each agency. In a pure simulated environment, there is only one reactive machine instance taking care of the physical migration of M-agents. In a real environment, on the contrary, the distribution of several agencies over the network and the asynchronous transfer of mobile agents involves several reactive machine instances executed in parallel; each machine manages its own instants without any form of synchronization.

When the agent platform supports a synchronous intra-agent activity model, the deployment of the agency is facilitated because the processing of newly arriving mobile agents can be postponed. This requirement is fulfilled by the *Jade* agent platform (see Section 1.5.2). In the next section, we examine in detail the implementation of the agency with *Jade*.

4.4.1 *Ecomobile* Agency

Jade defines containers as places which host agents. Although *Jade* supports intra-platform mobility between containers, we do not use this feature in *Ecomobile*. The *Ecomobile* agency is a stationary FIPA agent registered in the main container[1], which is created automatically at the starting time. The agency provides M-agents with a reactive machine, mobility services and controlled access to the local environment; it manages the places' configuration[2] and its related connectivity matrix as well as the associated blackboards. These components are not only used by the MBS, but also in the task objectives context.

The agency, which is assumed to be embedded in each network node, involves the presence of a FIPA agent platform. Of course, other FIPA agents can seamlessly share the same platform. Our definition of M-agents implies that no direct FIPA-based ACL communication is possible between a mobile agent and a stationary agent. The agency, which is a FIPA agent and therefore has the possibility to process ACL messages, can provide M-agents with two different service types in order to achieve this kind of communication: a *mediation* service which enables the communication between M-agents and FIPA

[1] Although there is no restriction to the use of other containers, the target container has to be designated by the agency when messages are exchanged so that the adoption of a single container name facilitates the addressing.
[2] Note that places are not *Jade* containers.

agents, or a *morphing* service which can transform an M-agent into a FIPA agent, and vice versa; this kind of service could be useful for the deployment within the network of high priority messages or tasks, and of sophisticated services implemented by means of a FIPA agents, for example. In this case, a task objective would contain the agent to be registered in the agent platform. The *mediation* and *morphing* services have not been implemented in *Ecomobile* yet; they are however considered in the *FIPA-mob* project (see Section 4.5).

The intra-agency activities rely on a cyclic behaviour (class `CyclicBehaviour`) in which the `action()` callback method constitutes the main entry point; the method invocation is performed by the *Jade* kernel, which is the internal agent scheduler. In our agency, two operations are performed at each activation: the behaviour checks for available ACL messages; it then activates the reactive machine by performing a call to `react()`. The *Jade* cooperative behaviour model prevents the ongoing reaction from being pre-empted by incoming ACL messages. Once the reaction has been completed, the message will be buffered and processed by the `MobilityService` class.

The *Ecomobile* agency model is depicted on Figure 4-6.

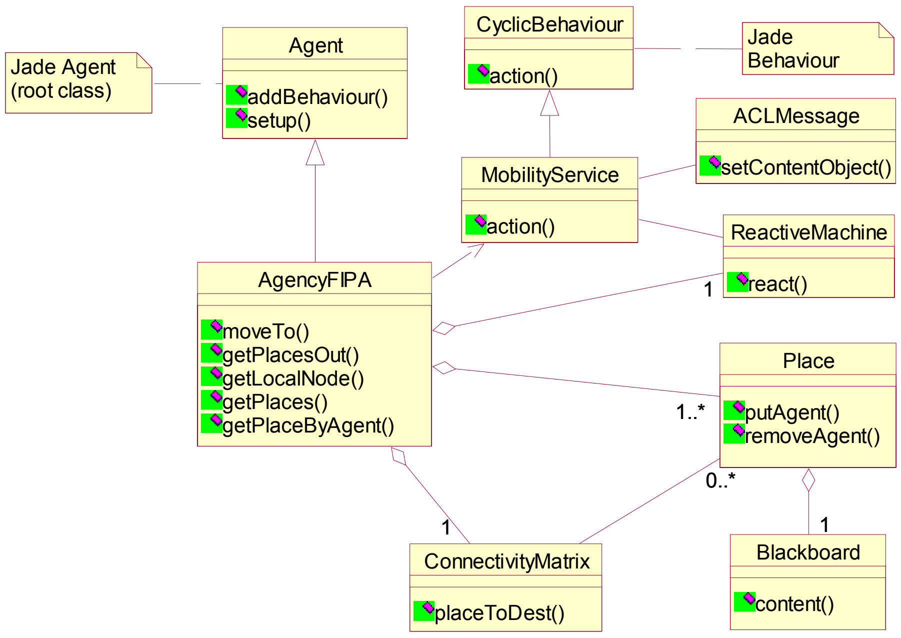


Figure 4-6. UML diagram of the *Ecomobile* agency model

In the current state of the implementation, the transfer of the M-agents is achieved by means of an ACL message exchange between agencies. *Jade* provides the `setContentObject()` method of the `ACLMessage` class, which allows any kind of object to be transmitted in the message contents by means of serialization and encoding based on the *Base64* format[1]. This mechanism, however, is not supported by FIPA yet. In *Java*, the de-serialization of a *Java* object requires the presence of the object class [Pi98].

[1] The "Base64" format, defined in RFC 1521, is used in MIME-encoded documents such as electronic mail messages with embedded images and audio files. More information at http://www.fourmilab.ch/webtools/base64

Since the *Ecomobile* agencies only have access to local information, the classes of task objectives also require to be transferred and dynamically bound to the JVM so that the agent context can properly de-serialize and instantiate the TOs. The TOs class transfer may rely on the same mechanisms used for agent migration provided by the agency, or can be controlled by a specific permanent *TO management* related task objective based on *TO_Inspector*, for example, which disseminate the classes within each agency. This kind of mechanism has not been implemented in the current version of *Ecomobile*; in the scope of our experiments, all TOs classes are visible by each agency. In the current release of the JVM (1.3.1_01), class unbinding is not possible and still remains an open issue [FM99].

Another approach addressing the problem of agent migration in a FIPA environment, which is being currently investigated by means of the FIPA-OS agent platform in the scope of the *FIPA-mob* project, will be introduced in Section 4.5. All the techniques used in the scope of this project can also be applied in a *Jade* environment.

Finally, the message exchange between two agencies could be improved in a future extension with the adoption of XML as a contents language and a particular ontology defining the tag semantics specific to *Ecomobile*. Exchange of more structured information related to the connectivity matrix will then enable a proof validation of the information received from the underlying node in case of connectivity changes. This procedure is also useful to update the connectivity matrix when no automatic detection can be triggered by the active node. The ACL-based protocol governing M-agent migration could also rely on the same ontology.

The environment of the *Ecomobile* agency with an illustration of the agent migration procedure is depicted on Figure 4-7.

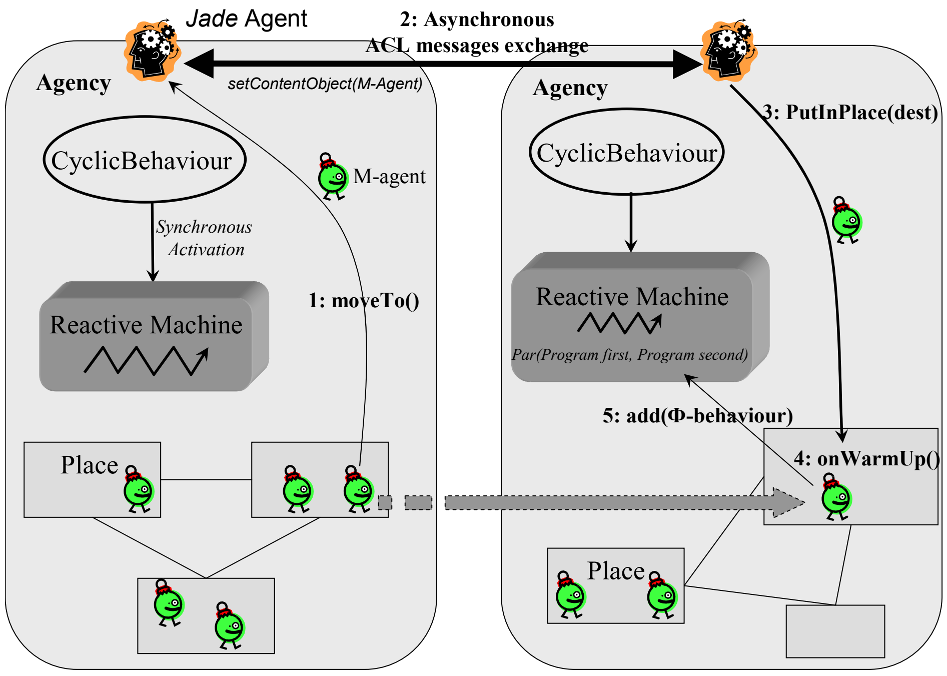


Figure 4-7. M-agent migration from one reactive machine to the other

We can for example assume that an M-agent requests the agency to move it: the $\Phi_{migration}$ behaviour invokes the `moveTo()` method (1) provided by the agency, and informs the agent of its destination. The agency uses the *UPI* to determine that an inter-agency migration is required. The agent is then directly embedded in an ACL message and immediately sent to the corresponding agency (2). The sender agency continues its activities while the remote agency stores the message in an internal buffer until the cyclic behaviour processes it. The agency then places the M-agent at the right place and resumes the execution of the M-agent by calling the method `onWarmUp()` (3). This method inserts into the reactive machine the Φ-behaviour (4) which will be executed at the next instant. The next behaviour is actually $\Phi_{\text{migration}}$ again, and it is called to perform a *post-migration* behaviour; internal variables related to the destination can be updated by the M-agent itself.

Another view of the interactions between the *Ecomobile* components is depicted on Figure 4-8.

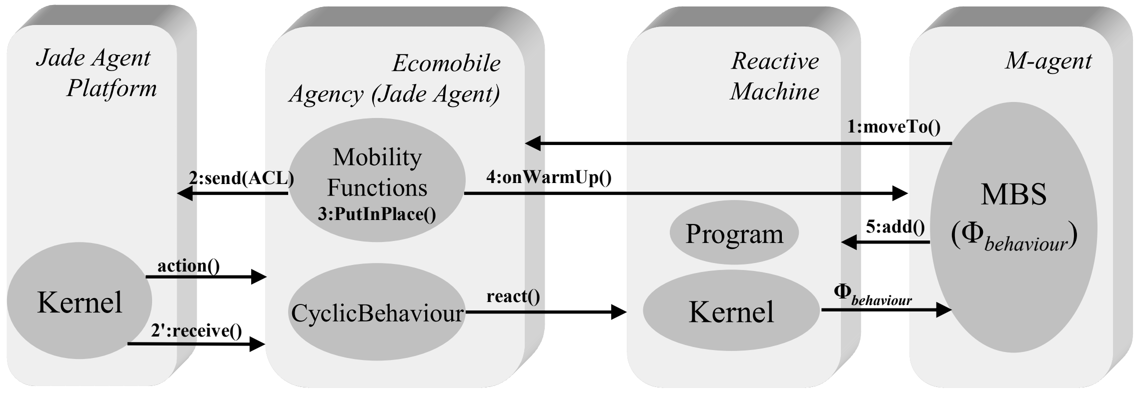


Figure 4-8. Interactions between the *Ecomobile* components during migration

The agency cyclically performs the activation of the reactive machine followed by a check (2') on the arrival of incoming ACL messages. If a message contains an M-agent, the agency puts it in the correct place (3) and activates it via the `onWarmUp()` method (4). At the moment, only migration messages have been implemented.

4.4.2 The LEAP Project

The *Lightweight and Extensible Agent Platform* (LEAP)[1] is a project which aims at the realization of a FIPA platform that can be deployed seamlessly on any *Java*-enabled device endowed with sufficient resources and with a wired or wireless connection, such as PDAs and smart phones. *LEAP* significantly contributes to providing network devices with an embedded agent platform.

LEAP is based on *Jade* and provides different kernel models, depending on the target environment in which the agent platform has to be deployed [BP01]. The target environment is characterized by one of the three different *Java* platforms provided by *Sun*: the *Java 2 Enterprise Edition* (J2EE), the *Java 2 Standard Edition* (J2SE) and the *Java 2 Micro Edition* (J2ME). The latter is intended for portable devices and includes two configurations: *Connected Device Configuration* (CDC), for devices with memory and a processing power comparable to that of a small PC, and *Connected Limited Device Configuration* (CLDC), for connected devices with strict restrictions concerning resources. Each platform has its own virtual machine model.

[1] http://leap.crm-paris.com

LEAP can be deployed on all three target environments thanks to an optimized and lightweight kernel module that manages the platform and the intra-agent activities according to the *Jade* behaviour model, and thanks to a communication module which handles the heterogeneity of the communication protocols based on different *Message Transport Protocols* (MTPs). The *LEAP* agent platform has been tested with the following emulators and end user devices: Palm Vx and Palm emulator, Quartz emulator, Psion 5MX coupled through IrDA with a mobile phone to the Internet and Siemens-MIDP-Emulator for mobile phones.

Several tests with *Jade* and *LEAP* have been performed on a small network of *Unix* machines and PCs; an *Ecomobile* agency has been launched within the main container on each host. We have chosen to use the *MTP HTTP* add-on to address the inter-platform communication, the intra-platform communication relying on RMI for *Jade* and, for *LEAP*, on a particular optimized protocol called *Jade Internal Command Protocol* (JICP), which is well suited to wireless links. These successful experiments have encouraged us to investigate the deployment of *LEAP* in the scope of a *Java*-based active node environment. Although active nodes are not limited by the same computing power and memory restrictions as mobile devices, the small footprint of the *LEAP* platform should facilitate its integration as an active application on top of operating systems such as *Janos* or *ANTS* (see Section 1.4.1).

4.4.3 Considerations about Efficiency and Scalability

We have measured the size of an M-agent transferred from one machine to another. When the agent context is empty that is, when no task objective has been loaded, the M-agent, which is *Base64*-encoded in an ACL message has a size of 1'028 bytes. Since the ACL message is fully specified by FIPA and since the encoded agent in the message contents is also standardized, the size of the M-agent does not depend on the underlying platform or machine. 668 bytes are added to the original 1'028 bytes to form the complete ACL message including sender and receiver addresses, ontology and language-related information. Because of its efficient computational model, the size of a serialized task objective is small; for example, the task *TO_Routing* is 472 bytes long. Assuming that the limit of task objectives within the agent context is 100 TOs, the maximal size of the agent context is 47'200 bytes, an M-agent is therefore inferior to 50 KBytes.

Ecomobile relies on an agent platform and therefore depends on the scalability of the agent platform itself. As far as our middleware is concerned, the addition of a new node in the network implies the set-up of the agent platform on the one hand, and the installation of the agency on the other hand. Ideally, the active node should be delivered together with a FIPA agent platform and the connection of this node to its neighbours should trigger an automatic detection mechanism in the *NodeOS* (see Section 1.4.1), so that the agent platform could inform the agency about the new platform and could launch a dynamic installation of the *Ecomobile* agency. Unfortunately, FIPA would first have to support mobile code before it could support such an automated agents deployment. Therefore, the agent still has to be started manually; an automatic update of the connectivity matrix can then be triggered by the *NodeOS*, which is aware of the intra/inter-node connectivity.

The installation of new agencies requires a registration to the *Agent Management System* (AMS) and may even require a registration to the *Directory Facilitator* (DF). Although the connectivity matrix can replace the DF in *Ecomobile*, intelligent agents belonging to other systems could wish to be informed about the presence of an *Ecomobile* agency and this information can be provided by the DF as long as the agent has registered automatically or manually.

According to the FIPA reference model, the ACL message transport layer relies on the *Agent Communication Channel* (ACC) as well as on different *Message Transport Protocols* (MTPs). This layered architecture allows the agent to send messages in a transparent way: the agent only needs to know the agent name but does not require any information concerning the remote platform transport mechanism. The ACC must therefore be informed of the existence of remote platforms and of their MTPs. The dynamic platform registration is not supported yet in the current implementation of most agent platforms.

4.5 THE MOBILITY SUPPORT WITH FIPA-OS

In the previous section, we have presented a possible implementation of *Ecomobile* with *Jade*. The agent migration is achieved by means of an ACL message exchange between *Ecomobile* agencies. The mobility paradigm in a FIPA agent platform however raises a wide range of issues related to code migration, such as the security issue or the agent architecture.

FIPA-OS is the agent platform which we have considered in the first place because it was the first publicly open source project on the one hand, and because of its adoption in the scope of the project *Shuffle*[1] on the other hand.

Still, the implementation of *Ecomobile* turned out to be more difficult to achieve with FIPA-OS than with *Jade* because of the asynchronous thread-based task model that defines the intra-agent activity in FIPA-OS. Two main problems actually remain to be solved: the processing of asynchronous messages should not pre-empt the reactive machine, and the reaction activation policy should be defined within the task; in particular, the absence of any reactive instruction in the machine should not trigger any reaction. In *Jade*, these two issues are obviously handled by the cooperative scheduling policy and by the cyclic behaviour model.

Nevertheless, in the scope of the *FIPA-mob* project[2], several investigations have been conducted in parallel with the development of *Ecomobile* towards the mobility support in the FIPA-OS agent platform. The *FIPA-mob* project aims at defining an agency implementing the minimal requirements to support the migration of mobile agents in a FIPA-compliant environment. An important related work is the mCode framework [Pie98], which relies on a flexible and general approach for the management of code transfer including multiple threads mobile agent. We can consider *FIPA-mob* as a subset of mCode functionalities dedicated to a specific agent synchronous execution environment.

FIPA-mob was not originally intended to be applied to *Ecomobile*, and thus was not designed to implement a reactive machine. This project explores new agent models and includes extensive performance tests. Our investigations have shown that *FIPA-mob* actually yields an efficient mechanism for agent transportation, platform registration, while it defines a flexible agent architecture model. All these concepts can be adapted to the *Jade* platform.

In *FIPA-mob*, the agency is called *Mobility Management System* (MMS): although it differs from the *Ecomobile* agency, it also pursues objectives similar to those of an existing work in this area [Mak00]. The mobile agent system supported by the MMS is fully based on a cooperative message-oriented interaction model: each communicative act between agents or between an agent and the MMS requires a

[1] *Shuffle* is a EU project involving *Swisscom Innovations*, which aims at studying agent-based approaches for the control of UMTS resources (http://www.ist-shuffle.org).
[2] *FIPA-MOB* is a project realized in the scope of a diploma work in the *Telecom Group* at the Department of Informatics of the University of Fribourg (Switzerland).

specific message. As in *Ecomobile*, each place is associated to a blackboard in which messages are stored and processed by the MMS via a priority queue. The communication is bound to a place and does not permit any communication between places. Any agent can send a message to a particular agent, broadcast it to all agents or simply deposit an anonymous message.

The transfer of mobile agents is achieved with a separate communication layer external to the agent platform and by means of a particular interaction protocol. In the current version of the MMS, the mobile agents are transferred via RMI according to a client-server model. The interaction protocol ensures that the remote MMS has agreed and is ready for the reception of an agent, and requests the agent class when necessary by means of a specific class loader (class `MMSClassLoader`); it can be extended with authentication and security mechanisms.

The *FIPA-mob* project also uses a modified version of the ACC - called *PaiACC*[1] - implementation in order to support an automated remote-platform ACC cross-registration. This change allows any platform to register to a foreign FIPA-OS platform without any re-configuration and therefore without any undesired reboot of the remote platform. This mechanism makes the MAS more scalable and allows the MMS to disregard the underlying MTP.

In addition to migration services, the MMS provides a *mediation* service, which allows mobile agents to communicate with FIPA agents, and a *morphing* service which allows mobile agents to be transformed into FIPA agents, and vice versa; in this case, the agents exhibit a polymorphism property. The morphing mechanisms are currently under investigations.

The evaluation of the approach suggested in *FIPA-mob* is still subject to the elaboration of specific metrics, such as migration and restoration time, as well as adequate measurement techniques.

4.6 SUMMARY

The reactive programming paradigm constitutes an efficient framework to tackle the complex issues raised by reactive systems such as *Ecomobile*; the *Junior* framework provides a *Java*-based library for reactive programming and implements a cooperative synchronous concurrency model allowing instantaneous reaction to the presence of an event and delayed reaction to absence.

Junior has been considered to implement the mobile behaviour scheme by means of reactive instructions; the advantages of such an approach are twofold: while the ecosystem behaviour can be simulated and analyzed with a time-reference model based on the reaction instants, the architectural design of the reactive machine including the dynamic insertion of reactive instructions authorizes a scalable deployment of *Ecomobile* in an asynchronous distributed agent platform.

As we have shown, the reactive behaviours can be mapped onto reactive instructions as long as a solution is found to the potential causality problem for the $\Phi_{interference}$- $\Phi_{dwelling}$ scheme; for example, the Φ-behaviours involved in the interaction can be contracted into one reactive instruction. The implementation of *Ecomobile* into a reactive programming framework finally leads to further investigations towards a formal analysis of the ecosystem behaviour and to the definition of other types of MBS; in this context, tools automatically generating formal automata derived from a reactive program [Bou00] should be considered.

[1] This modified ACC has been mainly elaborated by the *Parallelism and Artificial Intelligence* (PAI) Group at the Department of Informatics of the University of Fribourg (Switzerland).

Jade is a FIPA-compliant agent platform defining cooperative synchronous behaviours. *Jade* is therefore a privileged platform considering our objectives. As we have seen, the *Ecomobile* agency is an agent itself; it provides the M-agent with a reactive machine and with components such as places and blackboards. Consequently, it can be said that a (lightweight) mobile agent system is implemented into a stationary FIPA agent. *Jade*'s cyclic behaviour, which triggers the machine's reaction, can deal with the asynchronous ACL messages used to transport the M-agents. The *LEAP* kernel, which provides a small footprint of the agent platform for computing and for memory limited devices like mobile devices or PDAs, also resorts to the *Jade* mechanisms. Finally, a combination of an active node operating system with *LEAP* would allow *Ecomobile* to be deployed into active networks.

Chapter 5
Simulation and Results

The *Ecomobile* concepts have been implemented by means of the reactive programming framework described in Chapter 4. We now present a number of experiments with particular emphasis on adaptability to the different network topologies considered for transport networks. We have performed functional simulation of the threefold architecture of *Ecomobile* thanks to a specific framework called GNMT which is briefly described in the first section of this chapter and in Appendix A. All the components of our middleware have been simulated, from the agent system, which includes the reactive machine and the agency services, such as migration, connectivity matrix, etc., to the M-agents, including an implementation of the *MBS-low/high*, and of the task objectives with their instances in local blackboards. Since the physical migration and the network itself (nodes and links) are simulated, no agent platform has been used in these experiments.

Three network topologies have been considered, namely the *Square*, the *Fantasy* and the *Swiss* network. Each of these topologies has distinct characteristics and constitutes an interesting network configuration for the study of the ecosystem behaviour. The experiments have been performed with different Φ-behaviours parameters so that their influence on the system's stability can be discussed along this chapter.

The evaluation of the *Ecomobile* infrastructure obviously requires specific metrics adapted to behaviour analysis; according to our architectural model, there are two main categories of metrics: MBS-related metrics are specific to the M-agents general behaviour while TO-related metrics are specific to the task objectives. The MBS-related metrics can be divided into two subcategories: the metrics fully depending on the MBS, such as population size and visit frequency, and the metrics depending on the TO lifecycle, such as diffusion ratio and agent context size. Examples of TO-related metrics are "number of discovered paths" or "current Quality-of-Service".

The response of *Ecomobile* to the dynamic insertion of task objectives has been analyzed through four generic TO models presented in Section 3.6: *TO_Travel*, *TO_PathSelect*, *TO_ExhaustivePathFinder* and *TO_Routing*. The TO-related metrics which are presented in this chapter have been developed in this context.

5.1 THE GNMT FRAMEWORK

5.1.1 Introduction

The *Generic Network Management Tool* (GNMT) is a functional multi-layer network simulator which has been jointly developed by *Swisscom Innovations* and the *Telecom Group* of the Department of Informatics of the *University of Fribourg*. The GNMT development has been motivated by the primary objective of this thesis, which consisted in deploying intelligent and mobile agents in the optical transport network in order to address the complexity of network management and to build up an intelligent transport layer supporting new *optical* services. This is the reason why GNMT has been used as a simulation framework for *Ecomobile*. We can assume that multiple client layers may be interconnected to use the physical layer services. In this context, the simulator, which aims at supporting dynamic and asynchronous simulation with different traffic generators, provides a component-oriented framework which makes interactions between network layers possible.

The development of GNMT is part of the OPTIMA project (see Section 7.1) and is steadily becoming an *open source* project[1]. GNMT is related to similar projects like the open *Network Management System* project (open NMS)[2], which aims at the creation of an enterprise grade network management platform, or like *JavaSIM*[3], which is a component-oriented, compositional simulation environment. Both these projects are still at a developmental stage, but although they constitute interesting approaches, they do not meet our requirements as far as multi-layer transport network and mobile agent-based solutions are concerned.

GNMT has been developed by means of the *Model-View-Controller* (MVC) design pattern [GHJ$^+$95]. While the model relies on two different architecture types which will be detailed in Section 5.1.2, the view and the controller are presently based on a commercial third-party graphical library. GNMT supports both centralised and decentralised solutions, so that comparisons between these two approaches become possible. Centralised algorithms for optical path allocation, including routing algorithms, are currently being developed at *Swisscom Innovations*, whereas the simulator is used at the *University of Fribourg* for the implementation and the analysis of different approaches based upon mobile agent middleware, such as *Ecomobile*, and for the study of various deployment scenarios by means of FIPA agent platforms. A more detailed description of the GNMT framework and of its network model can be found in Appendix A.

5.1.2 The Core GNMT Network Model

The core GNMT network model is inspired by two different approaches characteristic of modelling layer networks: firstly, the generic functional architecture of ITU-T transport networks [G805_95] proposes a functional decomposition of the transport network in terms of layering and partitioning; while the layering deals with the separation between distinct transport technologies, the partitioning consists in subdividing the functional components within a single layer. This model supports multiple layers which can be interconnected according to a client-server relationship; in a layer network, the link connections formally provide connectivity between topologically adjacent sub-networks; they are provisioned by the services of a trail[4] at another layer. This layer is known as the server layer, while the layer in which link connections are issued is called the client layer.

The second approach is based on the ISO layered protocol model [T96]; it consists in a functional decomposition relying on a seven-layer protocol model. Since the ISO model was originally conceived within or around a single transport layer without any consideration for partitioning, it offers a perspective which is different from the approach proposed in the ITU-T model. The potential compatibility between the two models is still under discussion at the ITU-T.

[1] http://gnmt.sourceforge.net
[2] http://www.opennms.org
[3] http://www.javasim.org
[4] A *trail* is defined as the combination of the connection information augmented with additional overhead information used to achieve the *Operations, Administration and Maintenance* (OAM) objectives.

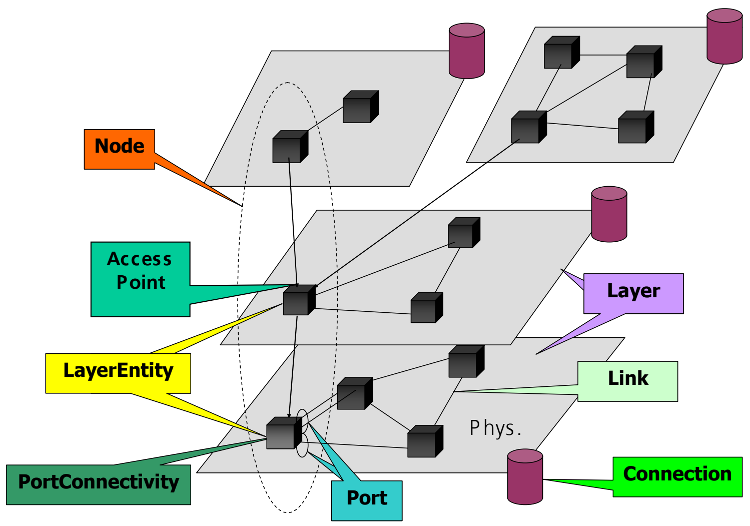


Figure 5-1. Core GNMT network model inspired from the ITU-T and ISO models.

The abstract core GNMT network model of is a generic representation of the main network components contained in a multi-layer transport network. Each component must therefore be specialized according to the network technology which is being considered in the simulation.

In the beginning, the network may be composed of multiple layers. A *layer* (class `Layer`) is defined as a collection of nodes associated with the same transport technology (IP, ATM, SDH, etc.) and therefore refers to a specific protocol. The layered model of the GNMT is composed of the lowest layer known as physical layer, on top of which logical or virtual layers are interconnected.

Within the same layer, the *node* is designated as a layer entity (class `LayerEntity`) and represents any kind of network element such as a cross-connect, an optical switch, a router, etc. Layer entities are *horizontally* connected with links (class `Link`) according to a certain topology. In a client-server multi-layer relationship, each layer entity belonging to the server layer can supply a collection of *Access Points* (APs) to other layer entities belonging to client layers (higher layers), so that the entities are *vertically* connected. According to the ITU-T model, the *client* layer entities, which have links to a server layer, correspond to *Connection Points* (CPs). Future releases of GNMT should allow a layer entity to implement several CPs.

Still, layer entities can only be added in the physical layer; the construction of virtual layers is performed through a selection of nodes existing in the physical layer. The connection between nodes is established freely, so that each virtual layer may have its own topology.

The concept of *node* (class `Node`) gathers all the layer entities belonging to the same physical entity that is, the lowest-level node in the physical layer. The node actually establishes a relationship between the ITU-T model and the ISO layered protocol model.

The *port* (class `Port`) is the abstract object which establishes a connection between two entities; it contains information related to the protocol and to *direction* property by means of the two attributes `in` and `out`, leading the links to be defined as uni/bi-directional. The port connectivity matrix (class `PortConnectivity`) gives information concerning the node connectivity and determines the association between *input* and *output* ports.

