

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Institut National des Sciences Appliquées de Toulouse (INSA Toulouse)

Discipline ou spécialité :

Informatique

Présentée et soutenue par :
Ismael Bouassida

le : 19 février 2011

Titre :
Gestion dynamique des architectures logicielles pour
les systèmes communicants collaboratifs
Dynamic Software Architecture Management for
Collaborative Communicating Systems

JURY

Khalil Drira, Ahmed Hadj Kacem
Mohamed Jmaiel, Riadh Robbana

Ecole doctorale :
Mathématiques Informatique Télécommunications (MITT)

Unité de recherche :

Directeur(s) de Thèse :

Christophie Chassot, Khalil Drira, Mohamed Jmaiel

Rapporteurs :

Flavio Oquendo,RiadhRobbana

To my parents

Acknowledgements

It is a pleasure to thank all the people who participate to made this thesis possible. It is difficult to overstate my gratitude to my Ph.D. supervisors, Khalil Drira, Christophe Chassot and Mohamed Jmaiel. This thesis would not have been possible without their kind support, trenchant critiques, probing questions, and remarkable patience. I wish to thank my wife, my parents, my brother and my friends Germán Sancho and Riadh BenHalima for helping me get through the difficult times, and for all the emotional support, camaraderie, entertainment, and caring they provided. I am indebted to my many colleagues in the group OLC at LAAS-CNRS and in the unit ReDCAD for providing a stimulating and collaboration environment which made learning a real fun. Finally my thanks go to all technical and administrative staff of LAAS-CNRS whose assistance has greatly facilitated the achievement of this work.

Contents

1 Introduction **1**

2 Adaptation for Context-Aware for CCS **5**

2.1 Adaptation Goals . . . 6
2.1.1 Corrective Adaptation . . . 6
2.1.2 Evolutional Adaptation . . . 6
2.1.3 Perfective Adaptation . . . 6
2.2 Objects of Adaptation . . . 6
2.2.1 User Interface Adaptation . . . 7
2.2.2 Service Adaptation . . . 7
2.2.3 Middleware Adaptation . . . 8
2.2.4 Transport Adaptation . . . 8
2.2.5 Content Adaptation . . . 8
2.3 Model Based Adaptation Approaches . . . 9
2.4 Autonomic Computing . . . 9
2.4.1 Basic concepts of autonomic computing . . . 10
2.4.2 Network level . . . 11
2.4.3 Application level with Agents . . . 12
2.4.4 Frameworks and architectural proposals . . . 13
2.5 "“Models@runtime” . . . 14
2.6 Collaboration . . . 15
2.7 Context . . . 15
2.7.1 Definition . . . 15
2.7.2 Taxonomy of the Contextual Data . . . 16
2.7.3 Context-Awareness . . . 17
2.7.4 Challenges . . . 18
2.8 Adaptation in CCS . . . 19
2.8.1 Classification . . . 19
2.9 Solutions for Context Adaptation in CCS . . . 21
2.9.1 Model . . . 22

2.9.2 Architecture . . . 23
2.9.3 Platform . . . 23
2.10 Discussion . . . 25

3 Proposed Approach **27**
3.1 Generic Modelling Approach . . . 27
3.1.1 Multi-level Architecture Modelling . . . 28
3.1.2 Generic Refinement and Selection Procedures . . . 28
3.2 Modelling of Communicating Architectures . . . 30
3.2.1 Design Time Side