Deploying *Ecomobile* into a transport network therefore consists in placing an agency at each layer entity of the physical layer. The contents of the switching matrix have to be reflected in the agency's place connectivity matrix.

5.2 Notation, Metrics and Assumptions

This section defines the MBS-related metrics. TO-related metrics will be introduced in Section 5.4. The simulation results are presented graphically according to the following definition: the *X-axis* indicates the number of *Junior* instants (see Section 4.1) and thus reflects the temporal dimension, while the value of the metrics is depicted on the *Y-axis*.

Notation

λ_i	an M-agent in the ecosystem population	(1)
$AC(\lambda_i , t)$	Number of TOs in agent λ_i at time t (agent context)	(2)
$\tau(\lambda_i , t)$	Indicates if there is at least one task objective in λ_i, at time t (0 if $AC(\lambda_i , t)=0$, 1 if $AC(\lambda_i , t) > 0$)	(3)
MBS-low/high	Mobile behaviour scheme with low/high diffusion	(4)

A detailed description of *MBS-low/high* (4) can be found in Section 3.4.

Metrics

$P(t)$	***Population size***, i.e. number of M-agents at time t	(5)
$V(N_i, t)$	Visit frequency of M-agents within the node N_i	(6)
$AG(N_i, t)$	Number of agents present in node N_i	(7)
$IN(N_i, L_j, t)$	Number of agents arriving via link L_j into node N_i	(8)
$OUT(N_i, L_j, t)$	Number agents leaving from N_i via link L_j	(9)

$$D(t) := \frac{\sum_{i}^{N_\lambda(t)} \tau(\lambda_i, t)}{P(t)}$$ **Diffusion ratio**, i.e. propagation speed of the TO within the ecosystem (10)

$$C(t) := \frac{\sum_{i}^{N_\lambda(t)} AC(\lambda_i, t)}{P(t)}$$ **Mean context size**, i.e. number of TOs in the Agent Context (11)

The population size *P(t)* (5), the node visit frequency *V(N_{i,} t)* (6), the number of agents residing within a node (7) and the link visit frequency (8)+(9) do not directly depend on the task objectives themselves, except for the execution time. These metrics mainly depend on the mobile behaviour scheme; in other words, they are typically related to the basic ecosystem behaviour. On the contrary, the diffusion ratio *D(t)* (10) and the mean context size *C(t)* (11) strongly rely on the TO lifecycle. The inter-TO cooperation strategy, in particular, plays an important part as far as dissemination speed and regulation of the agent context size are concerned; unless cooperation has been defined, the context can grow exponentially and lead to performance degradation.

The measurement of *D(t)* and *C(t)* therefore provides an important feedback on the ecosystem behaviour, which can be used for tuning and improving the efficiency of the cooperation strategy. A possible extension of *Ecomobile* would consist in discovering how to automate this process.

According to the MBS definition, the migration time, designated by *phiMigration*, the waiting time, designated by *phiDwelling*, and the execution time, designated by *phiAction* constitute the major ecosystem parameters. Details about the behavioural strategies are given in Section 3.4.2.

We have performed several experiments with different values of ecosystem parameters; in this document, we propose to examine the *Ecomobile* behaviour with the following significant values:

phiMigration	migration time for $\Phi_{migration}$ - random value between 0 and {10, 20}
phiDwelling	waiting time for $\Phi_{dwelling}$ - fixed value equals {0, 5, 15}
phiAction	execution time for Φ_{action} - 0, since we assume fast TO execution and the execution time can be combined with the migration time.

As will be described in Section 5.3.2, *MBS-low* implements a fixed waiting time value while *MBS-high* implements a heuristic-based waiting time function. Simulation based on the above values will be presented along this chapter. In most cases, the simulation has been performed during 10'000 reactions (instants).

Since the network topology is composed of nodes and bi-directional single links, an M-agent can travel along a link in both directions; for the same reason, an M-agent which is leaving a node can also select the link from which it has arrived.

5.3 Behavioural Analysis of the MBS

Three network topologies have been defined for the analysis of the ecosystem behaviour in various network configurations: the *Square* network is a regular topology composed of 21 nodes and 32 links and exhibiting symmetric properties; regular topologies are frequently used in optical transport networks such as rings or circular topologies. The *Fantasy* network, which is composed of 18 nodes and 23 links, is a random network presenting an "exotic" connectivity scheme with different topology patterns (square, isolated nodes, bus, etc.). Finally, the *Swiss* network, composed of 33 nodes and 50 links, represents the optical transport network in Switzerland, in which each fibre is assumed to contain a single wavelength.

In addition, we have experimented other topologies such as circular, multi-ring and random networks, with different values of ecosystem parameters. However, since simulation results for this kind of networks lead to the same conclusions as those presented in this chapter, they are not presented in this document.

5.3.1 Node Visit Frequency

Since task objectives are activated during the M-agents' node visits, the *node visit frequency* $V(N_i, t)$ constitutes a major issue in *Ecomobile*, as well as in most mobile agent systems. The node visit frequency metric therefore has a deep impact on the system's performance. A node with a high nodal degree[1], which is therefore regarded as a "critical" node, is also expected to receive more visits than a node with a low nodal degree so that actions implemented into the task objectives can be more frequently activated. The measurement results over 10'000 instants for a regular network regarding visit frequency, with *MBS-low*

[1] The nodal degree of a node corresponds to the number of links connected to this node.

and *MBS-high*, is depicted respectively on Figure 5-2 and Figure 5-3. This metric turns out to be particularly useful when the simulation is performed for debugging purposes; an abnormal growth of this number can quickly be detected as an indication of an MBS deficiency.

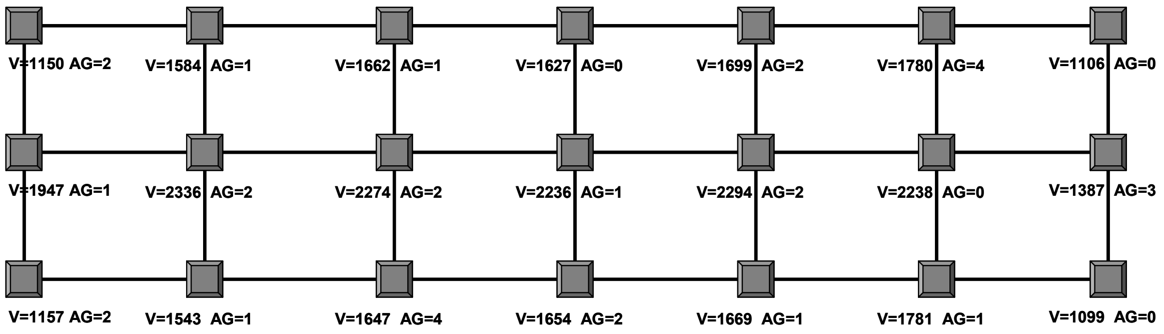


Figure 5-2. The *Square* network (MBS-low, phiDwelling=5, phiMigration=10)

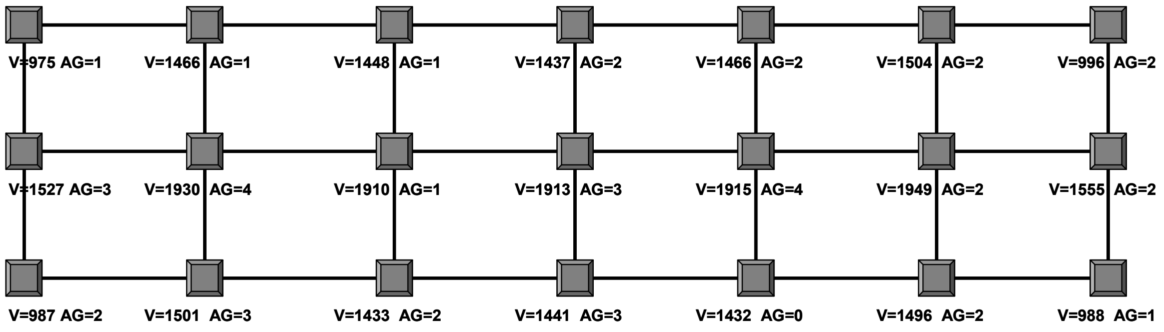


Figure 5-3. The *Square* network (*MBS-high*, *phiDwelling*=5, *phiMigration*=10)

The value of $AG(N_i, t)$ reflects the number of co-residing agents at the instant *t*; the value indicated on the figure corresponds to the last simulation instant.

The simulation results have shown that the number of M-agent visits, in this kind of network, is inferior by a factor of about ten to the total amount of reactions; the value of the visit frequency moreover depends on the nodal degree. Although the *MBS-high* exhibits a breadth-first parallel search behaviour, it also reveals a lower visit frequency for the same simulation duration. This surprising outcome is the result of the interference-absorption loop characteristic of the *MBS-high*: the M-agent makes sure that no other agent resides in the same place over a period defined by the waiting time and thus spends more time within the node it is visiting.

Similar differences can be observed between *MBS-low* and *MBS-high* regarding the other topologies; we will therefore present, for the next topologies, only simulation results based upon *MBS-low*.

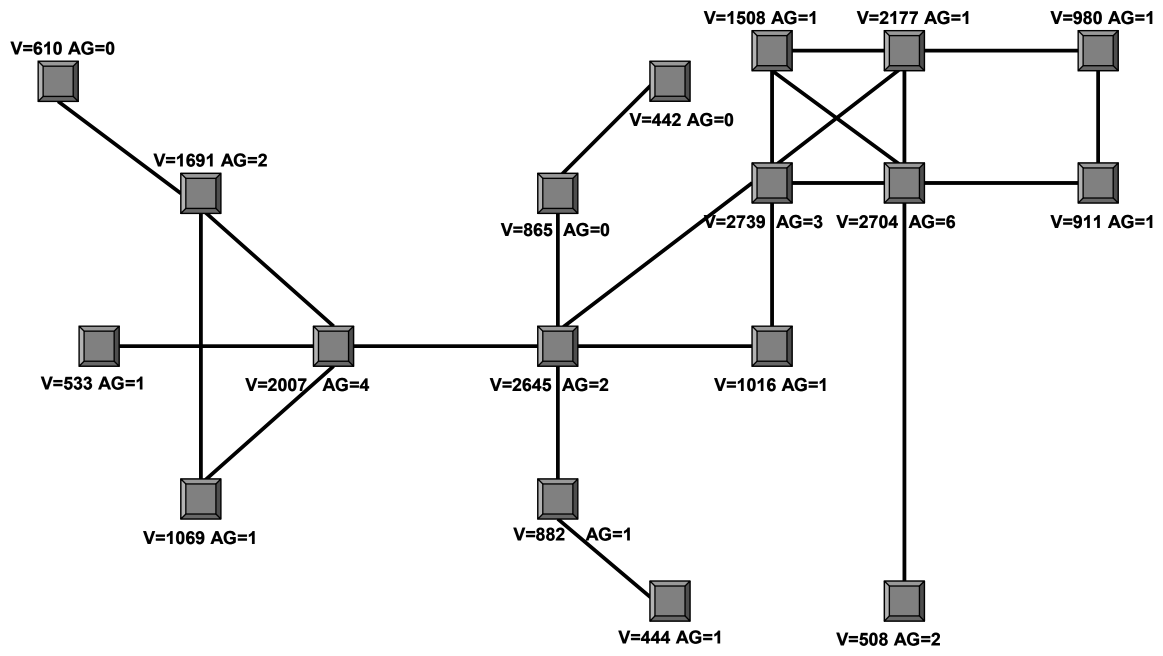


Figure 5-4. The *Fantasy* network (*MBS-low*, *phiDwelling*=5, *phiMigration*=10)

In the *Fantasy* network depicted on Figure 5-4, the visit frequency also appears to match the nodal degree. Although the network is asymmetric, the number of visits corresponds to a certain nodal degree reaching the same order of magnitude regardless of the node location. This interesting property results from the density-dependence property of the territoriality paradigm (see Section 3.4), on the one hand, and from the MBS definition on the other hand; the M-agents population is automatically regulated according to the visit frequency, which directly depends on the nodal degree; the cloning strategy defined in the MBS actually implies an investigation of all the links; the number of clones therefore equals the nodal degree. Obviously, this property can be observed when the migration time and the execution delay are statistically the same for each link and node.

Finally, the *Swiss* network has been simulated; the results are presented on Figure 5-5.

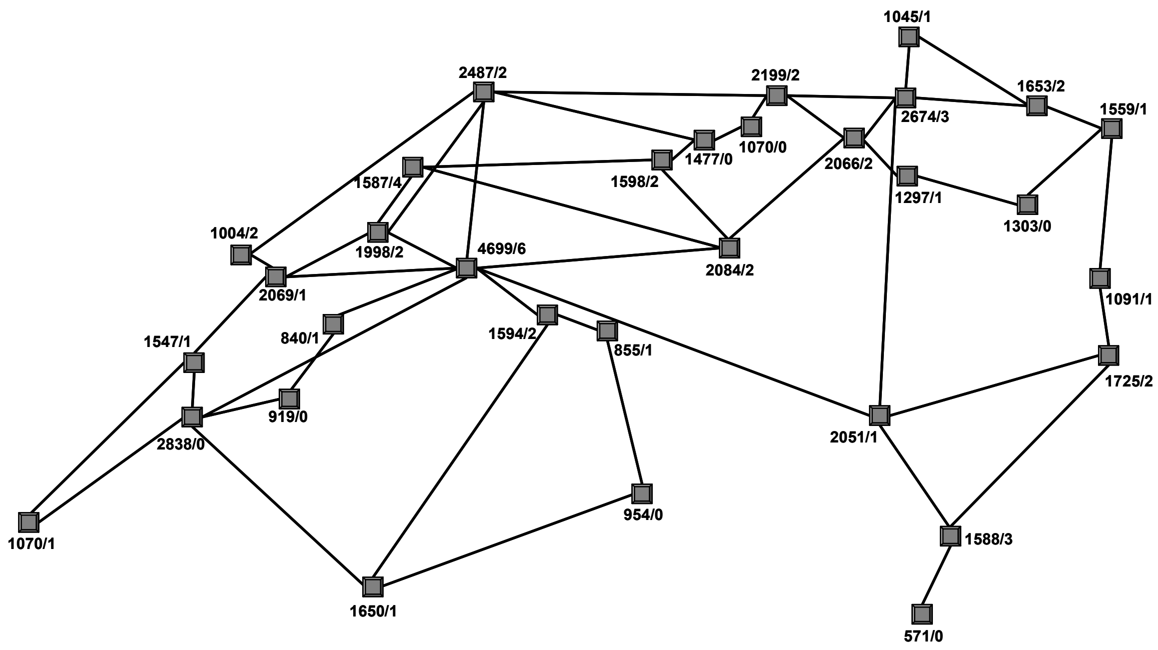


Figure 5-5. The *Swiss* Network (*MBS-low*, *phiDwelling*=5, *phiMigration*=10)

The first number represents $V(N_i, t)$ while the second number is $AG(N_i, t)$. As in the previous networks, the visit frequency depends on the nodal degree.

5.3.2 Link Visit Frequency

Whereas the visit frequency indicates the TOs' execution rate, the TOs' dissemination depends on the M-agents' movement. In this context, the number of agents arriving into a node and leaving it constitutes a precious indication. This metric is called *link visit frequency*.

The TOs are efficiently disseminated within the network as long as the number of M-agents arriving at a node is approximately identical to the number of M-agents leaving it. This condition was met in most cases when we implemented *MBS-low*. Figure 5-6 indicates the values of $IN(N_i, L_j, t)$ and $OUT(N_i, L_j, t)$ for a portion of the *Square* network.

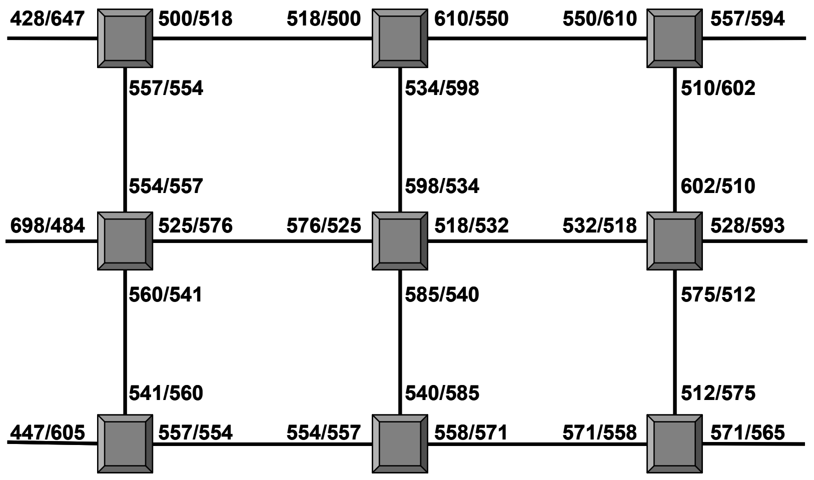


Figure 5-6. Number of M-agents arriving (first value) and leaving (second value) with MBS-low.

However, turning to *MBS-high*, we could observe that the link visit frequency was not well distributed. When a node exhibits a high nodal degree, the interference-absorption loop of *MBS-high* may induce the M-agent to spend an infinite time within a node. The resulting overload of meeting opportunities leads the TOs belonging to the residing M-agent to be blocked. The agent context is also subject to numerous transfers which may cause a profusion of TOs to be offloaded into the blackboard. This effect is depicted on Figure 5-7: while a lot of agents are arriving into the central node, only a few agents are leaving.

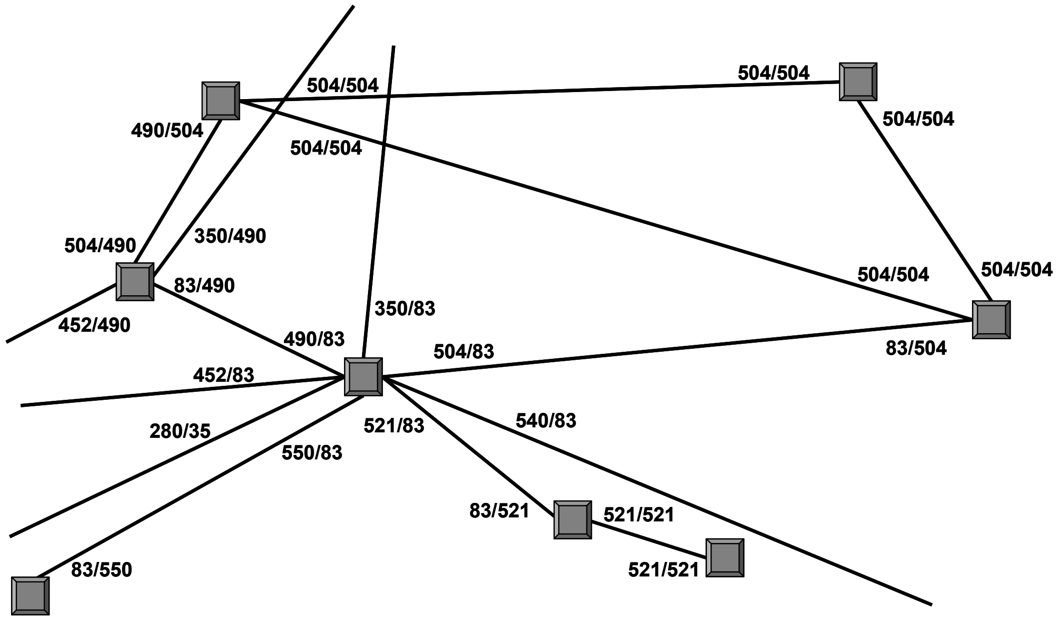


Figure 5-7. Overload of meeting opportunities (*phiDwelling*=5, *phiMigration*=10)

In order to overcome this problem, we have introduced a specific heuristic-based waiting time function which is defined by the following expression:

$$phiDwelling(N_i,t) = \begin{cases} 0, & if \sum_{j}^{\deg(N_i)} IN(N_i,L_j,t) - \sum_{j}^{\deg(N_i)} OUT(N_i,L_j,t) > 50 \\ 5, & otherwise \end{cases}$$

The maximum waiting time could also be adjusted dynamically according to the variation of *IN* and *OUT* by means of linear functions, but this still remains to be experimented.

The waiting time is now automatically adapted at each node. According to this expression, the difference between the number of arriving agents and the number of exiting agent must be less than a certain value[1]; if the difference exceeds this value, the waiting time is set to zero, so that the M-agent does not spend any more time expecting possible meetings. If the M-agents number is important, the interference-absorption loop remains efficient and continues to guarantee the population regulation.

Figure 5-8 shows the new values according to the conditional waiting time.

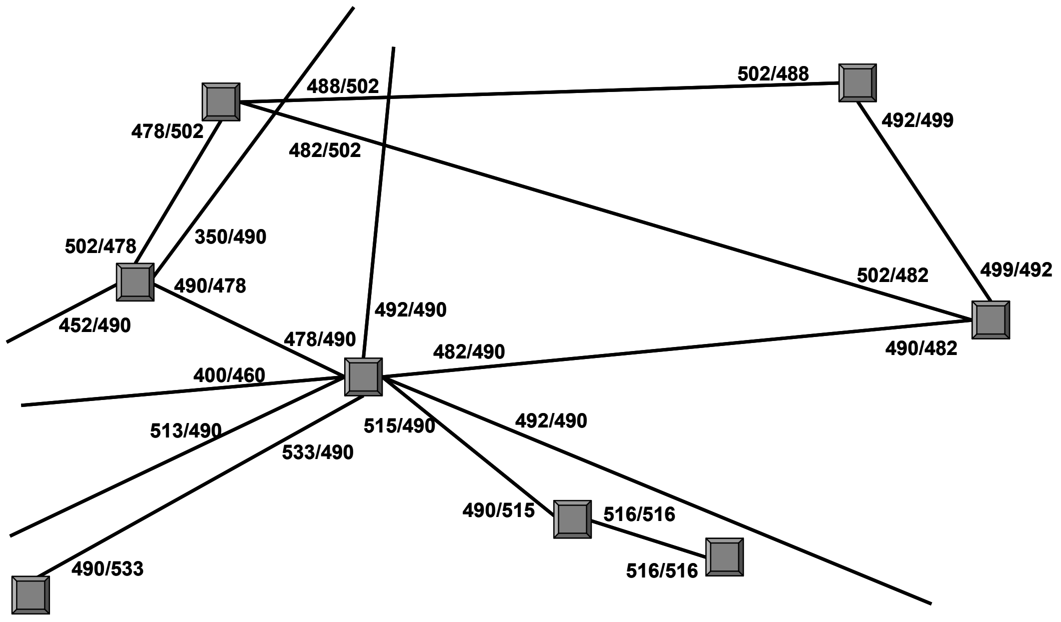


Figure 5-8. Link frequency visit with the heuristic-based waiting time function

It appears that the heuristic-based waiting time function described above has a particularly efficient balancing effect on the M-agents' migration. The number of exiting agents is now very close to the number of arriving agents. The same waiting time function could also be applied to *MBS-low* in order to slightly improve the link visit balancing effect, thus making the $\Phi_{dwelling}$ behaviour independent from the MBS.

[1] This value has been determined empirically and is adequate for the three topologies.

5.3.3 Population and Stability

We present simulation results concerning the evolution of the M-agent population over time; stability has been tested by means of trend lines and simple average calculation.

The monitoring process, which consists in storing the metrics values every 10 instants, maintains the number of M-agents and updates it when the Φ_{birth} and Φ_{death} behaviours have been performed. Figure 5-9 illustrates the general evolution pattern when *Ecomobile* is started by launching an M-agent anywhere in the network.

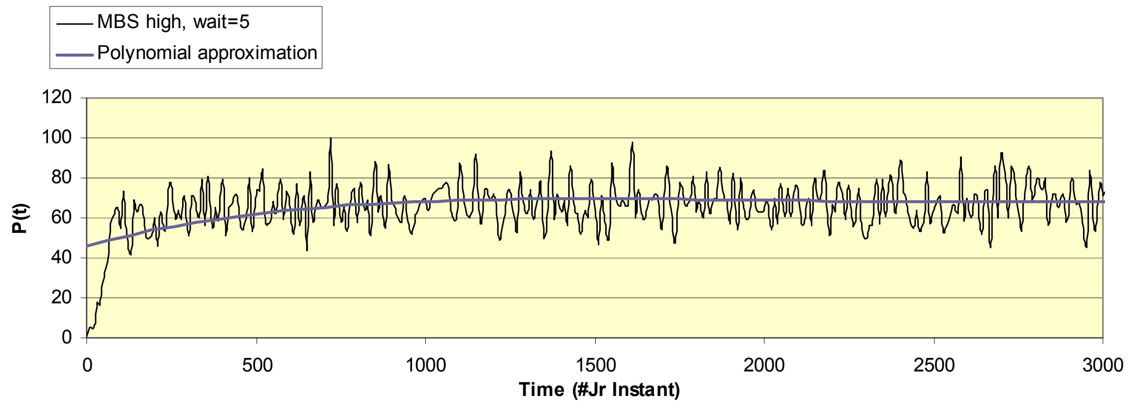


Figure 5-9. Example of the evolution of the M-agent population size over time

No M-agent is present at the beginning of each simulation and the population size is equal to zero. When the first M-agent starts its lifecycle in the network, it clones itself according to the MBS, and thus immediately contributes to the increase of the population; alternatively, meeting opportunities lead to the removal of agents and therefore contribute to the decrease of the population.

The first instants of the simulation are dedicated to the exploration of the network; the population grows rapidly until the first meetings between M-agents occur. A few additional instants are still necessary before a relative stability can be attained.

The evaluation of the ecosystem stability requires the observation of the population size average over a long period of time. In order to assess the mean value, we propose to use a polynomial trend line of order 6 calculated with the least squares fit through the points representing the value of *P(t)*.

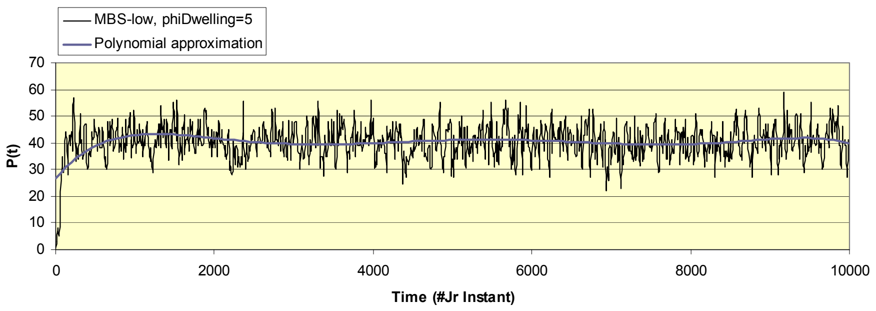
A more complex analysis of the ecosystem population and stability requires particular techniques referred to as *population ecology*, which provide different statistical approaches by means of data analysis techniques applied to temporal series, such as moving average or spectral analysis; this kind of investigation however remains out of scope of this work[1].

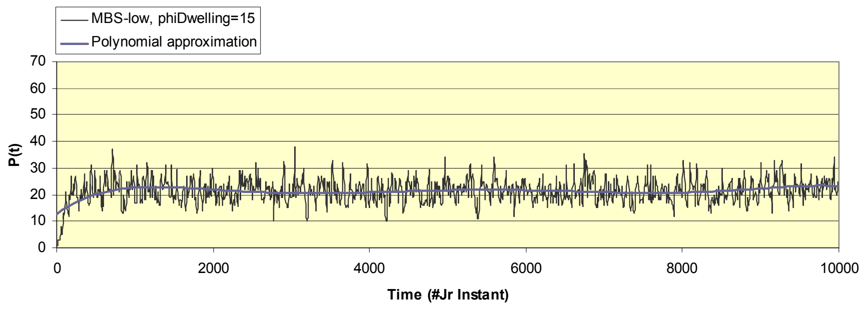
The M-agent population size in the *Square* and *Fantasy* networks respectively appears on Figure 5-10. The results reveal different graphic patterns according to the topology, to the MBS and to different values of $\Phi_{dwelling}$. Behaviour has been analyzed for a waiting time both inferior and superior to the migration time ($\Phi_{migration}$) which has been fixed to 10 instants. The mobile behaviour scheme *MBS-high* implements the heuristic-based waiting time function described in Section 5.3.2. In order to evaluate the difference of

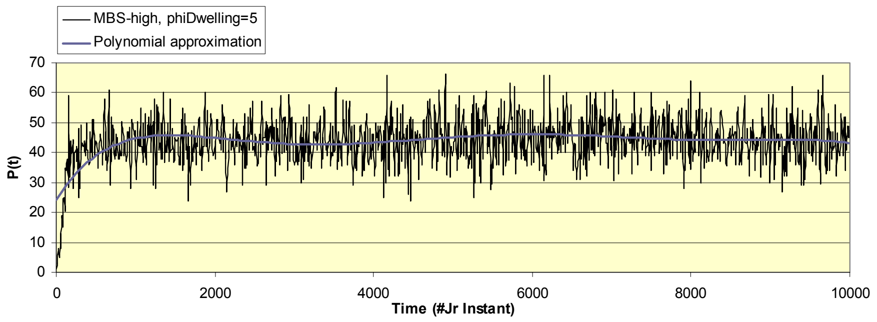
[1] A thorough discussion of population ecology can be found at http://www.ento.vt.edu/~sharov/popechome/welcome.html

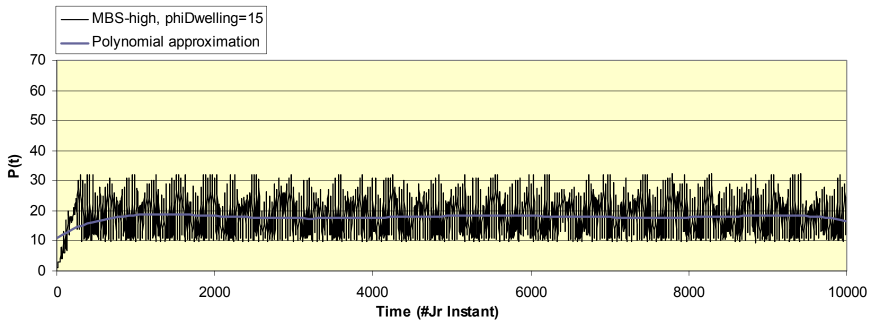
population size due to the introduction of our waiting time function, Figure 5-12 still illustrates the population size with *MBS-high* considering a constant waiting time.

***SQUARE* NETWORK** *(21 NODES)*









FANTASY NETWORK *(18 NODES)*

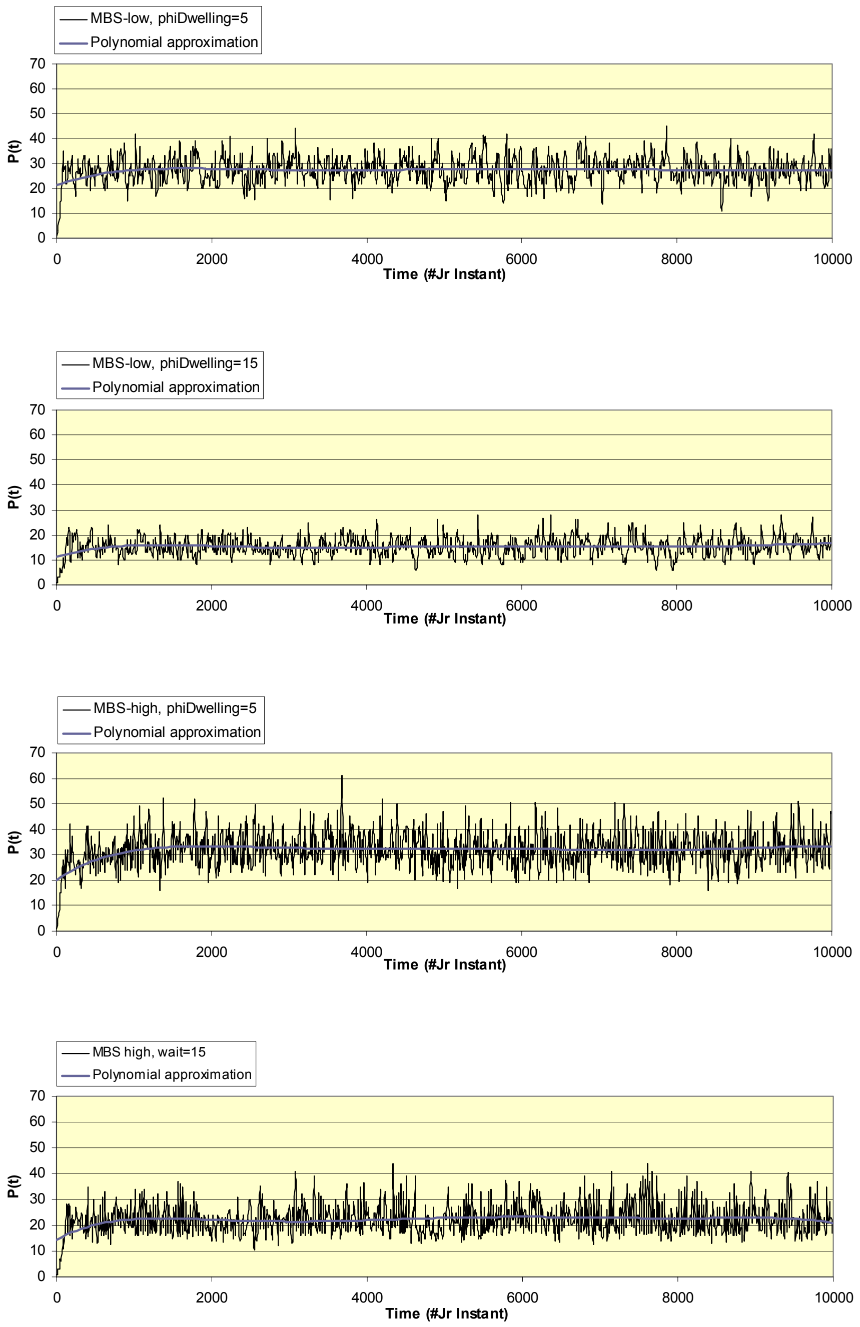


Figure 5-10. Evolution of the M-agent population over time (*phiMigration*=10)

Our observations start with the mean population size. Since the territorial behaviour of M-agents implies that each node is investigated by an M-agent "defending" its territory (see Section 3.4), the population size should approximate the number of nodes and grow when the migration time increases or when the waiting time decreases.

When *phiDwelling* is inferior to *phiMigration*, both MBS reveal a number of M-agents significantly superior to the number of nodes: the meeting opportunities are actually reduced because M-agents spend more time for migrating, and the cloning behaviour therefore occurs more frequently. The average number of M-agents is slightly higher in *MBS-high* than in *MBS-low*, because of the interference-absorption loop with systematic cloning. This behaviour also brings about important variations at each reaction, i.e. high number of births and deaths.

When *phiDwelling* exceeds *phiMigration*, *MBS-low* exhibits the best performance in terms of adequacy between the number of nodes and the population size. On the contrary, *MBS-high* reveals variations in the M-agent population which seem more difficult to explain. The graphic pattern first depends on the network topology: in regular networks like the *Square* network, the population size oscillates between its extreme values; the symmetric properties of the network combined with an important waiting time actually leads to a kind of lifecycle synchronism between all the M-agents. This synchronism effect does not appear in asymmetric networks like the *Fantasy* network, for example.

Our simulation has revealed that the M-agent population fits the network size when *phiDwelling* exceeds *phiMigration*, in *MBS-low*. In the other cases, the population size is comprised between the network size and twice the network size. Since the population size is correlated to the node visit frequency, which therefore determines the frequency of task objective activations, it appears preferable to opt for a population size slightly bigger than the network size. A more important population size will also indirectly favour M-agents with lower agent context size, since the TO instances will be spread over the agent population. For these reasons, we have chosen to keep the value of the waiting time inferior to the migration time.

We now propose to observe the impact of the migration time on the population in the *Swiss* network. Migration time is randomly computed between 0 and a maximum value (10 or 20 in this case). Figure 5-11 presents the M-agent population. In this case, the simulation duration has been extended to 20'000 instants.

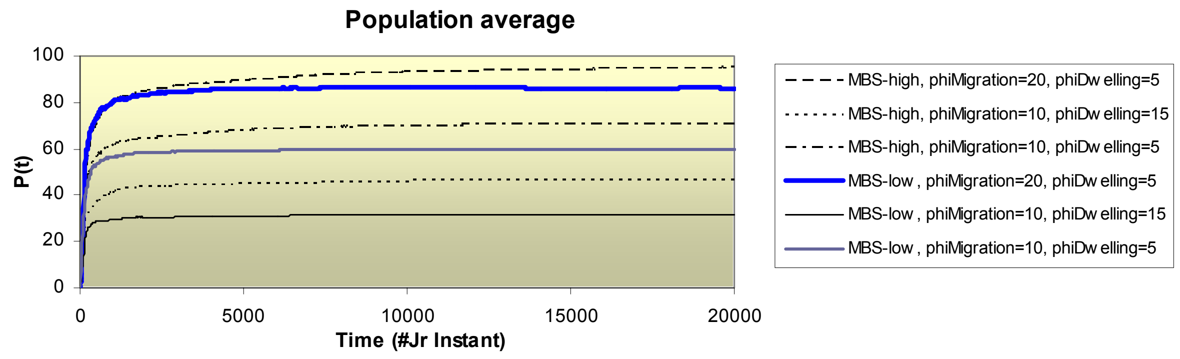


Figure 5-11. M-agent population in the Swiss network (33 nodes)

The population average is re-computed at each variation during the simulation. The increase of the migration time (or of the execution delay) obviously tends to favour an augmentation of the population, because agents which are migrating or performing a task cannot interfere with other agents.

Still, *MBS-low* including a migration time inferior to the waiting time reveals a population of M-agents matching the number of nodes.

Figure 5-12 depicts the evolution of the population when the M-agent exhibits a *MBS-high* with constant waiting time. There is a difference of about 10 agents in the M-agent population between this diagram and the one corresponding to the heuristic-based waiting time function (see Figure 5-10).

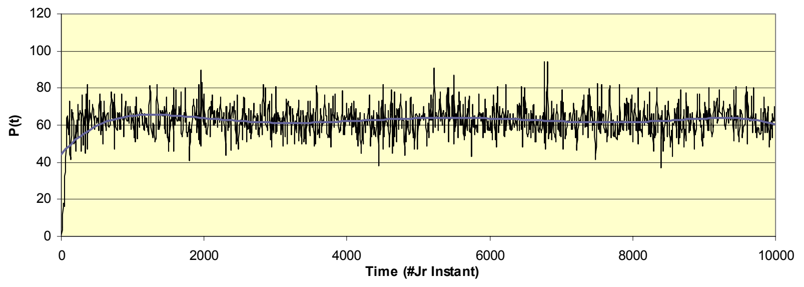


Figure 5-12. Population size with constant waiting time (*MBS-high*, *phiDwelling*=5, *phiMove*=10)

Although various scenarios with different waiting time and migration time have been considered for the analysis of the MBS, the experiments presented in Section 5.4 are based on the behavioural analysis of task objectives and are restricted to the simulation of M-agents implementing a waiting time of 5 instants and a migration time of 10 instants. Both MBS will be compared.

5.4 BEHAVIOURAL ANALYSIS OF TASK OBJECTIVES

This section is devoted to the study of the ecosystem response to the insertion of some of the task objective models presented in Section 3.6. The metrics defined in Section 5.2, in particular the mean context size *C(t)* and the diffusion ratio *D(t)*, will help us to evaluate their performance. Based on these metrics, particular attention will be paid on the implementation strategy as regards the inter-TO cooperation and the TO lifecycle. Further TO-related metrics will be defined in the corresponding sections.

According to the *Ecomobile* architecture, the task objectives are deposited into the blackboard of a particular place, i.e. of a particular node. As we have already seen, the M-agent population needs a few instants in order to reach an equilibrium so that it would not be appropriate to insert task objectives at the very beginning of the simulation. This is the reason why a mechanism has been implemented into the blackboard in order to program a delay for the loading of task objectives; according to the M-agent population analysis (see Section 5.3.3), we have associated a delay of 500 instants to each TO at the beginning of the simulation. It also has to be noted that the size limit of the agent context has been fixed to 100 TOs.

The persistence mechanism used for all the TOs leads the M-agent, which has no available place for hosting TOs, to offload them into the blackboard. This strategy prevents the TOs from being untimely removed from the ecosystem; the complexity of the behavioural analysis is consequently reduced, and the TOs keep the entire control of their lifecycles.

Finally, the *Swiss* network will constitute our major case study for these experiments; the three networks have been considered in the context of *TO_Routing* only.

5.4.1 TO_Travel

The *TO_Travel* task objective (see Section 3.6.1) successively builds cycle-free itineraries in a continuous way; these itineraries represent topology subsets. The cooperation mechanism removes redundant TOs containing identical itineraries. When a cycle has been detected, the itinerary is re-initialized and the task objective continues its trip. Therefore, and according to the TO lifecycle, a TO can only be removed from the ecosystem during the cooperation phase by another TO instance.

Figure 5-13 shows *C(t)* and *D(t)* for this particular TO model.

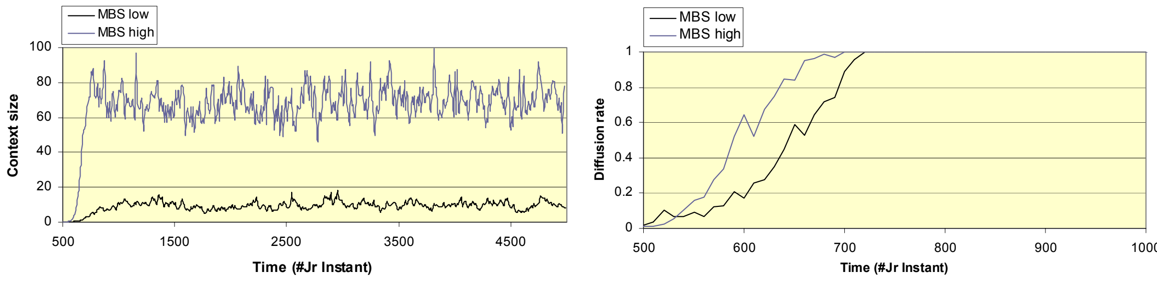


Figure 5-13. Analysis of *C(t)* and *D(t)* of *TO_Travel* with MBS-low and MBS-high

The mean agent context size *C(t)* is characterized by an important difference between *MBS-low* and *MBS-high*. The "high" diffusion of the latter favours a rapid exploration of different topology subsets; the M-agents are provided with TO instances owning distinct itineraries and consequently avoid their removal during the cooperation phase. On the contrary, *MBS-low* rapidly has to face TO instances having the same itinerary rapidly.

The cooperation strategy implemented into the task objective typically consists in eliminating the TOs simultaneously evolving in the M-agents; knowledge redundancy appears after a certain time in this kind of TO when the TOs' itinerary is cleared and re-started; the itineraries being exempt of cycle, two meeting agents can not have identical itineraries.

The diffusion ratio *D(t)* reveals the rapid TO dissemination when the M-agents have implemented the *MBS-high*. In both MBS, all the M-agents in the ecosystem contain at least one TO instance (*D(t)*=1) after 750 instants.

5.4.2 TO_PathSelect

The *TO_PathSelect* task objective implements a pre-planned itinerary to be followed by the TO. The path is described in the TO's `init()` method (see Section 3.6.4). In order to evaluate the TO's performance, we propose to introduce a new metric called *path completion* which indicates how many pre-specified nodes have already been visited by the TO, compared to the total itinerary length. Figure 5-14 shows the number of instants required by the TO to reach its destination.

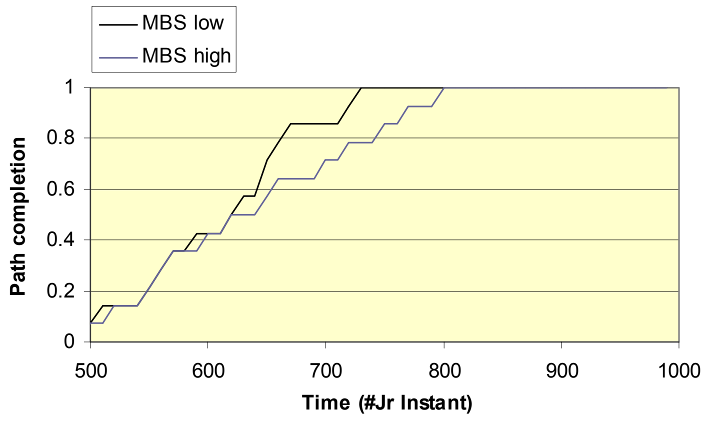


Figure 5-14. Path completion over time

The simulation results clearly show that the *MBS-high* requires more instants than *MBS-low* to reach its destination: as we have seen, M-agents, and therefore task objective instances, spend more time within a node because of the interference-absorption loop.

It has to be noted that, in this kind of TO model, the agent context is constantly equal to 1; the cooperation strategy guarantees that only one TO instance survives.

5.4.3 TO_ExhaustivePathFinder

The *TO_ExhaustivePathFinder* task objective is devoted to the extraction of all cycle-free topology subsets, or possible paths between two nodes given in the `init()` callback (see Section 3.6.5). Since each TO instance follows a different path, the co-operation mechanism does not step in. The blackboard is used to store internal knowledge of each TO instance and thus allows a systematic path investigation.

In *TO_ExhaustivePathFinder*, a simple test has been added in order to extract the longest path; this test simply consists in comparing the number of nodes visited by the task objective instance that has reached its destination, to the current path length. The simulation results are depicted on Figure 5-15.

This TO model, which does not implement any form of the *Dijkstra* algorithm or inter-TO collaboration, allows us to study the ecosystem behaviour with a minimal algorithm.

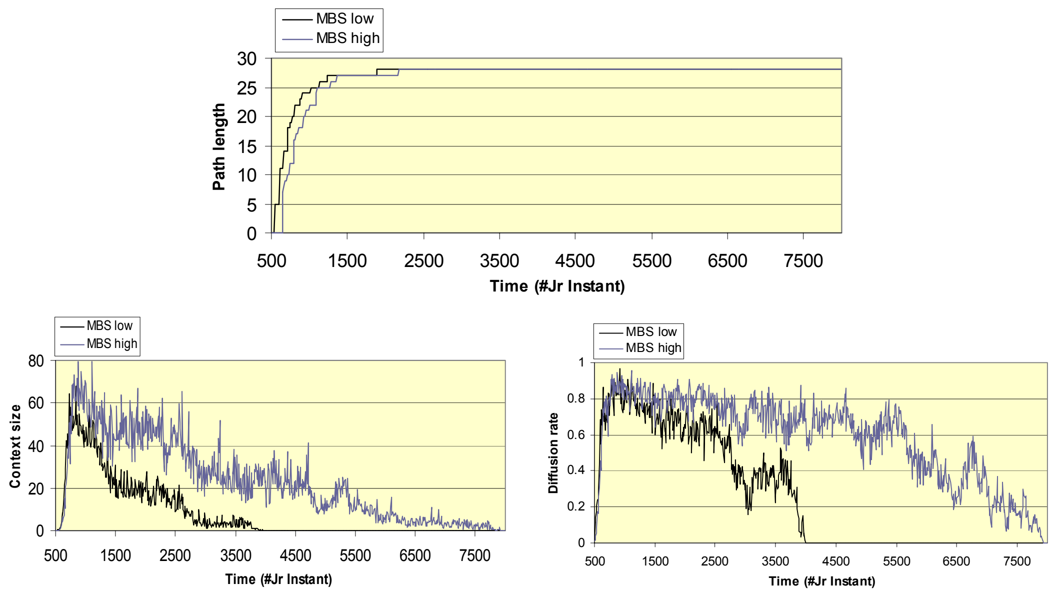


Figure 5-15. Finding all network paths between two nodes

The longest path has been established after 2'500 instants only; additional constraints on the path, like path quality, availability, and so on, will obviously lead to an increase in the convergence time.

The analysis of the agent context size and of the diffusion ratio reveals an important difference between *MBS-low* and *MBS-high*; in addition to the interference-absorption loop effect, we can observe that the general behaviour of the two MBS is significantly different: the systematic cloning mechanism of *MBS-high* implies that the mean context size undergoes important variations, because the TO has strong control over its trajectory and the TO instances are therefore often offloaded and then reloaded; this effect also influences the diffusion ratio.

According to this approach, the number of paths is generally high and strongly depends on the network topology; the number of paths may grow exponentially and considerable time is required to proceed to an exhaustive path search. The cooperation mechanism and additional nodal objects should improve the algorithms and reduce the number of instants; the introduction of *labels* as defined in *Dijkstra* algorithms, for example, provides a simple and efficient solution, so that the TO instance uses the local information to make immediate decisions. Whether it is necessary or not to improve the rapidity of convergence however depends on the type of application: an on-line monitoring process running parallel to working connections is not required to provide immediate responses.

5.4.4 TO_Routing

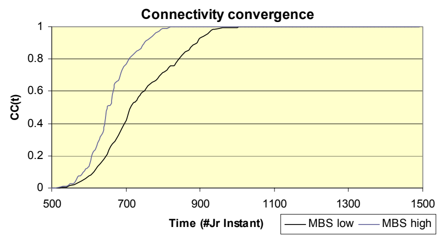
The *TO_Routing* task objective constitutes an important model for routing purposes (see Section 3.6.6); its behaviour has therefore been analyzed for the three network types. This TO is based on a stochastic navigation model and performs regular updates of the routing tables according to the connectivity discovered by the TOs. We assume that a routing table is present in each *port* (see Section 5.1.2) of each node, so that all the possible destinations linked to a certain output link can be determined, as happens with the OSPF routing algorithm. However, link-related metrics are specific to the network technology and they can be easily inserted into the TO body, so that they have not been considered in these experiments.

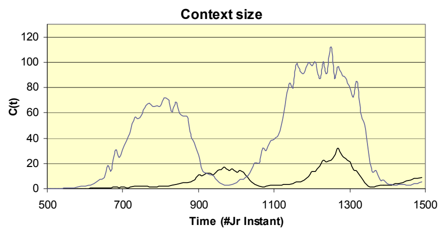
In order to measure the performance of *TO_Routing*, we have introduced a particular metrics called *Connectivity Convergence* (CC). *CC(t)* is defined as the number of entries of all the routing tables divided by the expected number of table entries. In the beginning, all tables are empty and *CC(0)=0*. *CC(t)=1* when all routing tables have been fully completed that is, when full connectivity has been discovered for each node.

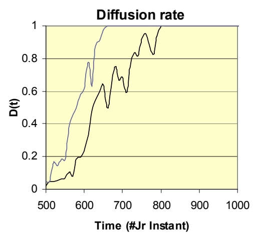
The cooperation strategy of this TO model consists in testing the itinerary and eliminating the slave instance whose knowledge is included in the master instance.

The simulation results are presented on Figure 5-16; connectivity convergence, agent context size and diffusion ratio have been measured for the three networks.

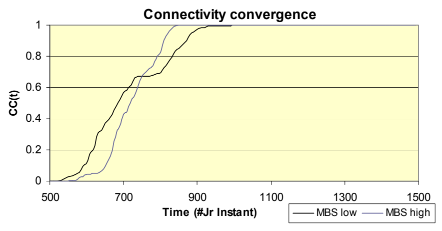
SQUARE **NETWORK**

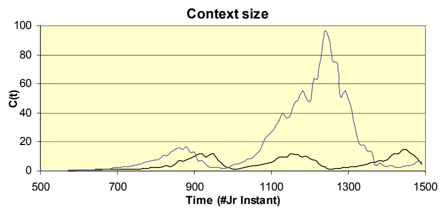


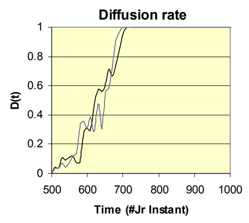




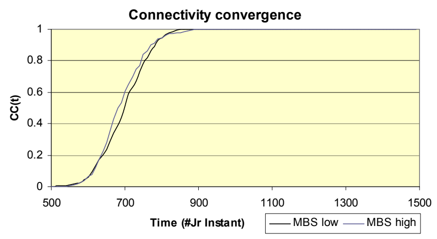
FANTASY **NETWORK**

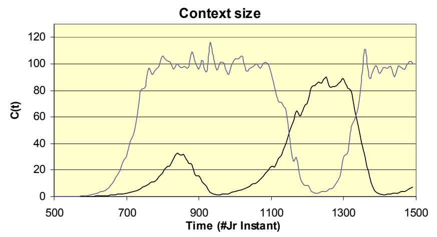






SWISS **NETWORK**





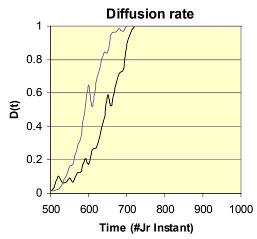


Figure 5-16. Convergence speed, mean context size and diffusion ratio

In this particular TO model, the connectivity convergence is approximately the same in both MBS; *MBS-high* exhibits a better performance in regular networks. In larger networks, like the *Swiss* network, the convergence is similar between the two MBS. The mean context size however differs between the MBS. *MBS-high* reaches the maximum number of TOs during a certain time; the cooperation strategy then involves a *natural* reduction of the size and a re-building of the connectivity information. This renewal effect actually results from the combination of the cooperation strategy with the refreshment of the itinerary, after a certain number of nodes have been reached. The analysis of this task objective reveals in particular that the inter-TO collaboration plays an important role in the regulation of the M-agent size.

Finally, the diffusion ratio increases with similar speed in both MBS. When the maximum has been reached, the M-agent society continuously accounts for at least one task objective instance in each agent context according to the TO lifecycle.

This TO might be improved by the exchange of the itinerary knowledge between two meeting agents, as in the *MITAgent* approach (see Section 1.3.1).

5.4.5 Discussion

As we have seen, the *MBS-high* does not generally exhibit better performance than the *MBS-low*. Two major reasons explain this phenomenon: the interference-absorption loop delays the M-agent in each network node on the one hand, and the agent context may lead to frequent TOs offload, temporarily suspending TO activity because of rapid growth of the mean context size, on the other hand. However, the task objectives are quickly disseminated in the network in the first instants of their lifecycle. *MBS-high* can be thus considered for critical monitoring functions which do not resort to complex knowledge and do not require an extensive exploration of the network. The introduction of the non-deterministic migration present in *MBS-low*, on the contrary, turns out to be more efficient for the implementation of explorative task objectives.

The simulation of TO models has revealed that the agent context size is generally inferior to 100 task objectives, which confirms that *Ecomobile* is particularly well suited to the transfer of small agents that can perform complex tasks in spite of their size.

The combination of two different M-agent societies, the first one implementing *MBS-low*, and the second *MBS-high*, may constitute an interesting approach improving the overall performance of *Ecomobile*. An internal monitoring function could be used in order to make decisions concerning the possible transfer of TOs from one type of society to another. More generally, this kind of behaviour resorts to the use of *social laws* leading to the control of agents by a particular population of agents acting as supervisors.

5.5 Summary

The analysis of the *Ecomobile* infrastructure has led to the development of a simulation framework called GNMT. The network environment relies on a network model inspired from the transport architecture defined by ITU-T and by the ISO model. Each network node implements an *Ecomobile* agency, and all the components belonging to the conceptual framework of *Ecomobile* have been implemented in order to simulate a *real* environment. No agent platform has been considered during the simulation, but the deployment of our infrastructure with *Jade* has been extensively discussed in the previous chapter. The time reference model issued from the reactive approach defines the *instant* during which the reactive behaviours are being executed.

Three kinds of network topology have been proposed: the *Square*, *Fantasy* and *Swiss* networks. Each network presents various topology configuration exhibiting different characteristics (regular/irregular, symmetric/asymmetric, etc.) as well as different network sizes.

The simulation of *Ecomobile* was followed by a behavioural analysis based on MBS-related metrics, such as population size, node and link visit frequency. Different values of waiting time and migration time have been considered and their impact on the ecosystem behaviour has been shown. The particular issue related to the interference-absorption loop in *MBS-high* has prompted us to define a heuristic-based waiting time function in order to achieve a better balance in the link visit frequency. A behavioural analysis of the task objectives has then been performed, based upon TO-related metrics such as agent context size and diffusion ratio. The *TO_Travel*, *TO_PathSelect*, *TO_ExhaustivePathFinder* and *TO_Routing* task objective models, which rely on different navigation models, have been simulated. In this context, specific metrics like path length, path completion and connectivity convergence have allowed preliminary evaluation of the tasks depending on the mobile behaviour scheme. The simulation has also allowed us to identify the parameters influencing the ecosystem behaviour as well as the response of the ecosystem to the insertion of task objectives. *MBS-low*, which contains a non-deterministic migration function, leads to a better performance for task objectives requiring complex knowledge and extensive exploration of the network whereas *MBS-high*, which presents similarities to breadth-first parallel search algorithms, exhibits a rapid diffusion of task objectives during the first instants of the simulation and therefore allows a rapid deployment of monitoring tasks within the whole network, for example.

Simulation on large scale networks still remains to be performed. We are also left with an open question related to the stability of the TO-related metrics: how does the mean agent context size evolve with regards to the network size, in case of the use of *TO_Travel*, for example? The territorial behaviour of M-agents might imply that the network size does not really influence the mean agent context size when the task objective itself does not depend on the number of nodes or on the network topology; this however remains to be proved.

The important amount of simulated objects may also require the simulated environment to run on several machines. In this context, a relatively simple extension of GNMT would consist in coupling it with an agent platform (like *Jade*); the *UPI*-based address would then allow the M-agent to move from one simulation environment to another, by means of the real agency services. In this case, each GNMT instance would simulate a part of the network.

Part III

Application to Optical Networks: A Case Study

Chapter 6
The Optical Transport Network

The improvement of management solutions for the future *Optical Transport Network* (OTN) actually constitutes one of the initial motivations of this thesis aiming at the development of a mobile agent based infrastructure. Transport networks based on *Synchronous Digital Hierarchy* (SDH) or on *Asynchronous Transmission Mode* (ATM) used to be considered as the favourite transmission media networks. Advances in the field of wavelength multiplexing have recently made the optical network the most promising technology for the support of a multitude of services demanding high bandwidth, such as voice, audio, video and Internet traffic. Since the technology of optical devices is constantly evolving, the business and technical costs bound to the replacement of legacy components however make full upgrades of transport networks difficult. Future backbone[1] optical networks will therefore have to cope with a wide range of multi-layer configurations and numerous heterogeneous network components. In this perspective, OTN is regarded as a physical layer able to support several types of clients, such as IP, SDH, ATM or Ethernet.

The incredible growth of the bandwidth supported by a single optical fibre, as well the complexity of future IP/WDM mesh networks, however require the implementation of efficient mechanisms for the management of OTN, especially as far as reaction speed in case of fibre break and/or component damage are concerned. Network survivability has to be considered in a context of increased competition and deregulation as well as high expectations from the network users. In addition, the complexity of multi-layer technologies and increased network capacities places stringent constraints on the network capabilities to recover from any type of failure [DWY99].

In spite of these critical issues related to the survivability of OTN, the emergence of advanced optical components, such as optical switches or transponders, allow the implementation of complex networking functions into the optical domain without the recourse to any optoelectronic conversion. Such *transparent* optical networks also promote the development of new value-added services and lead to the deployment of an attractive *intelligent* transport network, as long as they are endowed with appropriate management solutions.

In the third and final part of this document, *Ecomobile* is applied to the management of optical networks. Particular emphasis will be laid on the semantics of places, which are defined so that efficient TOs can be implemented with respect to the physical constraints of optical transport networks. This chapter presents an overview of optical components and ongoing efforts in the field of network management. Chapter 7 will be devoted to the development of a new value-added service which can easily be implemented into task objectives.

In the present chapter, we do not claim to cover all the problems related to optical networks, but we are trying to focus on a range of concepts that have already been considered in our research in order to implement an *active* optical network management. After having introduced the basic optical network

[1] The backbone or core network is made of "large" transmission lines forming a major pathway within a network and carrying data gathered from the access or edge network that interconnects with the core network.

components, we shall present an overview of ongoing efforts towards the development of management techniques applied to OTN.

6.1 AN OVERVIEW OF OPTICAL NETWORK COMPONENTS

The architecture of OTN has initially been defined by ITU-T [G872_99] according to the generic functional architecture of ITU-T (see Section 5.1.2). Briefly, OTN is defined by three sub-layers which are the *Optical Channel* (OCh) layer, the *Optical Multiplexing Section* (OMS) layer and the *Optical Transport Section* (OTS) layer. The OCh layer can be regarded as a server layer on which any kind of rate-independent clients (e.g. IP, SDH, ATM, etc.) can request connections between two nodes; each client is associated to a single wavelength (or channel). The OMS layer guarantees the integrity of a wavelength group (λ-multiplex) and the OTS layer is the interface between the OMS and the physical medium (see Section 6.3.1).

The following sections describe the main functionalities in optical network technology, which are *Wavelength Division Multiplexing* on the one hand and the optical nodes on the other hand.

6.1.1 Time and Wavelength Division Multiplexing

Time Division Multiplexing (TDM) allows several lower-speed data streams to take place in a high-speed data stream. Today, the highest transmission rate in commercially available systems amounts to about 10 Gbit/s [RS98]. The rate of the multiplexed stream can be increased via an approach known as *Optical Time Division Multiplexing* (OTDM), which allows the multiplexing functions to be fully performed in the optical domain. The resulting multiplexed stream should increase to the level of rates close to 250 Gbit/s. This technique, however, has not reached the stage of commercial implementations yet.

Wavelength Division Multiplexing (WDM) allows several wavelengths to be multiplexed onto a single fibre. Wavelengths have distinct frequencies and do not interfere with one another provided they are kept sufficiently far apart. WDM can currently support up to 160 wavelengths, each of them transporting a 10 Gbit/s data stream.

TDM and WDM are complementary approaches; while WDM leads to the increase of the transmission capacity by means of multiple channels at different wavelengths, TDM allows several low-speed data streams to take place in high-speed channel.

6.1.2 Optical Nodes

Optical network components can be classified according to four networking functions which have an impact on management functions, namely add-drop multiplexing, wavelength routing, wavelength conversion and wavelength switching. These functions[1] are depicted on Figure 6-1. Although they are all represented on a static basis for comprehension purposes, they can exhibit time-dependent dynamic behaviour.

The *Optical Add-Drop Multiplexing* (OADM) function results in the insertion and removal of a specific wavelength from the optical fibre. OADM is typically used between backbone and access network.

[1] In this chapter, we do not take into account optical network devices which are devoted to signal processing, such as amplifier or regenerator. Details about such components can be found in [RS98].

The *wavelength routing* function (λ-routing) allows wavelengths to be routed from input fibres to output fibres. Routing a wavelength actually involves routing the complete set of connections embedded in the wavelength. A routing of lower connection granularity requires a conversion to the electronic domain so that TDM functions can be performed. The mechanism which consists in placing client connections in a wavelength is also known as *grooming*.

The *wavelength conversion* function (λ-conversion) is performed via a transponder which converts a wavelength from a certain frequency to another frequency. This function serves adaptation purposes when interfaces are restricted to particular wavelengths.

Finally, the *wavelength switching* function (λ-switching) allows any wavelength to be switched from one fibre to the other; this function is used for fast protection switching, for example.

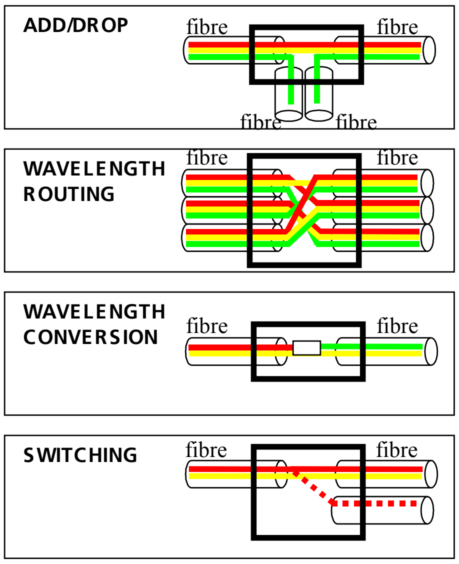


Figure 6-1. Four basic networking functions in the optical domain

The elements realising these functions can operate in the optical domain without requiring optical/electrical conversion. WDM equipment generally implements a concatenation of these functions. Full optical components able to route any input to any output on any wavelength will gradually appear on the market, evolving through intermediate ranges of limited equipments. The switching matrix is still limited to the order of 32 x 32 optical wavelengths and the optical transponder does not provide full conversion between all the wavelengths. The heterogeneity of optical components exhibiting different capabilities in terms of optical functions leads to particular constraints, such as the wavelength continuity constraint, according to which a client connection has to follow one and only one wavelength along the optical path. This kind of constraint induces potential blocking problems on the allocation of client demands. Optical nodes limitations can however be overcome under certain conditions and for particular network topologies; optical rings for example, which are particularly well suited to build survivable networks, support limited wavelength conversions [RS97].

When optical nodes are able to perform functions fully in the optical domain, the nodes, or the network for all the nodes, are known as *transparent* nodes or as a *transparent* network, respectively. On the contrary, nodes requiring optoelectronic conversion are known as *opaque* nodes; they are generally quite expensive and also hamper the evolution towards new services; they are also bit rate and format specific.

The ability of optical nodes to perform networking functions within the optical domain enables optical paths to be allocated and routed in the OTN. We now introduce the resource allocation problem specific to optical networks.

6.2 The Routing and Wavelength Assignment Problem

We define the *lightpath* as the optical path used for a client connection, i.e. the path between an optical source node and an optical destination node. A lightpath can rely on one or several wavelengths along the way. The *Routing and Wavelength Assignment* (RWA), which is concerned with the allocation of lightpaths in optical networking, is a central problem in the management of OTN.

RWA can be divided into two sub-problems: the *routing problem* that is, discovering the route between two optical nodes, and the *wavelength assignment problem* that is, finding the best wavelength allocation along the discovered route. Routing in the optical layer differs from routing in other data networking technology on three main aspects [SCT01]: the network elements are only reconfigurable under specific constraints, i.e. adaptation to specific wavelength frequency and different connectivity levels; the transmission impairments or linear effects, such as dispersion or nonlinear effects with crosstalk between channels, may render certain routes unusable; finally, diversity is defined as the relationship between lightpaths: two lightpaths are said to be diverse if they have no single point of failure. Diversity is influenced by the fibres themselves, by wavelength topology and by location. Furthermore, two circuits may be considered as diverse in one application and not in another. It consequently appears that the routing problem in the optical layer imposes additional constraints which are related to the physical environment.

From a theoretical point of view, the combination of the *routing problem* with the *wavelength assignment problem* leads to a solution minimising the number of wavelengths required in an optical network for a given traffic matrix. Both sub-problems are proved to be difficult to solve (NP-complete) and require specific heuristics. The RWA problem can actually be either *static* or *dynamic*.

The static RWA is originally considered during the *network design and planning process* when business costs are at stake. Most of the algorithms proposed to solve RWA are therefore based on a centralised architecture according to which the entire network topology is known in advance. RWA requires a traffic matrix, which is normally issued by statistical data. An overview of algorithms and heuristics as well as the mathematical formulation of the static RWA problem can be found in [E709_99]. At this stage, no consideration about dynamicity is included: the working and the protection paths are established in a similar way, which remains possible as long as the traffic matrix does not significantly differ from reality and the client demands are static. As soon as the client demands become dynamic and the protection requirements begin to differ from one customer to the other, the protection paths and the working paths have to be computed by means of a dynamic version of the RWA. In this case, since the optical network including lasers and fibres is already deployed, the major issue is the blocking problem, which occurs when there is no available lightpath or wavelength to satisfy a client's demand.

Different client layers with various topology, such as SDH, ATM, GbE, or IP, will be supported by OTN. Therefore, the RWA problem must be adapted so that OTN can take the client requirements into account and deal with the client, or *virtual*, topology. Generally speaking, no satisfactory solution to the RWA problem combined with the requirements of the virtual topology has been proposed yet.

While the static RWA has already been studied for many years, the development of dynamic and distributed algorithms for RWA is still at an early stage [ZJS+01]. According to different approaches, it has been suggested to combine simultaneously routing and wavelength assignment [KL02], or to separate the routing problem from the wavelength assignment problem as proposed by the *Distributed Relative Capacity Loss* (DRCL) algorithm [ZJM00]. The DRCL approach does not make assumptions concerning the routing algorithms but provides an interesting distributed algorithm for the wavelength assignment part, which is mainly based on the static version (RCL).

The complexity of dynamic RWA supporting multiple constraints complicates the realization of a fully decentralised network management. Although the ongoing efforts towards the management of OTN are mainly driven by IP packet-based client layers, the full integration of various client types also remains an important challenge.

6.3 Managing Optical Networks

Different approaches have been considered for the management of OTN. The first standardization efforts in this area have obviously been conducted by ITU-T, leading to the integration of optical components into the TMN framework (see Section 1.1.1) and the definition of corresponding GDMO objects representing the manageable resources of OTN, i.e. the optical network elements [G874_02]. The TMN approach introduces a separate communication channel known as the *Data Communication Network* (DCN), which isolates the management information flow from the transport network itself. As we have seen in Chapter 1, this approach mainly serves a centralised network management.

A TDM-based *Embedded Communication Channel* (ECC) can also be introduced within OTN itself in order to transport management information from one optical node to the other, and finally to the manager. This mechanism has been extensively used in SDH [Sat96], for example, according to which some bytes of the section overhead are reserved for management purposes. The processing of these management bytes therefore requires a TDM function. A specific protocol can be implemented by means of the ECC in order to ensure the information exchange between the nodes. This approach, which is currently being considered in OTN for wavelength management (OCh layer), leads to the development of the *digital wrapper* (see following section).

The *Optical Supervisory Channel* (OSC), which can be considered in addition to these management techniques, corresponds to a particular wavelength of a λ-multiplex that transports overhead related to management information regarding multi-wavelength optical signals. The OSC is therefore completely separated from client wavelengths.

Besides, ongoing efforts within ITU-T and IETF are driven by business requirements for the effective integration of IP networks with OTN, which is also known as IP over WDM (IP/WDM) and leads to the development of the *Automatic Switched Optical Network* (ASON) for the ITU-T and of the *Generalised Multi-Protocol Lambda Switching* (GMPLS) framework, for the IETF. These technologies strongly rely on a general ECC-based technique according to which on-line management functions require the implementation of particular signalling-based protocols.

In the perspective of a decentralised, flexible, scalable and self-adaptable management of OTN, we shall finally consider optical network elements as active optical nodes. We believe that the combination of *Active Network* technology with optical nodes constitutes an innovative approach.

6.3.1 The Digital Wrapper

The digital wrapper has been introduced by Lucent Technologies[1] at the end of the nineties as a way to provide a channel-associated optical channel overhead in OTN [NB99]. The proposal was submitted and approved by ITU-T and serves as a basis for the development of the OTN *Node Network Interface* (OTN-NNI)[2]. Although the digital wrapper is fully integrated into the optical node, it still defines a TDM frame structure composed of three main parts: the *overhead*, the *payload* and an optional *Forward Error Correction* (FEC) code, as depicted on Figure 6-2, which also presents the three section layers with the client layers.

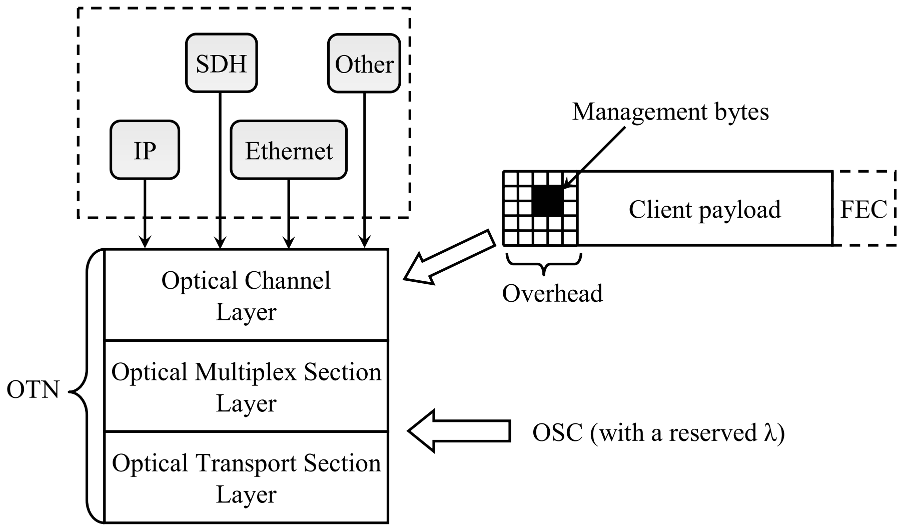


Figure 6-2. The three section layers defined by ITU-T and the digital wrapper

The *overhead* is composed of several bytes reserved for management purposes as well as information related to the payload; it also contains unspecified bytes for future usage. The *payload* corresponds to the client data injected into the wavelength. The FEC is responsible for detection and correction of signal impairments occurring during the transmission; a regenerator will detect eventual malformed frames. The digital wrapper is not bound to any specific client type and thus promotes the realization of a transparent optical network. The only restriction concerning the payload is that the client signal must be a constant bit-rate digital signal.

Certain bytes of the digital wrapper overhead can be used, without any restriction, for a dedicated ECC, in exactly the same way as in SDH management. This ECC could thus be used by *Ecomobile* for the transport of M-agents (see Section 7.2).

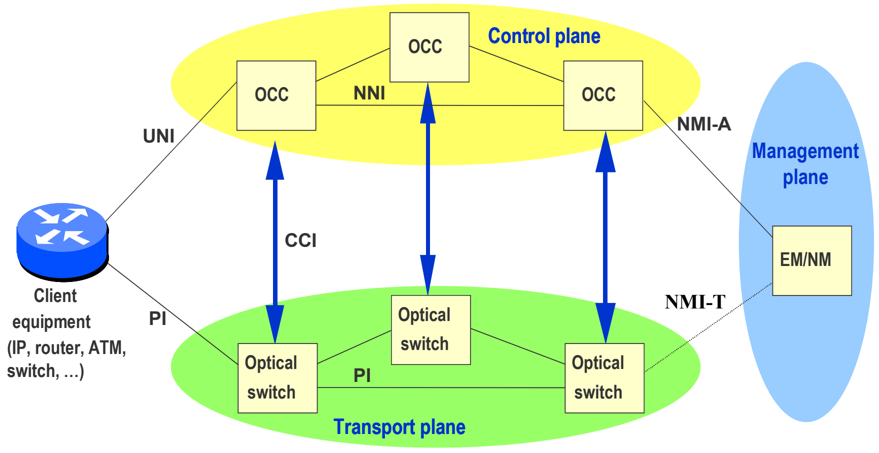
6.3.2 The Automatic Switched Optical Network

The *Automatic Switched Optical Network* (ASON) [ITU00] constitutes a significant step towards decentralised network management. It was originally conceived to dynamically allocate permanent and semi-permanent circuit-switched lightpaths within OTN via a specific user interface, and to support

[1] http://www.lucent.com

[2] The OTN-NNI is specified in draft recommendation G.709.

automatic routing of lightpaths. The main interest of this approach is to reduce the considerable time devoted to provisioning in OTN that is mainly due to bureaucratic processes and *manual* human intervention at several points in the resource configuration. An automatic control can be performed by ASON, which spreads out topology knowledge and information related to capacity and availability within the network. The proposed implementation of ASON relies on two networking planes: the *transport plane*, corresponding to the OTN composed of optical switches - or more generally optical nodes implementing the different networking functions described in Section 6.1.2 - and fibres, and the *control plane*, which enables the transport of signalling between *Optical Connection Controllers* (OCCs). The logical architecture of ASON is depicted on Figure 6-3.



CCI	Connection Control Interface	**OCC**	Optical Connection Controller
NMI-A	Network Management Interface for the ASON Control Plane	**PI**	Physical Interface
NMI-T	Network Management Interface for the Transport Network	**UNI**	User to Network Interface
NNI	Node to Node Interface		

Figure 6-3. The ASON Management planes

Signalling between optical switches and client equipment is exchanged through the *Node-to-Node Interface* (NNI), while service requests can be initiated by the customer himself via the *User Network Interface* (UNI).

While central functions can take place in the *management plane*, the management logic can be partially distributed over the OCC, so that decentralised network management can be achieved at different degrees. When most of the management logic resides in the *management plane*, ASON can be regarded as a distributed signalling-based infrastructure conceived for OTN, which does not really differ from a traditional SNMP/CMIP approach (see Section 1.1.1).

Although ASON aims at enhancing OTN with advanced management functions and therefore promotes the development of new "intelligent" *optical* services, such as fast provisioning, optical VPN, and on-line protection and restoration services, the ongoing development of ASON mainly focuses on the definition - and consequently on the standardization - of user and network interfaces. The elaboration of distributed algorithms specific to the control plane functions, such as automatic topology discovery or efficient multi-constrained routing, for example, remain an important issue in the development of ASON [R^+01]. Routing is performed on optical channels with a high granularity (high bit rate). More generally, the RWA implementation in the context of ASON still raises challenging issues due to the emergence of new customer-centric and topology-related requirements.

The *Generalised Multi-Protocol Label Switching*[1] (GMPLS) framework proposed by the IETF, which was issued from the Internet world, follows the same logical architecture as the ASON. GMPLS consists in applying to the optical switches the MPLS control plane techniques used for IP networks, and in developing IP routing algorithms to manage lightpaths in the optical network. GMPLS is therefore strongly oriented towards IP over OTN (IP/WDM) and does not rely on an intelligent *transparent* OTN like ASON.

OVERLAY AND PEER-TO-PEER MODEL

Nowadays, optical networking obviously witnesses a trend towards the acceleration of IP/WDM networks development in order to support the constantly growing number of Internet services. In this context, GMPLS constitutes an efficient intermediate step towards integrated management of both IP and optical networks through the definition of appropriate control planes. Still, worldwide spread IP networks remain more simple to manage than optical networks from the point of view of the multi-carrier: while IP routers can perceive the topology outside their management domain and interact fully with foreign routers, optical switches require to be managed within an operator or *carrier* domain which cannot interact with foreign optical switches. This is the reason why the architectural model based on two different control planes managing the routers and the optical switches respectively actually leads to the definition of two signalling architectures known as *overlay* model and *peer-to-peer* model.

According to the *overlay* model, the routers of the user domain "ask" the optical network for a connection or for other optical services according to a client-server model; OTN acts as the server of the provider domain whose different clients, such as IP, ATM or SDH, can access OTN through the UNI by means of appropriate signalling. The NNI enables signalling between the different optical switches within the OTN. The overlay model presents serious advantages for network operators because the user domain does not need to be aware of the optical topology, which can belong to different carriers. The router devices must however be compatible with ASON (or GMPLS) UNI provided by the carrier because of the two distinct control planes being used. The overlay model therefore implies restrictions from the network operator point of view in the choice of the network equipments. In addition, the limitation of network knowledge may result in a sub-optimal usage of network resources, as it is the case in any network configuration involving several layers.

According to the *peer-to-peer* model, routers and optical switches share a uniform and unified control plane. In this case, the routers are endowed with a thorough knowledge of available OTN resources, which may raise important problems related to the interaction of multiple operator domains. The quantity of information to be processed and algorithms resulting from the integration of IP/WDM processing into a single network component also lead to increasing complexity in the network. It therefore appears that this kind of architecture entails serious drawbacks in terms of scalability.

DISCUSSION

The ASON and GMPLS approaches strongly rely on signalling and consequently entail important restrictions concerning the evolution of an intelligent OTN; since the elaboration of new value-added services is still at an early stage, it is difficult to determine the kind of services and profiles which will be required by the future OTN. The introduction of a SLA-based management, for example, could make the

[1] ftp://ftp.isi.edu/in-notes/search.ietf.org/internet-drafts/draft-ietf-ccamp-gmpls-architecture-02.txt and http://www.lightreading.com

management within the control plane much more complicated than expected. Besides, the signalling-based approach requires network equipments to implement the complete protocols and services at design time. Each extension or modification of the signalling will therefore require considerable changes in the network infrastructure with important business-related costs. Signalling therefore turns out to be inadequate in the case of a flexible and extensible OTN.

Unlike the digital wrapper, that enables a fast processing of the management information because of its integration to the optical channel layer, the above described approaches do not specify the frame structure of the optical channel. According to a control plane approach, it is therefore difficult to imagine that every manufacturer will adopt a similar implementation of the optical channel.

A possible alternative solution would be to replace signalling by CORBA interfaces controlling the automatic setup of connections, alarm propagation, billing support, SLA management, etc. [Ger00]. This approach could however lead to a throwback to a traditional platform-centred TMN-like approach.

6.3.3 Active Management

A final technique applied to the management of OTN has to be described: in the context of our work, *active management* (see Section 1.4) in optical networks is an innovative approach emerging from the development of *Ecomobile*. Active management actually combines principles of the active network technology with the mobile agents paradigm developed in the previous chapters. Active management favours a decentralised network management characterized by fast reaction to external changes and by the capacity to deal with the physical constraints related to the infrastructure, so that it perfectly suits a transport network such as the OTN. Active management is not meant to replace architectures like ASON or GMPLS, but should rather be seen as a complementary approach enhancing the physical layer with advanced functionalities.

Two distinct steps can be distinguished in the development of active management: the development of fully distributed algorithms implementing most of the management logic within the network nodes is followed by the implementation of these algorithms into the active optical nodes. To the best of our knowledge, mobile agents had not been considered in optical network management when we began this work. Nevertheless, a solution to the RWA problem relying on ACO (see Section 2.4.1) and therefore similar to *AntNet* has been proposed in [VS99]; the possibility of actually implementing this approach however still remains an open issue; the algorithms require a considerable processing time and resort to a global update of the pheromone which is not compatible with a decentralised management approach.

The techniques related to active management will be developed and applied in Chapter 7.

6.4 SUMMARY

OTN is composed of a wide range of heterogeneous network devices which have the capability to perform networking functions fully in the optical domain, i.e. without optoelectronic conversion. Relying on wavelength division multiplexing, the optical nodes can process several wavelengths on a single fibre. The transport of several Tbit/s of data issued from different client layers on a single fibre and the implementation of networking functions into the optical domain make network management systems critical components of the transport network. The rapid evolution of the optical network domain consequently implies a decentralised management approach.

The physical impairments and connectivity restrictions on transparent optical nodes make the RWA problem a central issue in the management of circuit-switched lightpaths. The problem becomes even more complex with the emergence of new optical services which have an impact on routing and wavelength assignment. While the static RWA has proved to match the planning process of optical networks, the distributed implementation of the dynamic RWA dealing with dynamic traffic and on-line re-configuration is still at an early stage.

The digital wrapper has been proposed by *Lucent* in order to improve the management of OTN; it defines a TDM structure in three parts which allows the optical channels to be managed independently from the client layers. The digital wrapper overhead encompasses a portion of bytes dedicated to management purposes and therefore makes the definition of ECCs possible. An optional FEC allows the signal to be monitored and re-processed within the regenerators. The development of a digital wrapper is also associated with the elaboration of the OTN-NNI.

A logical architecture for an intelligent OTN has also been proposed by the ITU-T in the specification of ASON. This approach introduces a separation between the physical layer (transport plane) and the management functions (control plane). This separation involves the deployment of optical connection controllers for the implementation of the management logic. ASON is mainly characterized by the recourse to particular signalling between optical nodes themselves through the NNI and between optical nodes and the user through the UNI.

In order to deal with growing demands and requirements emerging from the transport of up-and-coming Internet applications over the optical fibre, the IETF has proposed to extend the IP/MPLS framework to the optical network by adapting the algorithms used within IP routers to the optical switches.

Chapter 7
Towards Active OTN Management

In Chapter 1, we have presented an overview of the basic components specific to the optical transport network which constitutes the main target application of the middleware developed in this thesis. The new generation of optical nodes encourages the development of new advanced services within the transport network, steadily transforming the traditional point-to-point OTN into a service-enabled *Automatic Switched Optical Network* (ASON). The diversity and complexity of customer requirements as well as limitations related to optical devices however bring about new technical hurdles in the management of these services.

The introduction of specific control planes, such as those proposed by ASON or GMPLS for the management of resource allocation and routing in the optical network would certainly favour the development of decentralised management systems; they would provide the NMS with information related to traffic engineering and network topology. However, the implementation of advanced algorithms achieving complex RWA in a dynamic situation still remains an important challenge in the management of OTN because of emerging requirements issued from a transparent multi-layer OTN and new customer requirements based on service level agreement. It also has to be noted that a signalling-based approach introduces potential interoperability issues between network devices issued from different vendors, and that the management logic must be implemented at the design phase.

Active management, which constitutes a new paradigm liable to overcome current limitations of optical nodes, combines code migration and mobile agents on the one hand, with an active network enhancing optical switches with the necessary infrastructure hosting dynamic code on the other hand. An approach based on active management is obviously not straightforward from the business model point of view; the management logic is subject to the control of the network operator, whereas the hosting environment must be implemented into active nodes by the manufacturer according to industrial standards.

In this chapter, we propose to examine how *Ecomobile* can be implemented efficiently within OTN and how its particular computational model leads to creation of new value-added services. After having introduced the OPTIMA project, which constitutes the predicted follow-up of this thesis, we will examine a possible way to deploy our mobile middleware in the optical layer, new *intelligent* wavelength services will be defined, and an example of an improved protection service will be given.

7.1 The OPTIMA Project

The *OPTical network management with Intelligent and Mobile Agents* (OPTIMA) project[1] aims at improving the network survivability and flexibility of optical transport networks by studying and developing adaptive algorithms based on autonomous software agents. Four main aspects are emphasized: the coordination models based on bio-inspired approaches and collective intelligence (stigmergetic coordination), mobile agent platforms, modelling and simulation.

[1] OPTIMA is currently a project funded by the Swiss National Science Foundation Science (SNF) - project nr. 2100-063717 (August 2001 - July 2003).

OPTIMA should result in a suitable coordination model and a simulation tool interacting with open mobile agent platforms and supporting irregular meshed network topologies, as well as multi-client scenarios (e.g. IP/WDM, AT/WDM) and different failure models.

In this perspective, the development of *Ecomobile*, which provides a mobile agent middleware deployed in active optical nodes by means of a FIPA-compliant agent platform constitutes preliminary work to the OPTIMA project. The computational model leads to the implementation of adaptive algorithms and GNMT provides an adequate simulation environment.

The design of task objectives could constitute one of the important tasks of the OPTIMA project. The RWA problem, for example, could be solved by the development of one TO for the routing problem and one TO for the wavelength assignment problem. In this context, the *TO_Routing* task objective model proposed in Section 3.6.6, makes the implementation of a distributed algorithm for wavelength assignment such as the DRCL (see Section 6.2) particularly straightforward; this implementation is currently under investigation. Since the TO handles a single *routing* table for each port, or for each link, it can be extended to store the corresponding RCL values.

7.2 DEPLOYING *ECOMOBILE* INTO ACTIVE OPTICAL NODES

In this section, we propose to examine an implementation of *Ecomobile* concepts into OTN. Our basic assumption is that optical nodes encompass an active node environment including a *NodeOS*, an EE (see Section 1.4.1), and an operating system enabling the development of *Java*-based active applications and the deployment of the agent platform (see Section 4.4).

As far as the ASON/GMPLS logical architecture is concerned, *Ecomobile* can be regarded as an intermediate layer between physical layer and control plane, which provides the transport plane with appropriate knowledge concerning OTN; metrics specific to new value-added services such as QoP (see section 7.4.2) can be implemented by means of specific task objectives and give the control plane the information used for subsequent path allocation or reconfiguration.

7.2.1 Optical Agents

The implementation of mobile agents within the optical layer has led us to define *optical agents* [RS00] (or λ-agents) which are supposed to be integrated into the optical frame of the optical channel layer (see section 6.3.1). From the point of view of agent communication, we actually propose to use a portion of the digital wrapper overhead to install a specific ECC devoted to the ACC. An adequate MTP, resorting to an IP-based HTTP protocol for example, can then be installed so that *Ecomobile* agencies can exchange ACL messages. In the context of *Ecomobile*, optical agents correspond to M-agents[1].

With regards to the active node, information concerning the overhead and intended for *Ecomobile* can be extracted from the digital wrapper, transformed into *active packets*, and sent to the active application.

Two classes of optical agents dedicated to OTN layers have been proposed to fit different applications: the *λc-agent* (channel agent), which is associated to a specific wavelength, suits wavelength-related functions, while the *λm-agent* (multiplex agent), which is associated to a single *λ-multiplex*, is appropriate for fibre-related functions like "finding disjoint routes". From the point of view of *Ecomobile*, this differentiation leads to distinct M-agent families evolving in the optical network in different ways and reflecting the network state at different granularity levels.

[1] In our implementation, the M-agent class constitutes an extension of the *LambdaAgent* (see Section 3.3).

In order to define a simplified approach, it is possible to propose a single family of M-agents which evolves at the lower granularity level and is therefore similar to *λc-agents*, and to control the navigation within task objective by means of a local connectivity matrix, which implies a representative equivalence between physical environment and ecosystem environment; this context requires the definition of place semantics.

7.2.2 Place Semantics

WDM technology allows the transport of several independent wavelengths in a single optical fibre. We have seen in Chapter 1 that optical nodes can present different configuration types according to the implemented networking functions. The constraints imposed by the node configuration directly influence wavelength connectivity and therefore also routing. Each wavelength may therefore represent different sub-topologies also known as *wavelength planes*. Associated connections within a specific wavelength plane correspond to switching capabilities provided by the optical nodes whereas associated connections between wavelength planes correspond to conversion capabilities also provided by the optical nodes.

According to this representation, each place can be associated with a specific wavelength, as depicted on Figure 7-1.

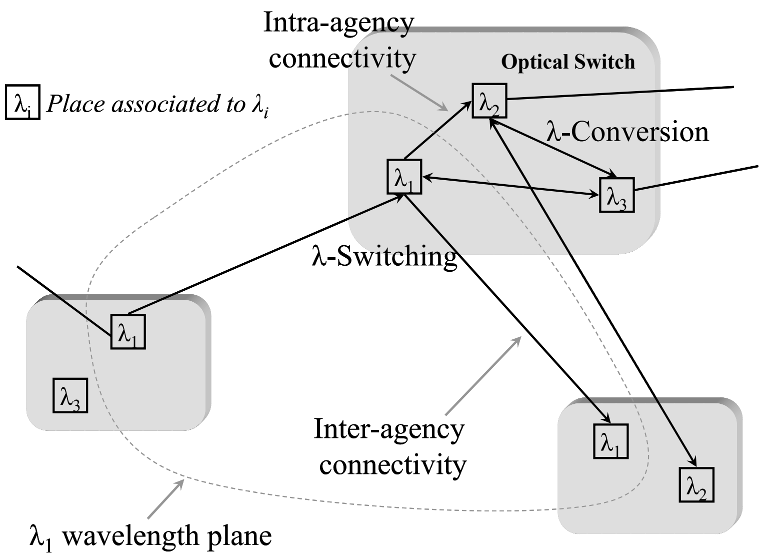

Intra-agency connectivity
Optical Switch
λ_i Place associated to λ_i
λ_2
λ-Conversion
λ_1
λ_3
λ-Switching
λ_1
λ_3
Inter-agency connectivity
λ_1
λ_2
λ_1 wavelength plane

Figure 7-1. Place semantics in the context of the basic optical functions

As far as the connectivity matrix of the *Ecomobile* agency is concerned, we propose to map the intra-agency connectivity onto the connections between wavelength planes and the inter-agency connectivity onto the connections within a single wavelength plane. Let us formally suppose that:

$UPI_a ::= (Node_m, \lambda_i)$ (1)
$UPI_b ::= (Node_n, \lambda_j)$ (2)
$f(UPI_x, UPI_y)$ being the elements of the connectivity matrix (3)

Then,

$f(UPI_a, UPI_b)$ = 1 if and only if ($Node_m$ ∫ $Node_n$ and $\lambda_i = \lambda_j$) or ($Node_m = Node_n$ and λ_i ∫ λ_j), and there is an optical function performing the connection so that ($Node_m$, λ_i) ö ($Node_n$, λ_j).

This kind of mapping matches the networking functions and allows the M-agent ecosystem environment to explore the physical agility of optical nodes in a more efficient way. According to the territoriality paradigm, the occupation of the environment by the M-agent population will also become more adequate when each place is not restricted to a single node; an overstated number of absorption and cloning behaviours can thus be avoided. It also has to be noted that fully tuneable and reconfigurable hybrid nodes allowing any input wavelength to be switched onto any output wavelength are not commercially available yet; optical nodes will gradually evolve from opaque to agile and transparent optical nodes [Rob01].

7.3 INTELLIGENT WAVELENGTH SERVICES

In this section, we present an overview of the most popular optical network services which are currently being developed. At the beginning of our investigations, we defined the term of *Intelligent Wavelength Services* (IWS) to designate the services relying on the *Ecomobile* middleware[1]. We now propose to extend the IWS with *optical services* emerging in the optical networking community, such as optical VPN or on-demand provisioning.

The development of new IWS will give customers new opportunities to take advantage of the optical infrastructure. In this context, the palette of services offered, their quick deployment, and the efficiency in their support constitute key differentiators for network operators [VSN$^+$01]. The introduction of new IWS will also coincide with more sophisticated customer needs. The management of services leading towards the optical Internet consequently relies on the definition of formal metrics and particular attributes which are specific to OTN and which will allow customers and network operators to stand up for their reciprocal commitments. In this context, service differentiation allows carriers to meet their customer needs more satisfactorily; differentiated optical services also enable network providers to achieve end-to-end QoS by means of an optical parameter set capturing the quality and the reliability of the optical lightpath [GNS00]. The key factor of differentiation is consequently becoming more and more important, and it does not only require the verification of *Service Level Agreements* (SLA) between the customer and the network operator, but also between multiple network operators, or simply between the network operator and the service provider. The development of an SLA framework based on attributes driving the network architectural design and including cost, availability, protection switching time and propagation delay [BAG01], is still at an early stage.

7.3.1 The Optical VPN

The optical *Virtual Private Network* service provides a secure and manageable environment allowing a group of clients to fully exploit the flexibility of the switched intelligent optical network. In the optical VPN, customers can make a contract for a specific network resource such as link bandwidth, wavelength, and/or optical connection ports, and these resources can be controlled as if they were customer-owned. Additionally, the customer can specify requirements concerning protection or restoration services. The network provider then performs the optical connection statically or dynamically, taking into account the SLA.

[1] The IWS also constitutes a GNMT package encompassing the OTN classes.

Customer requirements generally correspond to the virtual topology, i.e. a collection of nodes and links managed by client routers (IP, SDH component, etc.) and requiring to be mapped onto the OTN. The customer is normally deprived of any visibility in the underlying optical infrastructure, which remains under full control of the network provider.

Optical VPNs can be used to support a variety of applications, including ISP edge router networks, content delivery among a network of servers, bandwidth trading between carriers and storage WANs for enterprise networking [BTS$^+$01].

7.3.2 On-demand Provisioning

While static bandwidth provisioning in OTN may require a long time (sometimes several days), and is rather dedicated to longer holding time circuit-switched connections, on-demand provisioning or fast provisioning aims at supporting the management of switched connections which have much shorter lifetime [E1116_01]. On-demand provisioning therefore requires the connection establishment time to be as short as possible.

Like the optical VPN, this service may rely on SLA in which the customer specifies his requirements concerning bandwidth and client type, as well as those related to protection and restoration services. In the long term, customers should be able to ask for lightpaths by means of a Web interface.

7.3.3 Protection and Restoration in the OTN

The overall availability requirements can be evaluated to a percentage in the order of 99.999 percent, which implies that the network should not be dysfunctional for more than 6 minutes a year on average [GR00]. Network survivability is therefore a key factor in network design and management. With the introduction of IWS, new protection services can be proposed to the customers according to different service classes. From the network operator's point of view, for example, lightpaths can fall into one of the following classes: some that *must* be protected by the optical layer, some that *must not* be protected, some that are *indifferent* to protection, others that may be protected on a *best-effort* basis, and finally *low-priority* lightpaths that utilize protection bandwidth under normal circumstances and are pre-empted by protection of other lightpaths. In this thesis, particular emphasis has been placed on protection and restoration issues in OTN; we therefore propose to examine important aspects related to this topic in the context of a multi-layer OTN.

There are two possible survivability strategies in optical networks, namely *protection* and *restoration*, which are mainly inspired from the SDH networks [MBN99].

Protection can consist of two different schemes: *dedicated protection* (1+1) or *shared protection* (1:n, m:n). While the former requires disjoint paths, but not necessarily disjoint optical nodes, the latter reserves a certain amount of wavelengths as spare capacity for one or several working paths. An optical fibre can therefore share its wavelength capacity in both working and protection paths. Examples of optically protected rings are the *Optical Channel Dedicated Protection Ring* (OC-DPRing), the *Optical Channel Shared Protection Ring* (OC-SPRing), the *Optical Multiplex Section Dedicated Protection Ring* (OMS-DPRing) and the *Optical Multiplex Section Shared Protection Ring* (OMS-SPRing). In all these protection mechanisms, protection paths are pre-computed, so that the switching mechanism is performed in a short time frame (50ms) whenever a failure occurs.

Unlike protection, *restoration* occurs immediately after the failure. In this case, the discovery of alternative paths may require some time. Restoration is generally proposed as an ultimate way to keep a service running in case of failure when no protection has been set up. The IP layer, for example, provides a robust restoration mechanism based on packet re-routing in case of failure and integrated into the routing protocols.

The dynamic re-configuration capability characteristic of optical switches allows the protection paths to be re-configured on-line during a live connection. Unlike working paths, protection paths can be re-configured even when the running service is not aware of the operation.

In the context of a multi-layer OTN, client layers may have their own protection strategy, like SDH layers, for example. The protection of both client and optical layer improves the overall survivability of the network since failures can be immediately detected and repaired at the correct layer, which limits the propagation of alarms to other layers; this protection scheme also constitutes the only way of avoiding intra-site connection failures between client equipments and optical equipments.

Protection at the optical and client layers however requires particular attention to the mapping of virtual paths onto the OTN topology; inadequate mapping may lead to interoperability issues regarding the protection strategies in both layers; this phenomenon, which is known as *failure propagation* [Cro98], will be examined in section 7.4.

Integrated survivability is a cost-effective solution providing telecommunication services in a multi-layer network environment with differentiated levels of survivability [E718_01]. Multi-layer survivability raises two open issues which can be phrased as follows: what survivability functionality should be allocated at each layer? How should co-ordination between the network layers be defined? In the absence of co-ordination, unstable situations, undetermined network configurations and dynamics may actually occur.

7.4 Differentiated Protection Services

In the present section, we introduce a new value-added protection service issued from *Ecomobile* by implementing a specific task objective. This kind of service is referred to as *differentiated protection service* and allows customers to ask for a certain protection level or *protection quality* for a particular optical VPN [RRS01].

Dissimilarities between VPN topology and physical topology regarding protection strategies may lead to service disruption in case of failure at the physical layer, even if the VPN is still provided with spare capacity. This interoperability problem leads to dependencies between VPN layer and physical layer which are called *vertical dependencies*. The VPN protection paths consequently need to be carefully mapped onto the physical network so that the vertical dependencies can be reduced and propagation failures in the VPN layer can be avoided. The complex *NP-hard* problem raised by this operation has been thoroughly investigated and static heuristic-based solutions have been presented in [CLG00]. The solutions which have been proposed however fail to meet the expectations related to a dynamic environment with fully distributed control.

7.4.1 The Network Model

The network model considered in our study is depicted in Figure 7-2 and consists of two layers, the client-transparent optical VPN, which corresponds to the virtual topology, and the OTN, which represents the physical topology. The client VPN is not limited to a particular network technology; it simply claims to use a part of the underlying resources with restricted access. A VPN is composed of several end-to-end connections that may or may not be protected. In case of protected connections, the set of protection paths corresponds to the protection strategy attached to the connection. A VPN set can actually have several collections of working and protection paths.

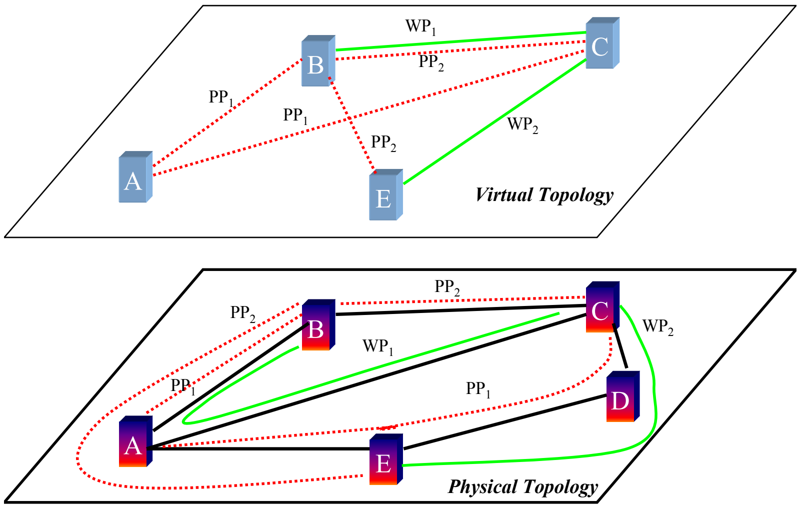


Figure 7-2. Two-layers network model with virtual and physical topology

Various protection levels are considered in both layers. The client can express his protection requirements in the form of a SLA, so that the task objective evaluating the protection configuration will be appropriately influenced in its choice of physical optical paths.

In our network model, the protection strategy consists in supplying the protection path as required in the virtual topology. In case the client requests a SDH Working Path (WP1) between node B and node C, for example, the protection strategy consists in re-rerouting the traffic, using Protection Paths (PP1) via node A, in case of failure on link (B,C).

According to our protection model, the *Protection Paths* (PPs) are set up dynamically. This operation might result in a combination of dedicated and shared protection paths, which means that the PP configuration could use the same fibres and could rely on the same nodes as the *Working Path* (WP), in which case the link would be protected against wavelength channel failure, but not against fibre failure. Intuitively, we can assume that the number of shared nodes and the number of shared links making up the protection path will impact on the *protection quality*. As the protection path configuration is subject to dynamic changes during the connection time, there might not be any protection path available for certain *low-QoP* based VPNs but, in case of failure, a restoration mechanism could be launched in order to preserve the connections.

7.4.2 Quality-of-Protection (QoP) and DPS Formulation

In the scope of our new protection service, the protection quality can be evaluated by means of the *Quality-of-Protection* (QoP) metric, a temporal metric for on-line assessment. In order to elaborate a QoP metric, we first need to define the entities of the optical layer which are subject to failures. The fibre actually reveals to be the least reliable component in the system [GR00], including the optical amplifiers deployed along the fibre. The other critical component is the entire node, which is composed of numerous transponder cards to interface with client layers and of all the fabric necessary for the realization of networking functions. This is the reason why we propose to define the following factors involved in the QoP metric: the amount of shared nodes of the protection path, the amount of shared links of the protection path, the *horizontal dependencies* between multiple domains, and the *vertical dependencies* between the physical layer and the client layer. Horizontal and vertical dependencies are depicted on Figure 7-3.

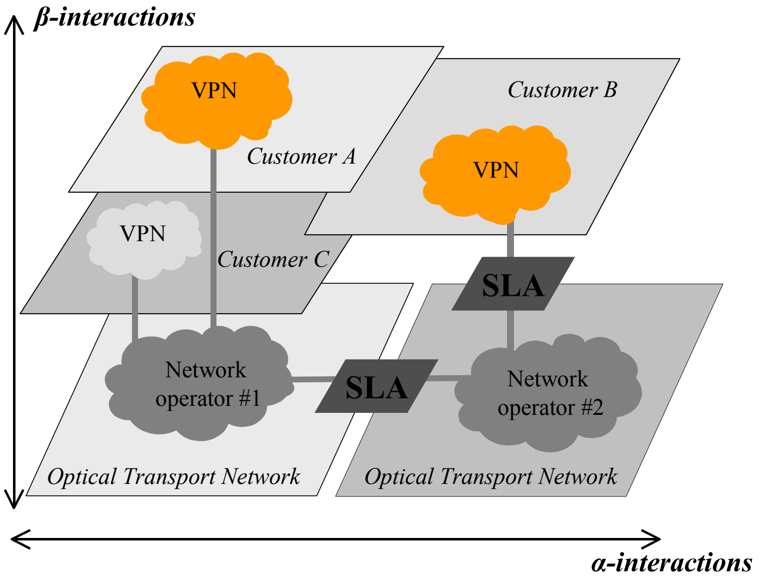


Figure 7-3. Two-dimensional interactions influencing the quality of protection

In order to calculate the QoP, we adopt a global network view involving interactions between multiple layers, as well as between multiple network operators. We assume that the business-related agreements are defined by means of SLAs.

We propose the following definitions:

i	End-to-end client connection (WP and PP)	(1)
D	Total of domain peers	(2)
$SN_i(t)$	Shared nodes at time t of connection i	(3)
$SL_i(t)$	Shared links at time t of connection i	(4)
ω	Weight associated to shared nodes vs. shared links (given by the client)	(5)
α_d	Horizontal dependencies between the domain peer d (currently $\alpha_d = 1.0$)	(6)
$\beta_i(t)$	Vertical dependencies associated to connection i	(7)
TL_i	Total of fibre links along the path of connection i (WP+PP)	(8)

SN(t) (3) is the number of shared nodes along a protection path, *SL(t)* (4) the number of shared links (fibres) along a protection path. The shared link implies that the protection and working paths are on the same link using different wavelengths. A fibre break obviously entails the loss of all connections going through the fibre. Shared links are acceptable only in the case of optical channel malfunction within the optical node.

$SN_i(t)$ and $SL_i(t)$ are time dependent and the configuration of protection paths can change over time. If $SL_i(t)$ is constantly equal to 0, it means that the protection path follows a dedicated (1+1) protection strategy. Moreover, if $SN_i(t)$ is equal to 0, the protection path uses disjoint nodes from those used in the working path. Horizontal dependencies, which are more difficult to evaluate because of the role played by business parameters, will not be detailed in this thesis.

Weight ω (5) gives the customer the opportunity to influence the importance given to shared nodes and links belonging to the protection path. This value may be determined from the customer's own experience with a network operator.

0 § α § 1 (6) is to be considered in case of multi-operators or multi-domains interactions. α is a constant value and refers to horizontal interactions. This factor can be used to assess the protection quality when several domains have to be traversed by an end-to-end connection.

0 § β § 1 (7) is a time function assessing the respect of constraints related to the interoperability between the protection strategies of the client layer and OTN. β depends on client demands and can change dynamically according to the current configuration of protection paths. It is related to multi-layer interactions. A primary approach towards this function has been proposed in [RoRS01].

Let us define $P_{NetworkFailure}$ as the probability of a failure in the OTN layer (fibre cut, node malfunction) and $P_{ServiceFailure}$ as the probability of a service disruption.

The *Dynamic Protection Set-up* (DPS) problem is stated as follows:

DPS: *given an end-to-end connection, find a protection path that maximizes QoP(t)*, with

$$QoP_i(t) = \alpha\,(\omega\, e^{\frac{-SN_i(t)}{TL_i - sn_i(t) - 2}} + (1-\omega)\, e^{\frac{-SL_i(t)}{TL_i}})\ast \beta_i(t) \qquad (9)$$

with $\alpha = \frac{\sum_{d}^{D} \alpha_d}{D}$

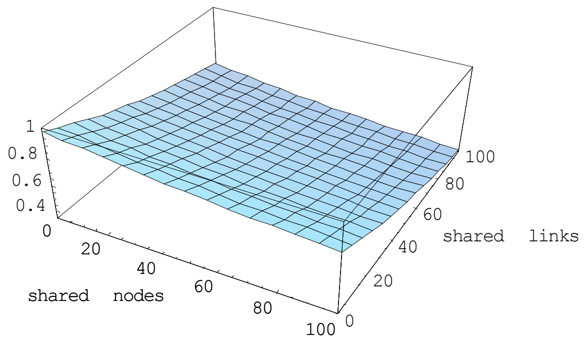


Figure 7-4. Evolution of the QoP according to the number of *SN(t)* and *SL(t)* (α,β = 1.0, ω = 0.3)

0 § *$QoP_i(t)$* § 1 (9) defines the quality of protection of the connection *i* at time *t*.

From the customer's point of view, the protection requirement can be expressed as the probability that the service will be disrupted. This can be simply computed as follows:

$$P_{ServiceFailure} = P_{NetworkFailure} \ast (1 - \text{QoP})$$

The $P_{ServiceFailure}$ value, which may be part of the SLA between customer and network operator, can be monitored in a continuous way. If QoP is equal to 1, the protection is maximal and the probability that the service will be interrupted is reduced to 0. Without any protection, QoP is equal to 0, and the probability of an interruption is equal to the probability of an intervention at the OTN layer.

The QoP of each individual connection in the VPN set is continuously assessed whenever protection paths emerge from the mobile agent ecosystem; the quality of protection can then be compared with the requirements specified in the SLA. Since our primary objective is to devise an adequate architecture and implementation for distributed solutions matching scalability and survivability requirements, the QoP optimisation for the total VPN set has not been investigated yet.

7.4.3 DPS-oriented Task Objective

According to the definition of QoP and the related DPS, we now propose to design the task objective implementing the new protection service. We assume that a customer wishes to establish a VPN connection and therefore requires a working and protection path according to a private protection strategy; these requirements will be embedded in the task objective.

The *TO_QoP*, which is outlined in Table 7-1, is mainly inspired from the *TO_Travel* and *TO_PathSelect* TO models presented in section 3.6. The TO may be launched at any location in the network. The first objective of the TO instances will consequently consists in finding the source node and then in discovering a working path accommodating the working strategy constituted of a set of nodes. At this moment, the cooperation strategy may consist in checking itineraries and discarding TO instances with redundant information. When the destination node has been found, the working path will be initiated and the TO instance will travel back to the source node following the same stochastic navigation model than in the beginning; TO instances, at this time, know that the protection path has to be discovered. If such a TO instance should meet another TO instance focused on a working path (issued from the initial colony of TO instances), the latter will immediately be forced to re-initialize its itinerary and to perform the same job as the former. When the source node has been found again, TO instances looking for the protection path also prepare intermediate values in order to compute the QoP. When they reach the destination node, the QoP is computed with the discovered path and compared with the current QoP, if any. Then, the TO instances restart their job in order to find new protection paths. This process is endless; the colony of TO instances is evolving over time.

Only relevant code is presented; the `clone()` method and the constructor are not revealed, and parts of the code which are not relevant to the understanding of the TO have been omitted in order to improve code readability. Finally, the class prefix for the two constants `WORKING` and `PROTECTION` is not mentioned: they belong to the class `OptChannel`.

```
public class TO_QoP extends TOJava{

  // Frontal objects
  Vector itinerary, protStrategy, workingVPN; // Itinerary & customer requirements
  int sn, sl;           // # shared nodes/links between working & protection path
  int mode;             // Indicates if the TO looks for working or protection path
  int cycle = 0;        // Check for cycle
  boolean track;        // Path tracking indicator (source/destination target)

  public boolean activate(TOWrapperInterface wrapper) {
    String addr = (OTNNode) wrapper.getAgency().getLocalNode().getAddr();

    // _cd represents the connection descriptor for this connection (retrieved from
    // the local node in the simulation). We do not show details about that.

    // Check if this node belongs to the working path
    if (addr.equals(_cd.getSourceNode())) {
      initStrategy();      // We are at the beginning of the path
      track = true;        // So, we are ready to discover the path
    }

    if (!track) return true;            // If we are not tracking a path, then continue
                                        // to explore the network

    if (_cd.isNodeIn(addr, WORKING)) {  // Check if the node already belongs to WP
      sn++;                             // Will be used in the QoP calculation
      if (mode == WORKING) {            // It is time to change the mode, no need
        mode = PROTECTION;              // to find the WP, since already discovered
        if (!addr.equals(_cd.getSourceNode())) {
          initStrategy();               // If it is not the source node, we have to
          track = false;                // find it in order to discover a PP.
          return true;
        }
      }
    }

    // Check if the link is already used by the working path
    if ((mode == PROTECTION) && !addr.equals(_cd.getSourceNode()))

      // Do not forget to check for both directions!
      if ((_cd.getLink((String) itinerary.lastElement(), addr, WORKING) != null) ||
          (_cd.getLink(addr, (String) itinerary.lastElement(), WORKING) != null))

        sl++;                           // Will be used in the QoP calculation

    itinerary.add(addr);                // Store the current location

    if ((mode == PROTECTION) && protStrategy.contains(addr))   // Check with requirements
      protStrategy.remove(addr);
    else if ((mode == WORKING) && workingVPN.contains(addr))
      workingVPN.remove(addr);

    if (addr.equals(_cd.getDestNode()) &&      // Destination reached? Strategy ok?
        (((mode == WORKING)    && workingVPN.isEmpty()) ||
         ((mode == PROTECTION) && protStrategy.isEmpty()))) {

      double QoP = _cd.computeQoP(itinerary.size()-1+_cd.quantLinks(WORKING), sn-2, sl);
      if ((mode == WORKING) || (QoP > _cd.getQoP())) {      // Better QoP

       // Close the current open connection and open the new one
       // with the nodes stored in itinerary.
       // …

        track = false;      // Reset everything in order to discover other paths
        initStrategy();
        mode = PROTECTION;
      }
    }
    return true;
  }

  public boolean beforeMigration(TOWrapperInterface wrapper) {
    if (itinerary.contains(wrapper.destination())) {     // Check for a cycle
      cycle++;             // Cycle counter to check if it is a "dead end"
      if (cycle <= 5) {
        offload(wrapper);  // Offloaded in the blackboard for subsequent activations
        return false;      // Bye bye M-agent !
      } else {
        track = false;     // No possibility to avoid a cycle, so we reset everything
        initStrategy();
      }
    }
    cycle = 0;         // The TO can continue, so we reset the cycle counter
    return true;       // Keep alive
  }

  public boolean cooperate(TOWrapperInterface wrapper, TOJava otherTO) {
    TO_QoP _other = (TO_QoP) otherTO;       // Reference to the slave TO

    if (!track && !_other.track) return false;   // Only one instance is sufficient
    if (!(track && _other.track)) return true;   // In this case, no cooperation

```

```
 93     if ((mode == PROTECTION) && (_other.mode == WORKING)) {     // TO adaptation
 94       _other.mode = PROTECTION; _other.initStrategy(); _other.track = false;
 95 
 96     } else if ((mode == WORKING) && (_other.mode == PROTECTION)) {
 97       mode = PROTECTION; initStrategy(); track = false;
 98 
 99     } else if (itinerary.containsAll(_other.itinerary))  // Same itinerary?
100       _other.discard();
101     else if (_other.itinerary.containsAll(itinerary))
102       return false;
103 
104     return true;
105   }
106 
107   // User method to initialize the customer requirements regarding strategy
108   public void initStrategy() {
109     workingVPN.clear(); protStrategy.clear();         // Clear the requirements
110 
111     // Describe the working path of the client VPN and the protection strategy
112     workingVPN.add("Bern"); workingVPN.add("Zurich");
113     protStrategy.add("Martigny"); protStrategy.add("Andermatt");
114 
115     sn = 0; sl = 0; itinerary.clear();                // Reset
116   }
117 
118   public void init(TOWrapperInterface wrapper) {
119     setPriority(10); cleanEnv(wrapper); setPersistent(true);   // As usual…
120     // Other initialisation statements …
121 
122     initStrategy();   // Prepare the client requirements
123 
124     mode = WORKING;   // First, we want to discover a working path
125     track = false;    // Not sure to be in the source node; TO may be launched everywhere!
126   }
127 }
```

Table 7-1. The *TO_QoP* task objective

It has to be noted that the VPN connection is described both by a source node and a destination node. The proposed *TO_QoP* task objective performs both working and protection path allocation based on the VPN connection requirements (112-113) given in the `init()` callback (118). This is the reason why the `mode` variable (6, 124) indicates the kind of path the TO instance is currently trying to discover, either `WORKING` or `PROTECTION`. In our simulation, we assume that the working path is allocated along wavelength 0 and the protection path on wavelength 1.

Since the TO is continuously evolving within the network and can be launched everywhere, it is necessary to differentiate, by means of the `track` indicator (8), the TO instances from their final target. A *source node target* (`track` is `false`) TO instance will therefore navigate until the source node has been discovered whereas, a *destination node target* (`track` is `true`) TO instance is searching for the destination.

In the `PROTECTION` mode, the `activate()` callback checks for shared nodes and links during its travel (25, 41); once the destination has been reached, the current QoP value is compared with the established protection path (57-58) and a re-configuration is performed in case it is necessary (60-62). The TO then continues its lifecycle by re-initializing its internal knowledge (64-66), navigating freely until the source node has been discovered for the beginning of a new exploration. Another strategy for going back to the source node may obviously consist in extending the task objective with the *TO_PathSelect* TO model (see Section 3.6.4) and initializing the reverse trajectory with the itinerary recorded in the TO.

The `beforeMigration()` method avoids cycles in the discovered path (73); if the TO instance finds itself in a dead end however, the callback will force the TO (79-80) to restart its exploration from the source node by means of an internal counter (`cycle`). The value of 5 has been chosen in accordance with the mean nodal degree of the *Swiss* network (see Section 5.3.1).

The inter-TO cooperation strategy deals with redundant TO instances; when two source node target TO instances are detected, only one survives. The meeting of two TO instances with different targets implies that the source node target TO, which is in the *`WORKING`* mode, is changed into the *`PROTECTION`* mode and its target becomes the destination node (93-97). Finally, TO instances with identical itineraries are also discarded (99-102).

Two different TOs may also be combined in order to implement the DPS task objective. While the first TO based on a *TO_Monitor* task objective model continuously monitors client connections' shared nodes and links and stores QoP related information locally, another task objective resorting to a non-deterministic pre-planned (working and protection strategy) navigation model (see Section 2.3.3) computes the protection path configuration by means of the current QoP values in each node. This approach is oriented towards emergent behaviour; it can be associated with the R-chemical and the R-agent of the *SynthECA* approach presented in Section 2.4.2.

7.4.4 Experiments and Results

In this section, we examine the response of the ecosystem to the insertion of the *TO_QoP* task objective described in the previous sections. We have opted for the *Swiss* network composed of one fibre per link and containing two wavelengths. According to an initial approach, wavelength 0 is dedicated to the working path and wavelength 1 to the protection path. No additional restriction concerning the optical nodes has been introduced.

The VPN connection is described by means of a node collection to be included for the working and protection paths which are embedded in the task objective as depicted in Figure 7-5.

In order to discover a working path, the task objective has to rely on the constraints related to the customer VPN, i.e. the set of nodes (*Bern, Zürich*) which must appear along the working path. The protection path (*Martigny, Andermatt*) follows the same mechanism. No shared node or link should appear at the client layer.

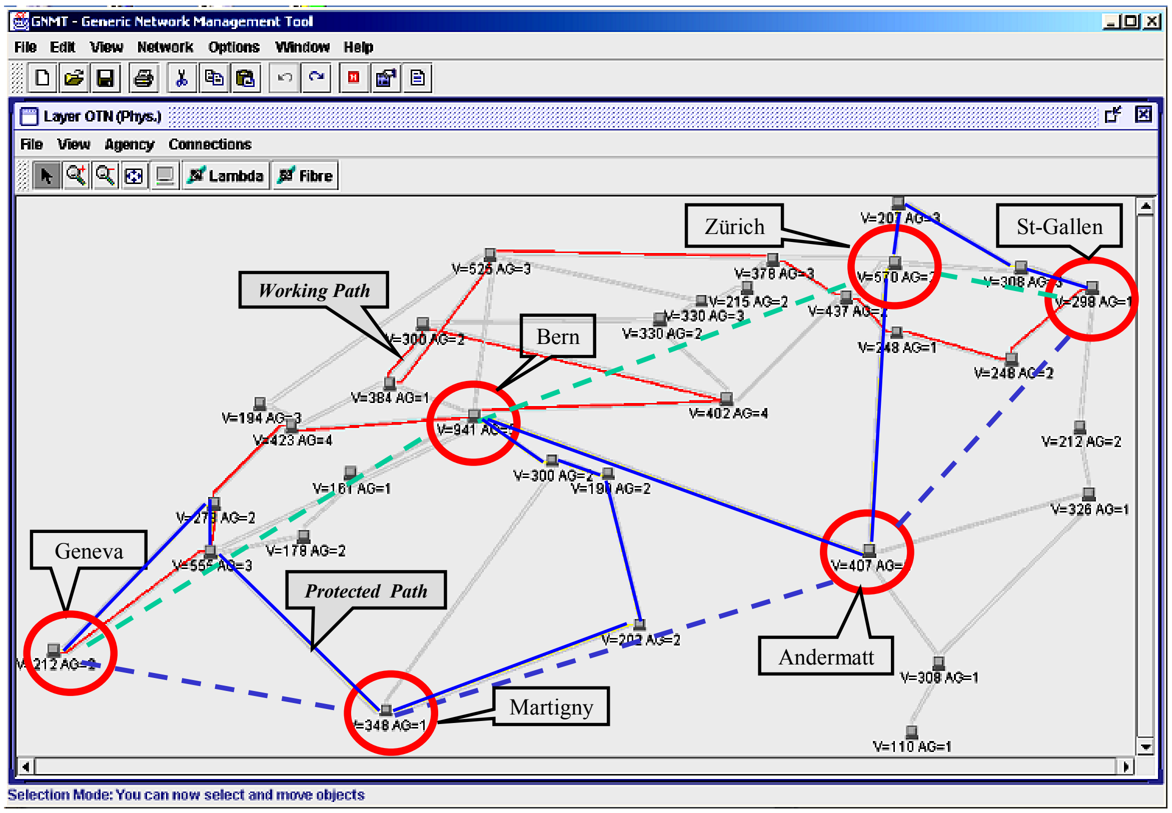


Figure 7-5. Working and protection paths in the optical VPN (dashed lines) and resulting allocation in the OTN

Since there are no additional constraints regarding path length, the first TO instance reaching the destination triggers the activation of the working path. The task objective then discovers a protection path in the OTN meeting the customer's requirements; a reconfiguration of the protection is performed each time a protection path leading to an improvement of the QoP is found. It has to be mentioned that the path length is not taken into account in the QoP either (see Section 7.4.2). As we can see on the figure, the mapping of virtual protection paths onto the physical topology unavoidably leads to a shared node and to a shared link. Figure 7-6 shows the values of the TO-related metrics with respect to this task objective.

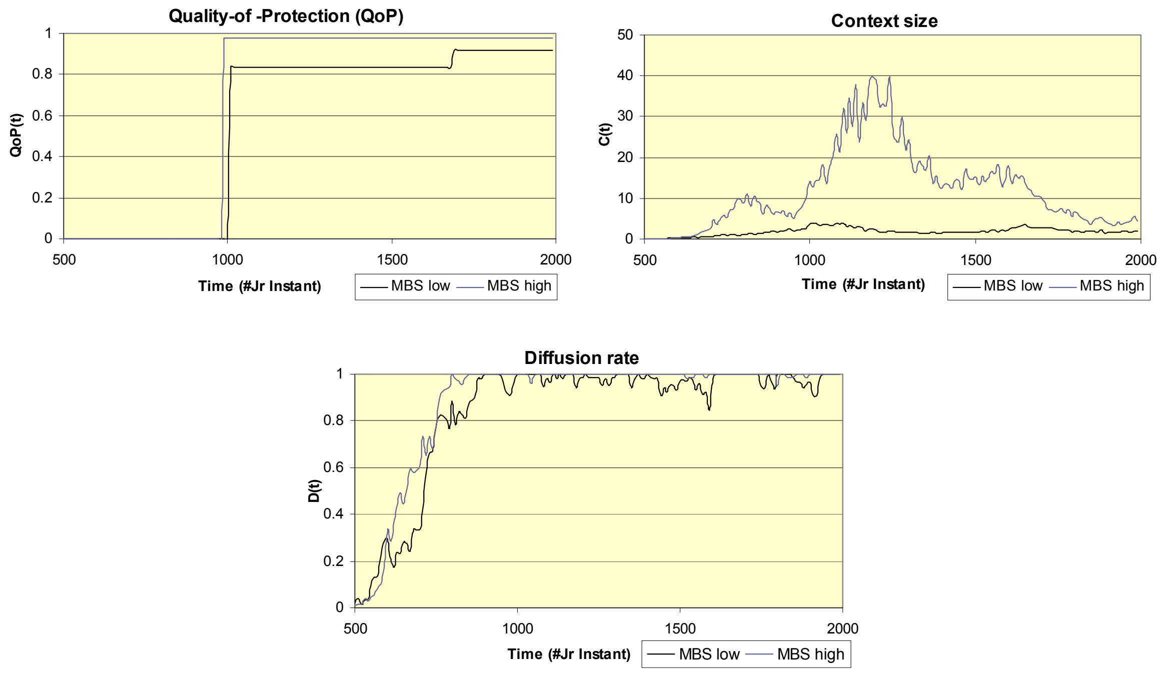


Figure 7-6. TO-related metrics for the DPS task objective

The task objective has been inserted after 500 instants, which corresponds to well-established stability of the ecosystem. As no constraint concerning the path length has been implemented, the discovery may lead to different paths, so that the QoP turns out to be higher for *MBS-high* than for *MBS-low* because of the slightly different working paths discovered in the two MBS.

The agent context size appears to be very small in *MBS-low*. On the contrary, *MBS-high* presents significant, but nevertheless reasonable, context size differences. *MBS-low* is therefore preferable when a large number of client connections have to be processed. As stated in Chapter 5, *MBS-low* is generally more appropriate for explorative tasks, such as the *TO_QoP* task objective. However, further experiments remain to be carried out in order to determine when the TOs exhibit better performance with *MBS-high* than with *MBS-low*, as far as QoP is concerned.

7.5 SUMMARY

In the present chapter, an approach to implementing active management based on *Ecomobile* within the future optical transport network has been proposed. We have introduced the OPTIMA project, which aims at the study and the implementation of intelligent and mobile agent based approaches towards the management of the future OTN. In this perspective, *Ecomobile* provides an adequate mobile agent infrastructure devoted to the transport network management, and can be considered as the basis for OPTIMA.

As we have seen, a combination of active network technology with enhanced optical nodes enables *optical* agents to be transported directly within the optical channel layer. The establishment of an ECC through a subset of the digital wrapper overhead and the recourse to an adequate MTP allows the agencies to exchange ACL-based messages between the optical nodes. Still, the deployment of *Ecomobile* into OTN requires an appropriate mapping of the place semantics. It has been proposed to associate intra-agency connectivity with the wavelength conversion function and inter-agency connectivity with wavelength switching in order to permit full investigation of the optical network by M-agents.

The introduction of networking functions into the optical domain and the capacity of OTN to support various client layers leads to the creation of new *optical* services, such as optical VPN, on-demand provisioning and protection services, associated to the arrival of new business requirements. In this context, SLAs will induce network operators to create differentiated services and to manage new business models involving service provider and customer.

The creation, deployment and management of value-added optical services can be achieved in the context of *Ecomobile* and of its particular computational model: the new value-added differentiated protection service based on the QoP metric presented in this thesis takes into account properties like the number of shared nodes and links between working and protection paths in order to evaluate the protection quality. This is the reason why we have developed the *TO_QoP* task objective, which deals with customer requirements regarding VPN connections, including the protection strategy. In the future, the vertical and horizontal interactions between network layers will also be investigated. The task objective performs allocation of working and protection path, and it appears that the QoP improves regularly. The behavioural analysis of this TO within a 2-wavelength *Swiss* network has shown that both MBS lead to a rapid convergence towards maximal QoP.

According to our experiments, the development of optical services with *Ecomobile* does not require any change in the control plane, so that it does not affect the signalling implemented into the OCC.

Conclusions

The recent progress in optical network technology, the explosion of Internet traffic, the creation of new multi-media services and the emergence of new business models involving network operators and service providers bring about profound changes in the conception of a transport network; as future multi-layer optical networks will be able to support several Tbit/s of client data, traditional platform-centred network management systems no longer fit the inherently distributed environment of future transport networks in terms of scalability, flexibility and time-to-market service deployment.

In this thesis, we have developed a self-adaptive mobile agent based infrastructure called *Ecomobile* in order to improve the management of future multi-layer WDM based optical transport networks. In the perspective of a decentralised management implemented into active components and of network resource control in a dynamic changing environment, we have considered a population of mobile agents in order to take advantage of powerful distributed and mobile processing techniques. *Ecomobile* allows us to deal with a number of issues related to the dissemination and activation of cooperative network management oriented tasks, which are not fully addressed by other mobile multi-agent systems. The network infrastructure is assumed to be composed of heterogeneous components and legacy systems.

We started Chapter 1 with the analysis of various mobile agent based network management approaches in the context of different network technologies and we tried to identify their main characteristics regarding agent behaviour. We placed particular emphasis on *Active Networks* because of their ability to transport code and to provide a rational execution environment for mobile agents. In the development of agent-oriented applications for large-scale networks, particular attention must be paid to standardization efforts. In this context, we examined agent standards such as FIPA and OMG MASIF. Although FIPA does not support agent mobility yet, the rich collection of agent-oriented specifications and the continuous efforts towards the improvement and extension of these specifications place FIPA as the favourite standards organization for agent-based telecommunication applications. While most mobile agent based applications are in fact based on a very limited use of mobility, i.e. a mobility mostly restricted to the download and remote execution of code, agent mobility can be considered as an *extension* to the intelligent agent paradigm, although mobile agents are predicated on reactive and small agents and stationary agents on deliberative "big" agents. In this perspective, mobile agents have the possibility to interact with stationary agents in order to take advantage of any agent-based service. We finally stressed our contribution towards the adoption of a unified view of the concepts of Active Networks, intelligent agents and mobile agents.

In Chapter 2, we identified three distinct abstraction models referring to the architecture of mobile agent systems: the computational, coordination and navigation models. The introduction of a loosely coupled task model allowed us to classify the dependencies involving the agent's operational behaviour together with the migration function and interaction schemes. We argued that the agent can implement loosely coupled task models when the information necessary for its migration is located within the environment, outside the agent itself. The development of a flexible and scalable mobile agent based network management system raises a number of issues addressed in Chapter 3. In most mobile agent systems, the number of agents is fixed and corresponds to an optimal size for the accomplishment of a specific task. According to our approach, the population of mobile agents and the network infrastructure are regarded as a whole, similar to an ecosystem composed of ecological individuals and their corresponding environment. We have therefore devised a novel agent architecture based on the three identified abstraction models and constituted of two distinct parts: the *Mobile Behaviour Scheme* (MBS) and the *Task Objective* (TO).

The mobile behaviour scheme was defined by means of reactive behaviours implementing the coordination and navigation models. Two mobile behaviour schemes were proposed in order to approximate the territoriality paradigm, an ecosystem principle referring to density-dependent intra-specific competition based on active interference between ecological individuals. Territorial behaviour plays a central role in the self-regulation of the agent population and allowed us to implement a "living" ecosystem-inspired mobile agent middleware fully aware of the network infrastructure.

The task objective, which is dynamically inserted into the mobile agent, represents the agent's operational behaviour; the TO has its own lifecycle and can be expressed in different programming languages, such as *Java* or rule-based languages like *JRules*, by means of specific wrappers. The particular computational model characterizing *Ecomobile* leads to a task design based on specific callbacks for the TO activation, migration and cooperation, the associated current location and the following destination. Still, we have proposed some generic TO models which correspond to basic functions of network transport management.

Ecomobile radically differs from other mobile agent systems because a self-regulated society of mobile agents navigate within the network before any specific task has been implemented. The ability to dynamically load and to manage different TOs, by implementing loosely coupled task models with different migration strategies, makes our middleware a flexible and universal network-aware mobile agent system particularly well suited to develop intelligent and adaptive management systems.

Our middleware relies on a threefold architecture involving the following active components: an agency, mobile agents called *M-agents* and implementing an MBS, and the task objectives, whose instances are stored in blackboards and in the M-agents. Similarities between *Ecomobile* and reactive systems have guided us to an implementation with a reactive programming formalism enabling discrete instant-based simulation; a synchronous cooperative approach significantly reduces the amount of non-deterministic effects appearing during the deployment of the mobile MAS. In Chapter 4, we presented the *Junior* micro-kernel, which provides an efficient *Java* API for reactive programming. The mapping of reactive behaviours onto reactive instructions led us to examine potential causality problems and to propose adequate solutions. The deployment of *Ecomobile* is achieved by means of the *Jade* FIPA-compliant agent platform which was originally intended for stationary FIPA agents; the migration of M-agents and class transfer are performed through ACL message exchanges between corresponding agencies. The *LEAP* agent platform derived from *Jade* also provides an interesting lightweight embedded agent environment designed for small devices and constitutes a promising technology for the deployment of agents within active components.

The deployment of mobile agents in accordance with this approach reveals several advantages: firstly, and thanks to the all-in-one agency model and its related synchronous execution environment, no additional resource-consuming components for the support of agent mobility, such as those present in other mobile agent platforms, are required. Stationary and mobile agents moreover share a common agent framework and can easily interact with each other; consequently, the local node resources will not be wasted because of the resort to several agent platforms. Finally, we promote the convergence towards a unique standard reference for intelligent and mobile agents in order to avoid undesirable misfits between different agent communities. In this context, FIPA turns out to be the most popular and promising agent organisation.

Chapter 5 introduces the *Generic Network Management Tool* (GNMT), which has been developed in the context of this thesis and is being transformed into an open source project. GNMT enables the functional simulation of a multi-layer optical network and enhanced nodes in which an *Ecomobile* agency has been implemented. A behavioural analysis of the M-agent population as well as the response of the ecosystem to the dynamic insertion of task objectives have been performed. The reaction instant has been considered as the time reference model and several MBS-related metrics such as population size, node and link visit frequency, and TO-related metrics like agent context size and diffusion ratio, have been defined. We discussed and compared the results issued from three regular and irregular network topologies with two MBS, *MBS-low* and *MBS-high*; while the population size reveals to be stable and, with adequate parameter values, to approximate the network size, the task objectives also exhibit good performance. We found that *MBS-low* leads to a better performance for task objectives requiring complex knowledge and extensive exploration of the network, whereas *MBS-high* exhibits a rapid diffusion of task objectives during the first instants and consequently induces the rapid deployment of monitoring tasks within the whole network, for example.

Since *Ecomobile* is characterized by self-adaptive and self-organising mechanisms evolving over time, our ecosystem is not really suited to the transfer of high-priority messages or tasks. In this case, the agency may be extended with *mediation* and *morphing* services in order to transfer messages or code directly via ACL messages. It also has to be noted that high priority messages in active networks can resort to special low-level active packets and thus avoid unwanted high-level processing.

An overview of optical network technology was presented in Chapter 6. The optical transport network constitutes our main application domain in this thesis. Optical nodes encompass basic networking functions operating fully in the optical domain without requiring any optoelectronic conversion, and which can provide new capabilities in terms of wavelength routing. Recent advances in the definition of a *digital wrapper* supporting client-transparent payloads within the optical channel structure encourage further investigations towards active optical nodes integrating an embedded communication channel in the wrapper overhead for the transport of so-called *optical* (mobile) agents.

The routing and wavelength assignment constitutes a major problem in optical networking; although the problem has been extensively discussed in its static form, the dynamic RWA including the development of distributed algorithms remains an important field of research. In Chapter 7, a possible implementation of RWA into optical networks with *Ecomobile* by means of specific task objectives was suggested. This *active management* approach allowed us to develop a new differentiated protection service and its associated *Quality-of-Protection* (QoP) metric, which takes into account horizontal and vertical dependencies in multi-layer optical networks, as a part of the customer SLA. We tried tow show that a simple task objective may lead to an efficient implementation of a new value-added service and that *Ecomobile* can therefore improve the management of optical networks and contribute to a future enhanced control plane.

Although the development of *Ecomobile* has been primarily devoted to transport network management, our middleware can be considered in a wide range of applications which give a central role to network intelligence. For instance, future content delivery networks intended for a huge amount of information related to interactive digital television will require self-organizing intelligent management solutions enabling providers to broadcast contents in a rational way. This kind of service also includes applications referring to service composition, in which logic is transferred from one location to another. In this context, *Ecomobile* provides an ideal infrastructure for investigating mobile information and application management.

Another range of applications is associated to the emerging *autonomic computing* paradigm devised by IBM[1] and mainly inspired from the autonomic function of the human central nervous system. A typical objective in this novel approach consists in developing distributed networks that are largely self-managing, self-diagnostic, and transparent to the user. The infrastructure and ecosystem-inspired concepts presented in this thesis may significantly contribute to this vision of future computing.

OPEN ISSUES

The current implementation of *Ecomobile* leaves a certain number of open issues. A statistical framework specific to population ecology should lead to a more fine-grained analysis of the ecosystem stability. Further investigations with different topologies and different network sizes should also be performed in order to confirm the density properties and the ecosystem stability revealed by our simulation results. At the current stage of our experiments, it is difficult to predict the evolution of certain TO-metrics in the case of large-scale networks, such as agent context size.

The current publicly available implementation of the *Junior* reactive machine should be improved in order to support a larger number of parallel instructions: the dynamic addition of reactive instructions relies on a recursive mechanism which may lead to a stack overflow problem when numerous M-agents are involved. An alternative way to overcome this limitation would consist in modifying the source code in order to implement tables or queues for handling reactive instructions. In this context, the authors of *Junior* are also working on a new version called *Storm*, which should provide significant improvements towards the processing of a huge number of events and instructions.

Finally, the transfer of code in *Java* requires the presence of classes in the remote agency. In particular, the TO classes must be transferred, so that TOs can be de-serialized by the M-agent context. Although classes can be transferred when it is necessary and although they can be loaded dynamically in the virtual machine, *Java* unfortunately does not allow a class to be unloaded. This problem is common to other mobile agent systems and remains to be solved in future versions of the JVM.

FUTURE WORK

The study of large-scale networks of 100 nodes or more, with various connectivity degrees, constitutes the next step of our investigations. In addition to the waiting time, we also intend to investigate dynamic adaptation under certain conditions according to other parameters in reactive behaviours, such as the number of clones. Improving the MBS itself by means of automatic switching between *MBS-low* and *MBS-high* in order to tune the ecosystem behaviour more finely and therefore to increase the TO performance, also constitutes one of our objectives. A statistical analysis of node visits, i.e. the frequency of TOs activation, would also provide helpful information for the measurement of TO performance and for its improvement. The development of a monitoring tool could finally provide a way to observe the ecosystem behaviour in real networks and to tune parameters in the environment dynamically in order to influence the agent population.

The validation of our system can be achieved by means of a simulated active node environment with the *LEAP* agent platform. A *Java* node operating system could be examined on top of the *NodeOS* for the construction of reliable active packets. Anchored in reactive systems, the formalization of *Ecomobile* based on SDL or Petri-nets and by means of partial ordering techniques would also clarify the influence of ecosystem parameters and allow us to define further MBS. The introduction of sexual reproduction as a

[1] http://www.research.ibm.com/autonomic

new reactive behaviour could reveal a new interesting way to explore the network and to disseminate the TOs efficiently.

The development of new task objectives for the management of optical networks constitutes one of the objectives of the OPTIMA project. Joint efforts of *Swisscom Innovations* and the *University of Fribourg* should lead to the elaboration of different state-of-the-art distributed RWA algorithms by means of task objectives. These algorithms could then be compared with centralised RWA algorithms currently supported by GNMT. The computational model of *Ecomobile* should allow us to combine several RWA algorithms with adequate parameterization. Specific rule-based task objectives will also be developed in the scope of an ongoing project dealing with the management of optical SLAs.

Last but not least, it could be interesting to invest some time and efforts in order to manage the complexity of a *terminode* network (see Section 1.4.3). In this final perspective, the combination of *Ecomobile* with a *MITAgent* approach (see Section 1.3.1) and the implementation of lightweight software components within the *terminodes* can be considered as one of the many possible future extensions of this thesis.

Appendix A The GNMT Simulation Framework

The present appendix aims at giving an overview of the simulation framework developed in the scope of this thesis. This chapter will serve as a basis for a future technical report [Ros02].

Generic Network Management Tool (GNMT) is a joint development effort of the *Telecom Group* (*Department of Informatics of the University of Fribourg*) and *Swisscom Innovations* (Bern), in Switzerland. Initially, the GNMT project was intended for the simulation of *Ecomobile* within an optical transport network infrastructure. In this appendix we will describe the GNMT architecture and each module in terms of *Java* classes and functionalities.

A.1 INTRODUCTION

GNMT is a 100% *Java*-based functional simulation tool dedicated to the study of new network-oriented technologies such as agent-based distributed network management or *Active Networks* applications. Since it is developed in *Java,* GNMT is not intended for real-time simulation. The GNMT framework provides a multi-layer network simulated environment in which the components of different layers can interact with each other and implement various capabilities and constraints. GNMT was originally designed to provide stationary and mobile multi-agent systems such as *Ecomobile* with an semi-emulated heterogeneous and dynamic network environment. Hybrid centralised and decentralised management approaches can be supported simultaneously.

In GNMT, the underlying transport network is assumed to be a WDM-based optical network on top of which different layers, such as IP, SDH or ATM, can be added. Each layer may have its own management philosophy and interactions between layers rely on a particular interaction model. All the elements will be detailed in this document.

The GNMT framework is divided into two component-oriented parts: the *kernel* and the *private extensions*. The *kernel* includes all the classes related to the core GNMT network model and basic user interfaces for simple simulations, while the *private extensions* refer to packages developed in the scope of students contributions at *Swisscom Innovations*.

GNMT is currently evolving to become an *open source* project[1]. The kind of public licence to be used is still under discussion and the official distribution including the kernel classes will be available as soon as these license-related issues have been clarified. Figure A-1 presents an overview of the existing packages.

The *Scalable TeRabit Optical NetworkinG* (STRONG) and the INNOVATE projects are two extensions to GNMT, which have been developed at *Swisscom Innovations*. They contain both *kernel* and *private extensions*; classes and functionalities belonging to the different parts however remain to be determined.

[1] http://gnmt.sourceforge.net

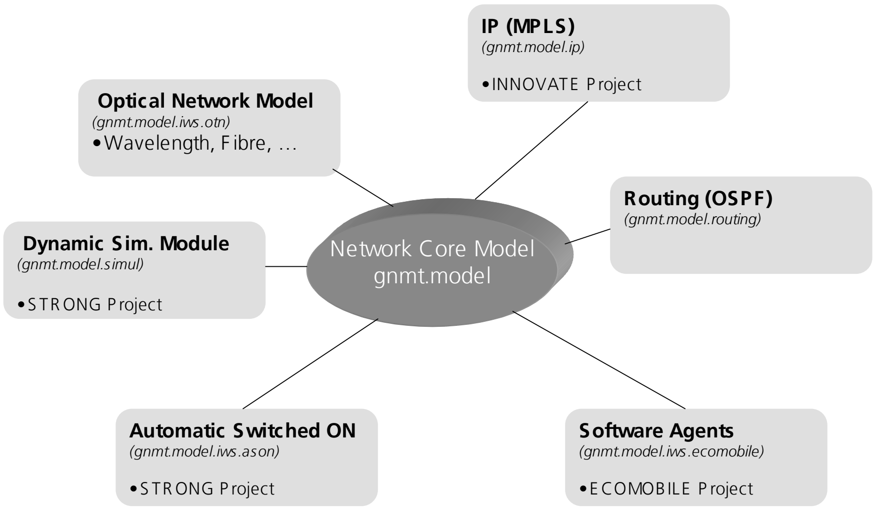
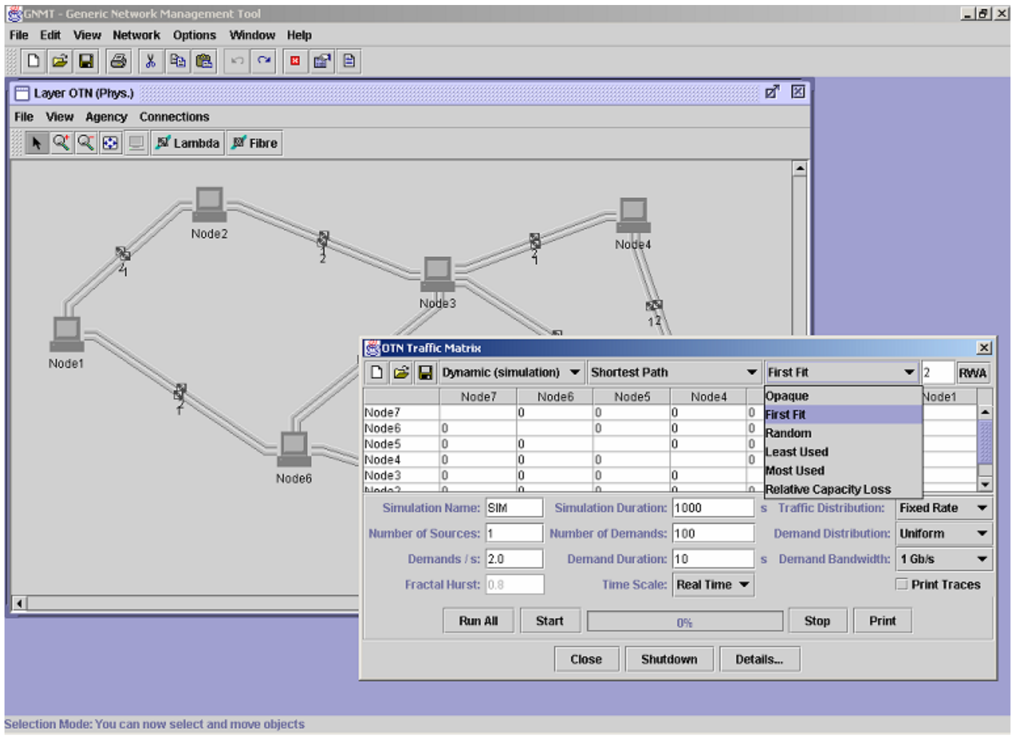


Figure A-1. Modules in GNMT

The core GNMT network model consists of abstract classes and basic functionalities extended and implemented by each layer. Basically, each network layer technology (OTN, ASON, IP/MPLS) corresponds to a particular package. Classes required for dynamic simulation, OSPF-like routing algorithms and the *Ecomobile* middleware constitute further distinct packages.

Figure A-2 gives a snapshot of the graphical interface of the GNMT tool including two windows: a physical optical network topology and the related traffic matrix.



Dynamic (simulation) | Shortest Path | First Fit | 2

	Node7	Node6	Node5	Node4	
Node7		0	0	0	0
Node6	0		0	0	0
Node5	0	0		0	0
Node4	0	0	0		0
Node3	0	0	0	0	

Simulation Name: SIM
Simulation Duration: 1000 s
Traffic Distribution: Fixed Rate
Number of Sources: 1
Number of Demands: 100
Demand Distribution: Uniform
Demands / s: 2.0
Demand Duration: 10 s
Demand Bandwidth: 1 Gb/s
Fractal Hurst: 0.8
Time Scale: Real Time
Print Traces

Figure A-2. A GUI snapshot of the GNMT

The GNMT root model is based on the *Model-View-Controller* (MVC) design pattern [GHJ+95] supported by the *Swing* architecture; Swing is a *Graphical User Interface* (GUI) component kit, part of the Java Foundation Classes (JFC) integrated into Java 2 platform. In our context, MVC allows for a separation between classes specific to the network and simulation model (Model), classes specific to the graphical components used by the GUI (View), and classes specific to user interactions involving mouse, keyboard and screen (Controller).

The current implementation of GNMT however does not rely on a fully MVC implementation yet [Zbi01]; the graphical components are handled by means of a commercial third-party library, called *Ilog JTGO*[1], which provides efficient network-oriented graphical components and facilities for the management of user interactions. The replacement of the *Ilog* library by an open graphical components library developed at the *University of Fribourg* is currently under investigation and will lead to a fully MVC implementation.

A.2 THE GNMT KERNEL

The minimal set of GNMT classes and functionalities necessary to perform a simple simulation belongs to the GNMT kernel which is freely available according to the *open source* philosophy. This section outlines the different packages forming the kernel.

The root tree structure is depicted on Figure A-3; the main package entry is called *iiuf.gnmt* ("iiuf" for "*Institute of Informatics of the University of Fribourg*"). In the near future, the kernel root package will be renamed to *edu.diuf.gnmt* in order to differentiate from the *private extensions* and to match with the new name of the department (*Department of Informatics*).

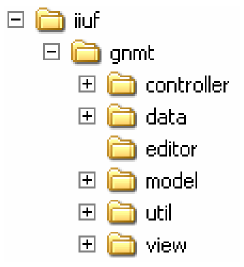


Figure A-3. GNMT kernel: classes tree with root packages

The `model`, `view` and `controller` packages implement the MVC design pattern. The `view` and `controller` packages contain the same package organization as the `model` package. In this section, we mainly focus on the `model` package, as the other packages are currently being re-designed. The *data* package contains sample network topologies described in XML. The *editor* package contains the main class (*Editor.java*) of the application whereas the `util` package contains diverse auxiliary routines and mathematical structures.

The tree structure of the `model` package is depicted on Figure A-4. The abstract classes of the core GNMT network model belong to the `model` package. The `routing` package contains generic routing algorithms whereas the `simul` package contains the necessary classes for dynamic simulation. This

[1] http://www.ilog.de/products/jtgo

package is subdivided into a publicly accessible *kernel* part (packages `iiuf.gnmt.X`), which is part of the GNMT kernel, whereas the *private extensions* part (packages `com.swisscom.gnmt.X`) consists in an extension specific to *Swisscom Innovations*.

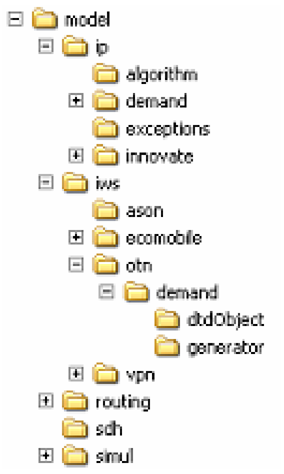


Figure A-4. General structure of packages in the `model` part

The abstract classes of the core GNMT network model belong to the *model* package. The `routing` package contains generic routing algorithms whereas the `simul` package contains the necessary classes for dynamic simulation. This package is subdivided into a publicly accessible *kernel* part (packages `iiuf.gnmt.X`), which is part of the GNMT kernel, whereas the *private extensions* part (packages `com.swisscom.gnmt.X`) consists in an extension specific to *Swisscom Innovations*.

A.2.1 The Core GNMT Network Model

This package describes the GNMT network model that is made of abstract classes which have to be extended and implemented into layer subclasses.

Our model relies on a combination of two different approaches that are characteristic for modelling layered networks; first of all, the generic functional architecture of ITU-T transport networks [G805_95] proposes a functional decomposition of the transport network in terms of layering and partitioning; while the layering deals with the separation between distinct transport technologies, the partitioning consists in subdividing the functional components within a single layer. This model supports multiple layers which can be interconnected according to a client-server relationship; in a layer network, the link connections formally provide connectivity between topologically adjacent sub-networks; they are provisioned by the services of a trail[1] at another layer. This layer is known as the server layer, while the layer in which link connection requests are issued is called the client layer.

[1] A *trail* is defined as the combination of the connection information augmented with additional overhead information used to achieve the *Operations, Administration and Maintenance* (OAM) objectives.

The second approach is based on the ISO/OSI layered protocol model [T96]; it consists in a functional decomposition based on a seven-layer protocol model. Since the ISO model was originally conceived within or around a single transport layer without any consideration for partitioning, it offers a perspective which is different from the approach proposed in the ITU-T model. The potential compatibility between the two models is still under discussion at the ITU-T. The overall core GNMT network model is depicted in Figure A-5.

The abstract core GNMT network model is a generic representation of the main network components constituting a multi-layer transport network. Each component must therefore be specialized according to the network technology which is being considered in the simulation.

In GNMT, a client demand is synonymous with client connection, or simply connection, and is defined as a general service associated to each layer.

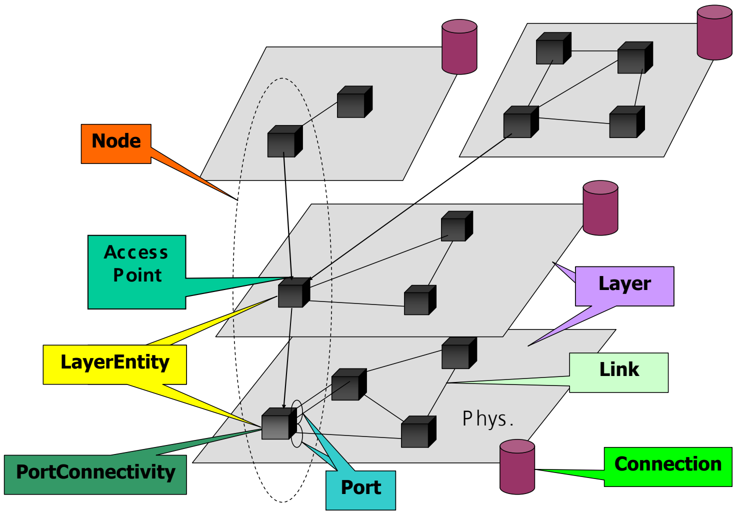


Figure A-5. Core GNMT network model inspired from the ITU-T and ISO models.

The abstract core GNMT network model is a generic representation of the main network components taking place in a multi-layer transport network. Each component must therefore be specialized according to the network technology which is being considered in the simulation.

In GNMT, a client demand is synonymous with client connection, or simply connection, and is defined as a general service associated to each layer.

IIUF.GNMT.MODEL

In the beginning, the network may be composed of multiple layers. A *layer* is defined as a collection of nodes associated with the same transport technology (IP, ATM, SDH, etc.) and therefore refers to a specific protocol. The layered model of GNMT is composed of the lowest layer called *physical layer* on top of which logical or virtual layers are interconnected. Each layer can implement a different management approach based on centralised management, i.e. the management logic is placed at the layer level, or decentralised management in which case the management logic is placed in each layer node and relies on a signalling protocol or any kind of information exchange mechanisms. Figure A-6 presents an overview of the classes belonging to the `model` package; the UML diagram only contains classes and relationships, as well as basic attributes and operations.

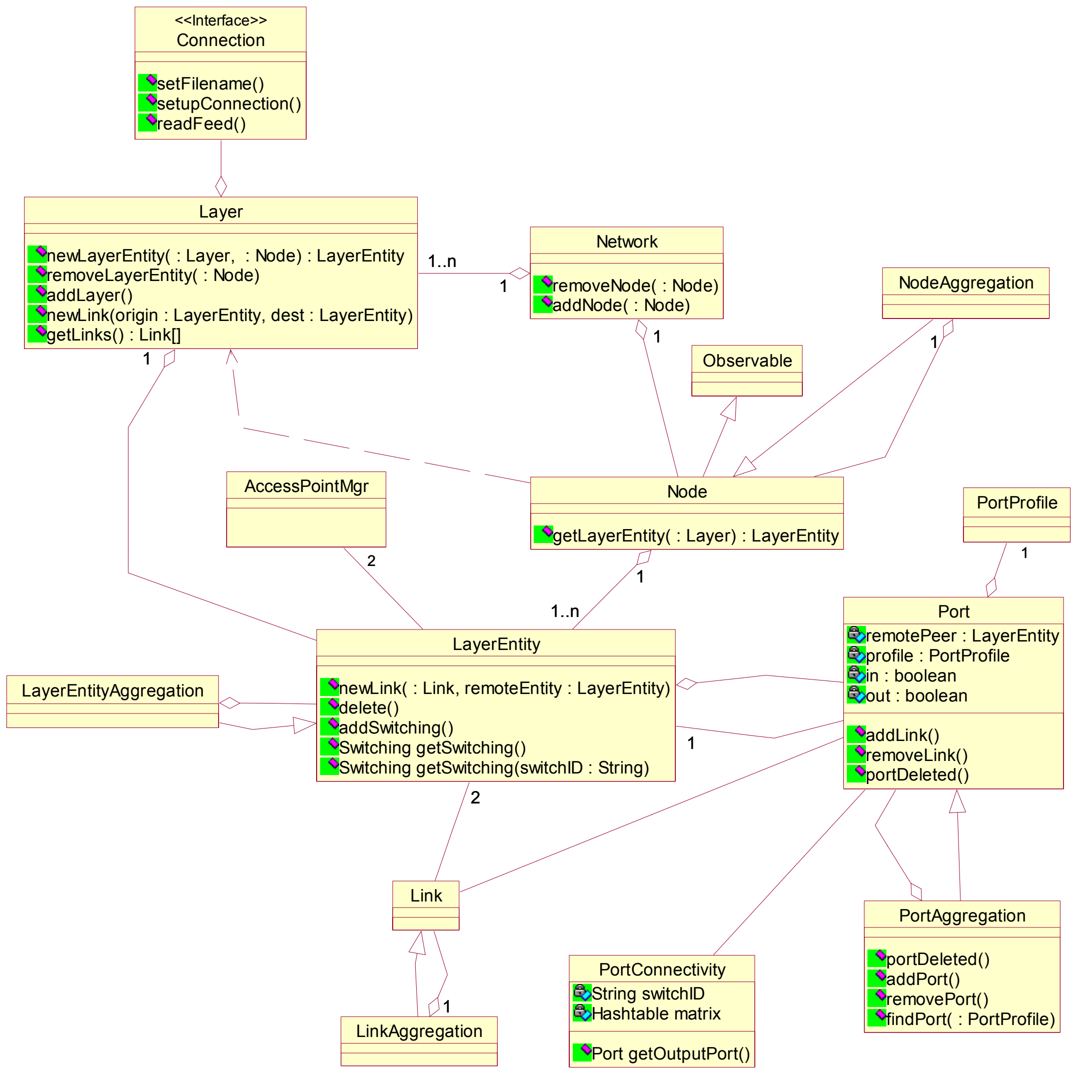


Figure A-6. UML Diagram of the core GNMT network model (package iiuf.gnmt.model)

The `Network` class contains all the network layers likely to be used in GNMT. A layer (class `Layer`) is composed of `LayerEntity` objects characterizing network nodes within the layer. Node functionalities and related management logic must be implemented into layer entities. The `Node` object represents an OSI node, i.e. a node which encompasses several layer entities belonging to different layers. The particularity of these layer entities is that they are placed at the same location, i.e. the positioning of the layer entities of a specific node is determined by the position of the layer entity in the physical layer. Such an approach allows a node to be regarded as an OSI node in which several protocol layers are implemented. According to this definition, a layer entity is the intersection of an OSI node with its layer. In order to avoid confusion, we plan to rename the `Node` class as `OSINode` and the `LayerEntity` class as `Node`.

In the current version of the GNMT, there is only one kind of network, called `OTNNetwork` (see Section A.2.3), in which the optical transport network constitutes the physical layer; the client layers can be either IP/MPLS, ASON, or a generic VPN. The physical layer remains the unique layer in which layer entities can be created; these nodes can then be reported to an upper layer by simply “attaching” nodes from the physical layer (optical layer) to the client layer, so that the corresponding nodes in the client layer include layer-specific functionalities. As a consequence, a client layer can only have a subset of network nodes of lower layers. The network topology, i.e. links between layer entities, can then be freely defined in each layer.

Layer entities (nodes at layer level) within the same layer are linked by means of `Link` objects. Links can be aggregated into a `LinkAggregation` object so that multiple links or channels can be defined between two layer entities. Interactions between layers are addressed using access points that will be explained later on.

The `Port` class defines a general abstract port which groups several links (including link aggregations). The port itself can be hierarchically organised. At the time of a link creation, the link is assigned to a "top-level" port which is automatically created. Then, the designer can associate the link to different ports. For example, an optical fibre can be associated to a port aggregation, which in turns contains several optical channel associated ports. Port attributes (`in, out`) indicate the link direction and thus tells whether the link is uni/bi-directional. In addition, a `PortProfile` object is associated to each port in order to hold specific attributes depending on the port type, such as wavelength frequency, bandwidth, and so on.

Routing can be achieved by means of the `PortConnectivity` object, which indicates which input ports can be routed to output ports. Switching information can be introduced statically before the simulation and/or dynamically during the simulation. The flexible notion of port permits a group of links to be routed at a time: a port can aggregate several links or even several ports.

In GNMT, the edition of nodes and links within a specific layer defines the *network topology*, which corresponds to network components and their physical or logical connectivity. Nothing about client connections is specified in a network topology. Connections, i.e. client connection/demand requests, are introduced by means of a `Connection` object when a topology has been designed; they can be either configured manually, directly by editing each connection link, or by means of a XML file or of a traffic matrix; connections can also be automatically generated thanks to a *demand generator*. Since the connection semantics and related algorithms strongly depend on the layer, the connection algorithms are not defined in the core model, but stored in a package called *demand* and located in each layer.

Finally, the *Access Point Manager* (APM) (class `AccessPointMgr`) enables interactions between multiple layers (vertical interactions). The concept relies on the access and connection points as defined in the ITU-T model according to a client-server paradigm. In our model, the APM manages a collection of layer entities belonging to the upper layer, those entities acting as connection points. Each layer entity therefore includes an APM.

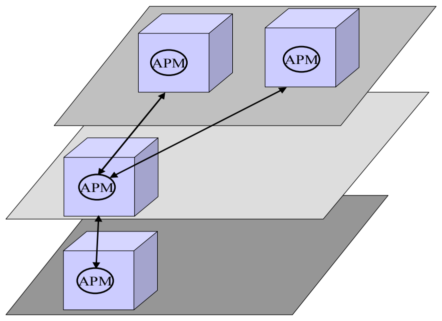


Figure A-7. Multi-layer management with Access Point Manager in GNMT

Only one APM has been implemented so far; however, future releases of GNMT will allow a layer entity to manage several APMs so that different node-related services can be distinctly provided to client layers.

A.2.2 Routing Algorithms

IIUF.GNMT.MODEL.ROUTING

This package contains conventional routing algorithms. Currently, only the *Open Shortest Path First* (OSPF) algorithm has been implemented and is used by the ASON package.

A.2.3 Intelligent Wavelength Services

IIUF.GNMT.MODEL.IWS

This package contains all the components related to the optical transport network, including intelligent services such as ASON, on the one hand, and *Ecomobile*, on the other hand.

IIUF.GNMT.MODEL.IWS.OTN

The *Optical Transport Network* (OTN) layer defines the optical components of the physical layer, as depicted on Figure A-8.

Besides the OTN layer, the *`OTNNetwork`* class is used by GNMT to create the multi-layer network environment at the starting point; an *`OTNNetwork`* contains an instance of each layer liable to be used in the network configuration. Consequently, it is not possible to have more than one instance of each layer.

The *`OTNLayer`* is composed of optical nodes, fibres and optical channels. Optical nodes refer to a general optical component implementing optical functions.

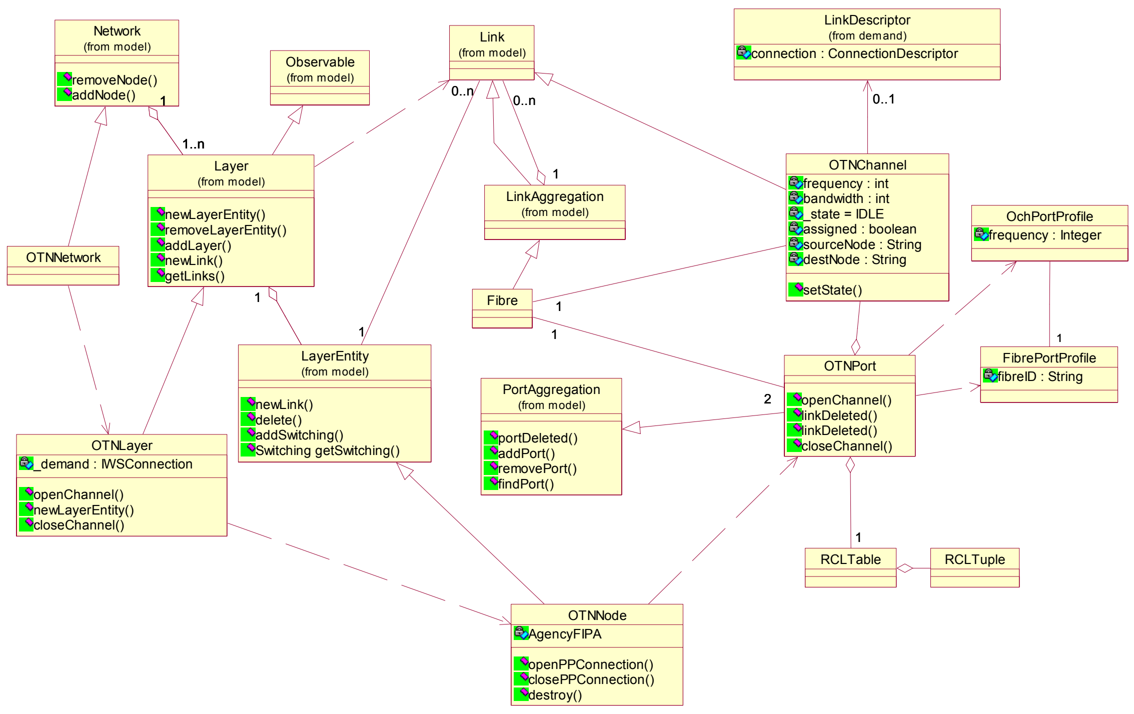


Figure A-8. UML Diagram of the OTN layer

The links are described by optical channels (class `OptChannel`) and fibres (class `Fibre`). A fibre is considered as a `LinkAggregation` which can contain several `OptChannel`. Like links, optical ports are organized hierarchically. For commodity reasons, an implementation choice has led to the definition of the *top-level* port (class `OTNPort`) as a container, in which two `PortAggregation` instances are stored, one gathering optical fibres, another gathering optical channels. Each fibre contains references to embedded optical channels.

Two corresponding profiles have been defined, `FibrePortProfile` and `OptChannelPortProfile` respectively. These two profiles allows a link to be uniquely identified. A fibre or an optical channel can be retrieved according to a particular profile; when the profile is equal to `null`, whichever link, actually the first element of the container, is returned.

IIUF.GNMT.MODEL.IWS.OTN.DEMAND

The connections that take place in OTN are defined in this package. A connection is defined by a connection descriptor (class `IWSConnection`) which defines a client connection entirely by means of link descriptors (class `LinkDescriptor`). A connection descriptor may contain both *working* and *protection* paths. The source and destination nodes are given in the connection descriptor and the links can be entered manually or be generated automatically by the corresponding routing algorithms.

Connections can be configured by means of a traffic matrix. In the `otn.demand` package, for example, the simulation is profiled and started from the traffic matrix. Each layer can define its own traffic matrix.

When interactions between several layers take place, the upper layers request the lower layers to which they are interconnected for client connections. This model basically corresponds to the overlay model considered in GMPLS.

IIUF.GNMT.MODEL.IWS.ASON

The *Automatic Switched Optical Network* (ASON) package contains an implementation of different centralised algorithms (layer level) related to the *Routing and Wavelength Assignment* (RWA) problem [Mah01][Stu02] for both transparent and opaque optical nodes.

Routing algorithms are: Fixed Routing, Fixed-alternate routing, and Adaptive Routing (*Least-congested-path* (LCP) and *Least-Loaded* (LL)). Wavelength assignment algorithms are: Random, First Fit, Least-used, Most-used, Opaque. Further information concerning these algorithms is out of the scope of this document.

IIUF.GNMT.MODEL.IWS.ECOMOBILE

This package contains the implementation of *Ecomobile*.

A.2.4 Dynamic Simulation

IIUF.GNMT.MODEL.SIMUL

With GNMT, it is possible to perform a dynamic simulation by means of a XML script file containing specific commands [Roc01] in order to create nodes, remove nodes, create links, remove links, simulate a fibre cut, etc. This mechanism is also used to store and restore network topologies within GNMT.

At the beginning of the simulation the XML script is entirely read and all commands are stored in a priority queue according to their execution order. The commands are performed according to an event-based scheduling. The available XML commands are described in Table A-1.

Command	*Description*	*Status*
ADD_NODE	Add a node in the topology	Implemented
REMOVE_NODE	Remove a node	Not implemented
ADD_LINK	Add a link in the topology (wavelength + fibre)	Implemented
REMOVE_LINK	Remove a link	Not implemented
ADD_DEMAND	Open a connection between two nodes	Implemented
GENERATE_DEMANDS	Turns on the demand generator according to specific distribution	Implemented
FAILURE	Simulate a failure at a node or at a link	Not implemented

Table A-1. List of simulation commands

When a network topology has to be saved, a script file is automatically generated. Each node and link is described by the *`ADD_NODE`* and *`ADD_LINK`* commands respectively, with attributes indicating their graphical position, type, etc. The topology can then be restored by *playing* the script file.

The XML data structure is defined by means of a DTD file which describes the syntax and grammar of each command and attribute. The DTD file is then processed by a third-party framework called *Zeus*[1], and a *Java* class is produced for each XML *tag*. Resulting classes are put in a sub-package called `dtdObjects` placed in the `demand` package associated to each layer. The overall process generating *Java* classes and using XML file is depicted on Figure A-9.

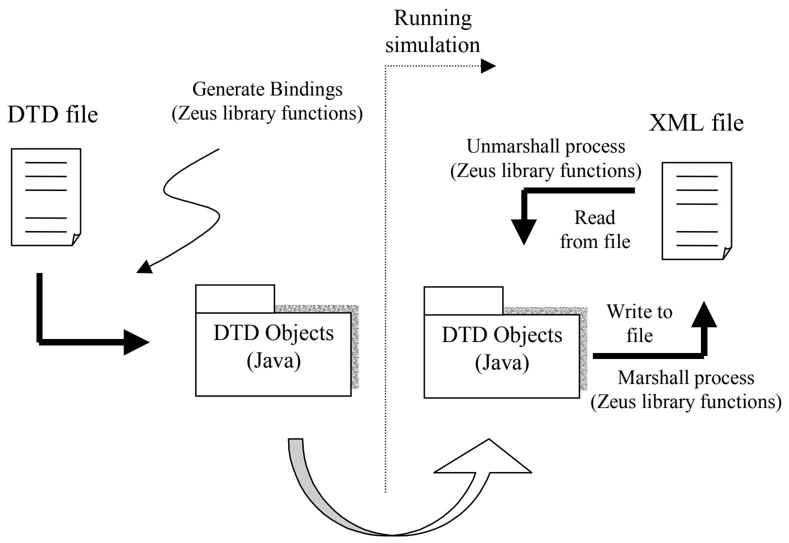

Running simulation
Generate Bindings (Zeus library functions)
DTD file
XML file
Unmarshall process (Zeus library functions)
Read from file
DTD Objects (Java)
DTD Objects (Java)
Write to file
Marshall process (Zeus library functions)

Figure A-9. XML and binding with Java objects

Table A-2 shows an example of an XML script file containing the creation of one node and one fibre with two wavelengths. The attribute `exec_order` indicates the place within the priority queue in which the command takes place.

```
<?xml version="1.0" encoding="UTF-8" ?>

<XMLFILE>

<ADD_NODE name="Node8" coordy="30.0" coordx="259.0" exec_order="1" />

<ADD_LINK type="FIBRE" exec_order="9" node_dest="Node1" num_of_lambdas="2" node_orig="Node3">
<LAMBDA status="IDLE" wavelength_ID="0" />
<LAMBDA status="IDLE" wavelength_ID="1" />
</ADD_LINK>

<DELAY time="20" exec_order="33" />

</XMLFILE>
```

Table A-2. Example of a XML script

The `simul` package is composed of a kernel part and a private extension part. The event-based scheduler, the mechanisms used for processing XML files and the demand generation based upon *Poisson* distribution are considered as the kernel part, whereas the other distributions, such as On-Off process and Fractal Renewal processes, are private extensions.

[1] *Zeus* is freely downloadable at http://zeus.enhydra.org

A.3 Private Extensions

A.3.1 The IP Layer

IIUF.GNMT.MODEL.IP

This package integrates previous work achieved in the *Innovate* project [Jia01] at *Swisscom Innovations*. This project aimed at simulating MPLS *Traffic Engineering* (TE) techniques with the *Constraint Based Routing* (CBR) algorithm. In this context, the simulation of a IP over WDM network, implementing both peer-to-peer and overlay models, has been realized.

This package illustrates a kind of *private extensions* that can be developed in the GNMT framework.

A.4 Conclusions

In this appendix, we have presented an overview of the GNMT functional simulation framework and its main components, namely the core GNMT network model, packages related to the *kernel* and *private extensions* part. GNMT currently becomes an *open source* project so that the *kernel* part will be publicly accessible. However, the type of licence to be used is still under consideration.

GNMT has been originally developed in order to implement *Ecomobile* and to perform simulations within an optical network environment. In addition, several private extensions have been developed in order to evaluate routing and wavelength assignment algorithms, on the one hand, and IP/MPLS related algorithms, on the other hand. These extensions have been developed during student work at *Swisscom Innovations*.

The GNMT design relies on a *Model-View-Controller* design pattern. The graphical objects resort to a commercial third-party library called *Ilog JTGO* which provides nice features for drawing nodes and links. In particular, the framework provides facilities to handle link aggregation. The graphical part of GNMT is currently being re-designed by means of a non-commercial framework in the perspective of the *open source* release.

The core GNMT network model stems from two main network models: the ITU-T transport network model and the OSI model. The combination of these two models makes the simulation of multi-layer network possible. Currently, GNMT implements the optical transport network layer as the physical layer on top of which different layers, such as IP/MPLS, or generic VPN can be simulated. The demand generator enables dynamic traffic according to specific call distribution.

Possible improvement of GNMT would consist in using a basic desktop-oriented framework such as *Netbeans*[1] or *IBM Eclipse*[2] in order to benefit from standard functionalities for data manipulation, window management, file handling, etc.

[1] http://www.netbeans.org
[2] http://www.eclipse.org

Bibliography

WORKSHOP/CONFERENCE

Daniel Rossier, Rudolf Scheurer, **"Ecosystem-inspired Mobile Agent Middleware for Active Network Management"**, in Proceedings of MATA'02, Fourth International Workshop on Mobile Agents for Telecommunication Applications, October 2002, Barcelona, Spain

Daniel Rossier-Ramuz, Rudolf Scheurer, **"ECOMOBILE: A Mobile Agent Ecosystem for Distributed Network Management"**, in Proceedings of ECUNM'02, 2nd European Conference on Universal Multiservice Networks, April 8-10, 2002, Colmar, France

Daniel Rossier-Ramuz, Daniel Rodellar, Rudolf Scheurer, **"Dynamic Protection Set-up in Optical VPN using Mobile Agent Ecosystem"**, in Proceedings of DRCN'01, Third International Workshop on Design of Reliable Communication Networks, Budapest, October 2001

Daniel Rossier-Ramuz, Daniel Rodellar, Rudolf Scheurer, **"An Intelligent and Mobile Agent-based Approach for Dynamic Protection Set-up in Future Optical Networks**", in Proceedings of ONDM'01, Fifth Working Conference on Optical Network Design and Modelling, Vienna, February 2001

Jingming Li Salina, Daniel Rossier, Manuel Dinis, Laurie Cuthbert, Laurissa Tokarchuk & John Bigham, "**Agent-based resource management for 3G networks**", Proceedings of Mobile Communications Summit, Galway, Ireland, 1-4 October 2000

Daniel Rossier-Ramuz, Rudolf Scheurer, "**An Introduction to Optical Agents: Intelligent and Mobile Agents for WDM Optical Network Management**", Proceedings of IMPACT'99, *Impact of Agent Technology on Telecommunications*, Seattle, USA, 2-3 December 1999, pp. 131-139

BOOK (CHAPTER)

Daniel Rossier-Ramuz et al., **"An Intelligent and Mobile Agent-based Approach for Dynamic Protection Set-up in Future Optical Networks"**, in '*Towards an Optical Internet*', edited by Admela Jukan, in book series "International Federation for Information Processing", Volume 204 (Kluwer Academic, 2001)

Daniel Rossier-Ramuz, Rudolf Scheurer, **"Implementation of Mobile Agents for WDM Network Management**", Hayzelden (ed.), *'Agent Technology for the Communications Infrastructure'* (Wiley & Sons Ltd, 2000)

TECHNICAL REPORT

Daniel Rossier, **"A Description of the Generic Network Management Tool"**, Technical Report, Department of Informatics, University of Fribourg, Switzerland, August 2002

Daniel Rossier-Ramuz, Rudolf Scheurer, Beat Hirsbrunner, **"A Mobile Agent Ecosystem for Rule Based Network Oriented Applications"**, Technical Report 01-22, Department of Informatics, University of Fribourg, Switzerland, July 2001

JOURNAL

Daniel Rossier-Ramuz, "**Ecomobile: a Mobile Agent Ecosystem for Active Network Management**", submitted to *Special Volume on Distributed and Mobile Software Engineering* of the "Annals of Software Engineering", Vol.17, Kluwer Academic Publishers, 2002

Daniel Rossier, "**Software Intelligent Agents: The Next IT Revolution**", Swisscom Comtec, January 2001

Curriculum Vitae

Daniel Rossier-Ramuz
Les Vuarines 12
1782 Belfaux

Born on May 26th, 1968

Married, two daughters: Aline (1997) and Emilie (2000)

Education

1983 - 1986	College St-Michel, Fribourg
1991 - 1992	CMS (Cours de Mathématiques Spéciales)
1992 - March 1996	Computer Science Engineer EPFL (Ecole Polytechnique Fédérale de Lausanne)
April 1997	Postgraduate Certificate EPFL (signal and pictures digital processing)
1999 – 2002	PhD at the University of Fribourg performed in an industrial context at Swisscom Innovations Ltd, Bern

Professional Experience

Dec. 1986 – Nov. 1989	Software Developer at ILFORD AG, Marly (FR)
Dec. 1989 – Sept. 1990	Software Developer at SIBRA S.A., Fribourg (FR)
July – Oct. 1994	Practical course at Telecom, Bern (BE)
April 1996 – Sept. 1998	Software Engineer at Kudelski S.A., Cheseaux-sur-Lausanne (VD)
Oct. 1998 -	Research Engineer & Project Leader at Swisscom AG, Bern (BE)

Languages

French (mother tongue), English, German

Hobbies

Piano, reading, HAM radio (HB9 HFP), squash

References

[BAG01] Martin Bolduc, Evangelos Armiros, Yves Gagnon, Chris Hamilton, "Developing a Service Level Agreement Framework in an All-optical Network Environment", in Proc. of NFOEC (National Fiber Optic Engineers Conference), Baltimore, USA, July 2001

[Bau99] Joachim Baumann, "Control Algorithms for Mobile Agents", PhD thesis, Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR) der Universität Stuttgart, 1999

[BBB$^+$99] Andrzej Bieszczad, Pratik K. Biswas, Walter Buga, Manu Malek, and Hai Tan, "Management of Heterogeneous Networks with Intelligent Agents", Bell Labs Technical Journal, October-December 1999

[BBC$^+$01] Ljubica Blazevic, Levente Buttyan, Srdan Capkun, Silvia Giordano, Jean-Pierre Hubaux and Jean-Yves Le Boudec, "Self-Organization in Mobile Ad-Hoc Networks: the Approach of Terminodes", available at http://www.terminodes.org, 2001

[BBC$^+$98] M. Breugst, I. Busse, S. Covaci, T. Magedanz, "Grasshopper – A Mobile Agent Platform for IN Based Service Environments", IEEE IN Workshop, Bordeaux, France, May 1998

[BCT$^+$02] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, Giovanni Rimassa, "Jade Programmer's Guide", available at http://sharon.cselt.it/projects/jade, 2002

[BDW01] Bernard Burg, Jonathan Dale, Steven Willmott, "Open Standards and Open Source for Agent-Based Systems", Contribution to FIPA, available at http://fipa.umbc.edu/mirror/members/input.html, April 2001

[BHM98] Markus Breugst, Lars Hagen, and Thomas Magedanz, "Impacts of Mobile Agent Technology on Mobile Communications System Evolution", IEEE Personal Communication Magazine, August 1998

[BHS01] Frédéric Boussinot, Laurent Hazard, Jean-Ferdy Susini, "Programming with Junior", EMP/CMA-INRIA, available at http://www-sop.inria.fr/mimosa/rp, Draft, May 2001

[BHT90] Begon, Harper, Townsend, "Ecology - Individuals, Populations and Communities", Blackwell Scientific Publications, ISBN 0-86542-111-0, Second Edition, 1990

[Bie97] Andrzej Bieszczad, "Advanced Network Management in the Network Management Perpetuum Mobile Procura Project", SCE Technical Report SCE-97-07, July 1997

[BM98] Markus Breugst, Thomas Magedanz, "On the Usage of Standard Mobile Agent Platforms in Telecommunication Environments", Intelligence in Services and Networks (IS&N'98): Technology for Ubiquitous Telecom Services, May 1998

[Bo00] Frédéric Boussinot, "Objets réactifs en JAVA", Presses Polytechniques et Universitaires Romandes, Suisse, April 2000

[Bo01] Raúl Acosta Bermejo, "Reactive Operating System and Reactive Java Objects", in Electronic Journal on Networks and Distributed Processing, No. 11, ISSN 1262-3261, March 2001

[Bou00] Frédéric Boussinot, "Junior Automata", available at http://www-sop.inria.fr/mimosa/rp, October 2000

[Bou01] Frédéric Boussinot, "Java Fair Threads", Technical Report Nr 4139, Institut National de Recherche en Informatique et en Automatique, February 2001

[BP01] Federico Bergenti and Agostino Poggi, "LEAP: A FIPA Platform for Handheld and Mobile Devices", in Proc. of 8th Intl. Workshop on Agent Theories, Architecture and Languages (ATAL'2001), Seattle, USA, August 2001

[BP98] Andrzej Bieszcad, Bernard Pagurek, "Application-Oriented Taxonomy of Mobile Code", IFIP/IEEE Network Operations and Management Symposium (NOMS'98), February 1998

[BPR99] Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa, "Jade - a FIPA-compliant agent framework", in Proc. of the 4th International Conference on the Practical Application of Artificial Intelligence and Multi-agent Technology (PAAM'99), pp.97-108, London, UK, 1999

[BPW98] A. Bieszczad, B. Pagurek, T. White, "Mobile Agents for Network Management", IEEE Communications Surveys, September 1998

[BrM98] Markus Breugst and Thomas Magedanz, "Mobile Agents – Enabling Technology for Active Intelligent Network Implementation", IEEE Network Magazine, Special Issue on Active and Programmable Networks, Vol.12, No.3, May-June 1998

[Bro86] Brooks, R. A., "A robust layered control system for a mobile robot", in IEEE Journal of Robotics and Automation, RA-2, p. 14-23, 1986

[BS02] Frédéric Boussinot, Jean-Ferdy Susini, "The SugarCubes Tool Box", INRIA EMP-CMA/Meije, available at http://www-sop.inria.fr/mimosa/rp, 2002

[BTS^{+}01] David Benjamin, Richard Trudel, Stephen Shew, and Ed Kus, "Optical Services over the Intelligent Optical Network", in IEEE Communications Magazine, September 2001

[BW89] G. Beni and J. Wang, "Swarm Intelligence in Cellular Robotic Systems", in Proc. of the NATO Advanced Workshop on Robots and Biological Systems, Il Ciocco, Tuscany, Italy, 1989

[BZW98] Walter Brenner, Rüdiger Zarnekow, Hartmut Wittig, "Intelligent Software Agents", Springer-Verlag Berlin Heidelberg, 1998

[C^{+}93] Case, J.D. et al., "Simple Network Management Protocol", RFC 1157, May 1990

[CabLZ00] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli, "MARS: a Programmable Coordination Architecture for Mobile Agents", IEEE Computer Magazine, Vol. 33, No.2, February 2000

[CabrLZ00] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli, "A Web Infrastructure for People and Agent Interaction and Collaboration", IEEE Ninth Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), CS Press, NIST (USA), June 2000

[Cal99] K. L. Calvert, "Architectural Framework for Active Networks", Version 1.0, University of Kentucky, Active Network Working Group, available at http://protocols.netlab.uky.edu/~calvert/arch-docs.html, July 1999

[CaLZ00] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli, "Mobile-Agent Coordination Models for Internet Applications", IEEE Computer Magazine, February 2000

[CCM$^+$98] Wilmer Caripe, George Cybenko, Katsuhiro Moizumi, and Robert Gray, “Network Awareness and Mobile Agent Systems”, IEEE Communications Magazine, July 1998

[CD98] Gianni Di Caro, Marco Dorigo, “AntNet: Distributed Stigmergetic Control for Communications Networks”, Journal of Artificial Intelligence Research 9 317-365, 1998

[CFF99] M. Calisti, C. Frei and B. Faltings, "A distributed approach for QoS-based multi-domain routing", AAAI-Workshop on Artif. Intelligence for Dist. Information Networking, Orlando, Florida, July 1999

[CLC99] F. Chantemargue, P. Lerena, M. Courant, "Autonomy-based multi-agent systems: statistical issues", in Proc. of the Third World Multiconference on Systemics, Cybernetics, and Informatics (SCI'99), Orlando, Florida, USA, August 1999

[CLG00] Olivier Crochat, Jean-Yves Le Boudec and Ornan Gestel, “Protection interoperability for WDM Optical Networks”, IEEE/ACM Transaction on Networking, Vol. 8, No. 3, June 2000

[CLZ00] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli, “Weak and Strong Mobility in Mobile Agent Applications”, Proc. Of the 2nd International Conference and Exhibition on The Practical Application of Java, Manchester (UK), April 2000

[CLZ97] Giacomo Cabri, Letizia Leonardi, Franco Zambonelli, "Coordination in Mobile Agent Applications", Università di Modena, Technical Report No. DSI-97-24, October 1997

[Cro98] O. Crochat, “Wavelength Division Multiplexing Networks And Failure Protection”, PhD thesis Nr 1851, Ecole Polytechnique Fédérale de Lausane, 1998

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni, "The Ant System: Optimization by a colony of cooperating agents", in IEEE Transactions on Systems, Man, and Cybernetics Part B, Vol. 26, No. 1, 1996

[DWY99] Piet Demeester, Tsong-Ho Wu, Noriaki Yoshikai, "Survivable Communication Networks", IEEEE Communications Magazine, August 1999

[E1116_01] Eurescom Project P1116, "Carrier Requirements for providing optical transport services to its IP clients", Project SCORPION (Scalable Optical Transport Network), (not yet published), January 2001

[E709_99] Eurescom Project P709, "Planning of Full Optical Network", http://www.eurescom.de, 1999

[E712_98] Eurescom P712 and P815, "Agent Based Computing", a booklet for executives, available at http://www.eurecom.de, 1998

[E712_99] Eurescom Project P712, "Intelligent and Mobile Agents and their Applicability to Service and Network Management", http://www.eurescom.de, 1999

[E718_01] Eurescom Project P718, "Integration of IP over Optical Networks: Networking and Management", http://www.eurescom.de, 2001

[E907_01] Eurescom Project P907, "MESSAGE: Methodology for Engineering Systems of Software Agents", available at http://www.eurescom.de, September 2001

[EB99] M. El-Darieby, A. Bieszczad, “Intelligent Mobile Agents: Towards Network Fault Management Automation”, Sixth IFIP/IEEE International Symposium on Integrated Network Management, May 1999

[FC82] Forgy, C.L., “A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem”, Artificial Intelligence, Vol. 19, 1982

[FIPA01] FIPA, “FIPA Agent Management Support for Mobility Specification”, available at http://www.fipa.org, August 2001

[FM99] Stefan Fünfrocken, Friedemann Mattern, “Mobile Agents as an ArchitecturalConcept for Internet-Based Distributed Applications”, Kommunikation in Verteilten Systemen (KiVS), 11.ITG/GI - Fachtagung, Darmstadt, March 1999

[G805_95] ITU G.805, “Generic Functional Architecture of Transport Networks”, November 1995

[G851_96] ITU G.851.1, “Management of the transport network - Application of the RM-ODP framework”, November 1996

[G872_99] ITU G.872, “Architecture of Optical Transport Networks”, Pre-published version, April 1999

[G874_02] ITU G.874.1, "Optical transport network (OTN) protocol-neutral management information model for the network element view", Pre-published version, January 2002

[Ger00] Ori Gerstel, “Optical Layer Signaling: How Much Is Really Needed?”, IEEE Communications Magazine, October 2000

[GF01] Rune Gustavsson and Martin Fredriksson, "Coordination and Control in Computational Ecosystems: A Vision of the Future", in "Coordination of Internet Agents", Springer, 2001

[GHJ$^+$95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "Design Patterns, Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995

[GJ98] M.A. Gibney and N.R. Jennings, "Dynamic Resource Allocation by Market-Based Routing in Telecommunications Networks", 2nd Int. Workshop on MAS and TCom (IATA'98), Paris, France, 1998

[GLV01] Sergio Gonzalez-Valenzuela, Victor C. M. Leung, Son T. Vuong, "Multipoint-to-Point with QoS Guarantees Using Mobile Agents", in Proc. of Mobile Agents for Telecommunication Applications (MATA'2001), Montreal, Canada, August 2001

[GNS00] Nada Golmie, Thomas D. Ndousse, and David H. Su, "A Differentiated Optical Services Model for WDM Networks", National Institute of Standards and Technology, in IEEE Communications Magazine, February 2000

[GR00] Ornan Gestel and Rajiv Ramaswami, “Optical Layer Survivability: A Services Perspective”, IEEE Communications Magazine, March 2000

[GY95] Germán Goldszmidt and Yechiam Yemini, "Distributed Management by Delegation", in Proc. of the 15th International Conference on Distributed Computing System, June 1995

[HaB99] Alex L. G. Hayzelden, John Bigham, “Software Agents for Future Communication Systems”, Springer-Verlag Berlin Heidelberg, 1999

[Hal93] N. Halbwachs, "Synchronous Programming of Reactive System", Kluwer Academic Pub., 1993

[HB01] Alex L. G. Hayzelden, Rachel A. Bourne, "Agent Technology for Communication Infrastructures", Wiley & Sons Ltd, 2001

[HB99] Alex L. G. Hayzelden, John Bigham, "Agent Technology in Communications Systems: An Overview", Knowledge Engineering Review, Vol:14, No. 3, 1999

[HLG^{+}01] J.-P. Hubaux, J.-Y. Le Boudec, S. Giordano, M. Hamdi, L. Blazevic, L. Buttyan, M. Vojnovic, "Towards Mobile Ad-hoc WANs: Terminodes", in IEEE Communication Magazine, January 2001

[HMW99] L. Hagen, J. Mauersberger, C. Weckerle, "Mobile agents based service subscription and customization using the UMTS virtual home environment", Computer Networks 31, 1999

[Hog89] Dieter Hogrefe, "Estelle, LOTOS und SDL", Springer-Verlag, ISBN 3-540-50477-X, 1989

[HP85] D. Harel, A. Pnueli, “On the development of Reactive Systems”, In logic and Models of Concurrent Systems, NATO Advanced Study Institute on Logics and Model of Verification and Specification of Concurrent Systems ASI Series F, Vol. 13, Springer-Verlag, 1985

[HSB99] Laurent Hazard, Jean-Ferdy Susini, Frédéric Boussinot, "The Junior Reactive Kernel", Inria Research Report 3732, July 1999

[HuB01] Jarle G. Hulaas, Didier Buchs, "An Experiment with Coordinated Algebraic Petri Nets as Formalism for Modeling Mobile Agents", in Proc. of the workshop on Modeling of Object Components and Agents (MOCA'01), Aarhus, Denmark, August 2001

[ITU00] ITU-T (USA), “WORK ON THE AUTOMATIC SWITCHED OPTICAL NETWORK”, Delayed contribution D.697 (WP3/15), 2000

[J01] Nicholas R. Jennings, “An Agent-based Approach for Building Complex Software Systems”, Communications of the ACM, Vol.44, No.4, April 2001

[Jia01] Li Jiang, "Service Assurance for IP-VPN with QoS Guarantees", Professional Thesis at Swisscom Innovations, Confidential, Eurécom, July 2001

[JIDM97] Object Management Group, “JIDM Specification Translation”, Preliminary Specification, NMF Version, doc. P509, February 1997

[JIDM98] Object Management Group, “JIDM Interaction Translation”, Final Submission to OMG’s CORBA/TMN Interworking RFP, October 1998

[Kar00] Stamatis Karnouskos, “Agent-populated Active Networks”, Proceedings of the 2nd International Conference on Advanced Communication Technology, 2000

[KG99] David Kotz, Robert S. Gray, "Mobile code: The Future of the Internet", Workshop Mobile Agents in the Context of Competition and Cooperation (MAC3) at Autonomous Agents, Seattle, Washington, USA, May 1999

[KKL99] Gwan Joong Kim, Young Boo Kim, Hyeong Ho Lee, "A Design of Distributed Network Management System for Open Telecommunication Network", Management Issues - Network and Service Management, Telecom 99, 1999

[KL02] Jong-Seon Kim, Daniel C. Lee, "Dynamic Routing and Wavelength Assignment Algorithms for Multifiber WDM Networks with Many Wavelengths", in Proc. of 2nd European Conference on Universal Multiservice Networks (ECUMN'02), Colmar, France, April 2002

[KMM99] Kwindla Hultman Kramer, Nelson Minar, and Pattie Maes, "Tutorial: Mobile Software Agents for Dynamic Routing", Mobile Computing and Communication Review, Volume 3, No. 2, March 1999

[LDA$^+$98] P. Lagasse, P. Demeester, A. Ackaert, W. Van Parys, B. Van Caenegem, M. O'Mahony, K. Stubkjaer, J. Benoit, "A European view by the HORIZON project and the ACTS Photonic Domain", Draft, June 1998

[Lea00] Doug Lea, "Concurrent Programming in Java - Design Principles and Patterns", Addison Wesley Longman, Second Edition, February 2000

[Lec95] Christopher Leckie, "Experience and Trends in AI for Network Monitoring and Diagnosis", in Proceedings of the IJCAI95 Workshop on AI in Distributed Information Networks", 1995

[LO98] Danny B. Lange, Mitsuru Oshima, "Programming and Deploying Java Mobile Agents With Aglets", Addison-Wesley, ISBN 0-201-32582-9, 1998

[LRD$^+$00] Jingming Li Salina, Daniel Rossier, Manuel Dinis, Laurie Cuthbert, Laurissa Tokarchuk & John Bigham, "Agent-based resource management for 3G networks", Mobile Communications Summit, Galway, Ireland, October 2000

[M$^+$01] P. Marrow et al., "Agents in Decentralised Information Ecosystems: The DIETApproach", AISB'01 Symposium on Information Agents for Electronic Commerce, March 2001

[M3100_96] ITU M.3100, "Principles for a Telecommunications management network", May 1996

[MAF98] OMG, "Mobile Agent System Interoperability Facilities Specification", Joint Submission, http://www.omg.org/cgi-bin/doc?orbos/98-03-09, March 1998

[Mah01] Florent Mahoudeau, "Automatically Switched Optical Networks Routing", Practical course report, Swisscom Innovations, Bern, Switzerland, September 2001

[Mak00] Milla Mäkeläinen, "Agent Mobility in FIPA-OS", Project Report available at http://fipa-os.sourceforge.net/contributions.htm, 2000

[MBN99] James Manchester, Paul Bonenfant, and Curt Newton, "The Evolution of Transport Network Survivability", Lucent Technologies, in IEEE Communications Magazine, August 1999

[MG01] F. Muscutariu, M.-P. Gervais, "On the Modeling of Mobile Agent-Based Systems", in Proc. of Mobile Agents for Telecommunication Applications, Montreal, Canada, August 2001

[MKM98] Nelson Minar, Kwindla Hultman Kramer, and Pattie Maes, "Cooperating Mobile Agents for Mapping Networks", Proceedings of the First Hungarian National Conference on Agent Based Computing, May 1998

[MKM99] Nelson Minar, Kwindla Hultman Kramer, Pattie Maes, "Cooperating Mobile Agents for Dynamic Network Routing", MIT Media Lab, Cambridge, USA, in [HaB99], 1998

[MRK96] T. Magedanz, K. Rothermel, S. Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?", INFOCOM'96, San Francisco, CA USA, March 1996

[NB99] G. Newsome and P. Bonenfant, "A Proposal for Providing Channel-Associated Optical Channel Overhead in the OTN", ANSI T1X1.5/99-002, available at http://www.t1.org/index/0816.htm, January 1999

[NMF98] TeleManagement Forum, “SMART TMN, Technology Direction Statement”, Version 1.1, September 1998

[Num95] Chisato Numaoka, “Biologically Inspired Evolutionary Systems”, Report in Workshop on Biologically Inspired Evolutionary Systems (BIES’95), Tokio, June 1995

[OMG01] Object Management Group (OMG), "Agent Technology – Green Paper", Agent Working Group, OMG Document ec/2000-03-01, March 2001

[OPB00] James Odell, H. Van Dyke Parunak, Bernhard Bauer, "Representing Agent Interaction Protocols in UML", AAAI Agents 2000 conference, Barelona, Spain, June 2000

[PCV99] Menelaos K. Perdikeas, Fotis G. Chatzipapadopoulos, Iakovos S. Venieris, Gennaro Marino, "Mobile agent standards and available platforms", in Computer Networks, Volume 31 (1999), Number 19, August 1999

[Pi98] Gian Pietro Picco, "Understanding, Evaluating, Formalizing, and Exploiting Code Mobility", PhD Thesis, Politecnico Di Torino, 1998

[Pie98] Gian Pietro Picco, "mCode: A Lightweight and Flexible Mobile Code Toolkit", in Proc. of the 2nd International Workshop on Mobile Agents (MA'98), Stuttgart, Germany, September 1998

[PieJan01] Bertrand Emako Lenou, Roch Glitho & Samuel Pierre, "Technologies des agents mobiles et applications", Technique et science informatiques, Montreal, January 2001

[PNJ99] Pietro Panzarasa, Timothy J. Norman, Nicholas R. Jennings, "Modeling Sociality in The BDI Framework", Proc. 1st Asia-Pacific Conf. on Intelligent Agent Technology", Hong Kong, December 1999

[PWW00] B. Pagurek, Y. Wang, and T. White, "Integration of Mobile Agents with SNMP: Why and How", IEEE/IFIP Network Operations and Management Symposium, NOMS'2000, Honolulu, April 2000

[R$^+$01] J. Robadey et al., “Implementing the ASON: interest and critical issues for the operator”, Proc. of 6th European Conference on Network & Optical Communications, June 2001

[RG$^+$95] Rao, A. S., Georgeff, M. P., "BDI Agents: From Theory to Practice, in: Proceedings of the First International Conference on Multi-Agent-Systems (ICMAS)", San Francisco, 1995

[Riv00] Patricia Cuesta Rivalta, "Mobile Agent Management", M.S. Thesis, Carleton University Ottawa, Ontario, Canada, October 2000

[Rob01] J. Robadey, "Optical Nodes Complement", draft contribution for the FASHION (Flexible, Automatically SwitcHed client Independent Optical Networks) Eurescom Project, Swisscom, May 2001

[Roc01] Jordi Roca I Carles, “Dynamic Demands Simulation on IP over WDM Networks”, Diploma Work, Swisscom Innovations, Bern, Switzerland, September 2001

[RoRS01] Daniel Rossier-Ramuz, Daniel Rodellar, Rudolf Scheurer, "Dynamic Protection Set-up in Optical VPN using Mobile Agent Ecosystem", in Proceedings of DRCN'01, Third International Workshop on Design of Reliable Communication Networks, Budapest, October 2001

[Ros02] Daniel Rossier, “A Description of the Generic Network Management Tool”, Technical Report, University of Fribourg, Switzerland, to appear, August 2002

[RRS01] Daniel Rossier-Ramuz, Daniel Rodellar, Rudolf Scheurer, "An Intelligent and Mobile Agent-based Approach for Dynamic Protection Set-up in Future Optical Networks", Proceedings of the fifth Working Conference on Optical Network Design and Modelling (ONDM'01), Vienna, February 2001

[RS00] Daniel Rossier-Ramuz, Rudolf Scheurer, "Implementation of Mobile Agents for WDM Network Management", Hayzelden (ed.), chapter in 'Agent Technology for the Communication Infrastructures', Ed. Wiley, 2000

[RS02] Daniel Rossier-Ramuz, Rudolf Scheurer, "ECOMOBILE: A Mobile Agent Ecosystem for Distributed Network Management", in Proc. of ECUNM'02, 2nd European Conference on Universal Multiservice Networks, Colmar, France, April 2002

[RS97] Rajiv Ramaswami, Galen H. Sasaki, "Multiwavelength Optical Networks with Limited Wavelength Conversion", in Proceedings of Infocom'97, 1997

[RS98] R. Ramaswami and K. Sivarajan, "Optical Networks, A Pratical Perspective", Morgan Kaufmann, 1998

[RS99] D. Rossier-Ramuz, R. Scheurer, "An Introduction to Optical Agents: Intelligent and Mobile Agents for WDM Optical Network Management", in Proceedings of IMPACT'99, Impact of Agent Technology on Telecommunications, Seattle, USA, December 1999

[RSH01] Daniel Rossier-Ramuz, Rudolf Scheurer, Beat Hirsbrunner, “A Mobile Agent Ecosystem for Rule Based Network Oriented Applications”, Technical Report 01-22, Department of Informatics, University of Fribourg, Switzerland, July 2001

[RSP00] Anca Rarau, Ioan Salomie, and Kalman Pusztai, "On Synchronization in a Mobile Environment", in Proc. of Mobile Agents for Telecommunication Applications (MATA'00), Paris, France, September 2000

[S99] Peter Sapati, "Mobile Processing in Distributed and Open Environments", Wiley Series on Parallel and Distributed Computing, Albert Y. Zomaya, Series Editor, 1999

[Sat96] Ken-ichi Sato, "Advances in Transport Network Technologies", Artech House, Inc., Boston and London, 1996

[Sch01] Michael Schumacher, "Objective Coordination in Multi-Agent System Engineering", Springer-Verlag, Berlin, 2001

[SCH99] Michael Schumacher, Fabrice Chantemargue, Béat Hirsbrunner, "The STL++ Coordination Language: a Base for Implementing Distributed Multi-Agent Applications", In Third Int'l Conference on Coordination Models and Languages, COORD'99, April 1999

[SCT01] John Strand, Angela L. Chiu and Robert Tkach, “Issues For Routing In The Optical Layer”, IEEE Communications Magazine, February 2001

[Sel95] Rüdiger Sellin, “TMN - Die Basis für das Telekom-Management der Zukunft”, R.v. Decker’s Verlag, 1995

[Sha98] Uma Shanker, "Autonomous and Mobile Agents in Distributed Network Management and Monitoring System", Distributed Computing on the Web (DCW'98), Proceedings of the Workshops, 1998

[SHMar99] Ruud Schoonderwoerd, Owen Holland, "Minimal Agents for Communications Network Routing: The Social Insect Paradigm", Hewlett-Packard Laboratories, Bristol, UK, in [HaB99], 1999

[Shu00] Shuffle IST Project Consortium, "Specification of System", deliverable D1 of Shuffle Project, available at http://www.ist-shuffle.org, 2000

[SS99] Sajjad H. Shami and Mark C. Sinclair, "Co-evolutionary Agents for Telecommunication Network Restoration", Proc. Recent Advances in Soft Computing'99, Leicester, UK, July 1999

[SSC$^+$98] Onn Shehory, Katia Sycara, Prasad Chalasani, Somesh Jha, "Agent Cloning: Agent Mobility and Resource Allocation", IEEE Communications Magazine, July 1998

[Stu02] Ido Stumpges, "Planning and Analysis of Optical Networks to Support Wavelength on Demand", Professional Thesis at Swisscom Innovations, Confidential, Eurécom, July 2002

[T96] Andrew S. Tanenbaum, “Computer Networks”, International Edition, Prentice-Hall, Inc., Third Edition, 1996

[THL01] Patrick Tullmann, Mike Hibler, Jay Lepreau, "Janos: A Java-oriented OS for Active Network Nodes", in IEEE Journal on Selected Areas in Communication, March 2001

[UK01] A.M. Uhrmacher, M. Krahmer, "A Conservative, Distributed Approach to Simulating Multi-Agent Systems", in Proc. of European Multi-Conference on Simulation (ESM'2001), Prague, 2001

[UKL02] A.M. Uhrmacher, Bernd Kullick, Jens Lemcke, "Reflection in Simulating and Testing Agents", in Proc. of the 16th European Meeting on Cybernetics and System Research (EMCSR'02), 2002

[UTT00] Adelinde M. Uhrmacher, Petra Tyschler, Dirk Tyschler, "Modeling and Simulation of Mobile Agents", in “Future Generation Computer Systems”, Vol. 17, 2000

[VS99] Griselda Navarro Varela, Mark C. Sinclair, "Ant Colony Optimisation for Virtual-Wavelength-Path Routing and Wavelength Allocation", in Proc. Congress on Evolutionary Computation (CEC'99), Washington DC, USA, July 1999

[VSN+01] Eve L. Varma, Sivakumar Sankaranarayanan, George Newsome, Zhi-Wei Lin, and Harvey Epstein, "Architecting the Services Optical Network", IEEE Communications Magazine, September 2001

[Whi00] Tony White, "SynthECA: A Synthetic Ecology of Chemical Agents", Ph.D., University of Carleton, Ottawa, Canade, available at http://www.sce.carleton.ca/netmanage/publications.html, August 2000

[WhP99] Tony White, Bernard Pagurek, "Emergent Behavior and Mobile Agents", MAC3, Mobile Agents in the Context of Competition and Coordination, Seattle, WA, USA, May 1999

[WJ95] Wooldridge and N.R. Jennings, "Agent Theories, Architectures, and Languages: a Survey", In M. Wooldridge and N.R. Jennings, editors, Intelligent Agents, number 890 in LNCS, pages 1-39. Springer Verlag, 1995.

[WP99] Tony White, Bernard Pagurek, "Distributed Fault Location in Networks Using Learning Mobile Agents", Proceedings of PRIMA’99, Springer-Verlag Pub., Kyoto Japan, December 1999

[WPB+98] Tony White, Bernard Pagurek, Andrzej Bieszczad, George Sugar, Xuong Tran, “Intelligent Network Modelling Using Mobile Agents”, In Proceedings of the IEEE Global Telecommunications Conference GLOBECOM 98, November 1998

[WPB99] Tony White, Bernard Pagurek, Adrzej Bieszczad, "Network Modeling For Management Applications Using Intelligent Mobile Agents", special issue on Mobile Agents of the Journal of Network and Systems Management, September 1999

[XD00] Dianxiang Xu and Yi Deng, "Modeling Mobile Agent Systems with high Level Petri Nets", in Proc. of IEEE International Conference on Systems, Man, and Cybernetics'2000, Invited paper, Nashville, October 2000

[YGY91] Yemini, Y., Goldszmidt, G. and Yemini, S., "Network Management by Delegation", in The Second International Symposium on Integrated Network Management, Washington, DC, April 1991

[Zbi01] Patric Zbinden, “Generic Network Management Tool”, Schlussbericht zur Seminararbeit, University of Fribourg, Switzerland, October 2001

[ZJM00] Hui Zang, Jason P. Jue, Biswanath Mukherjee, "A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Networks", Optical Networks Magazine, January 2000

[ZJS^{+}01] Hui Zang, J.P. Jue, L. Sahasrabuddhe, R. Ramamurthy, B. Mukherjee, "Dynamic Lightpath Establishment in Wavelength-Routed WDM Networks", IEEE Communications Magazine, September 2001

[ZP97] Jim Zyren, Al Petrick, "IEEE 802.11 Tutorial", available at http://www.computer.org/students/looking/summer97/ieee802.htm, 1997

[ZZ98] Dianlong Zhang, Werner Zorn, "Developing network management applications in an application-oriented way using mobile agent", Computer Networks and ISDN Systems, Vol. 30, 1998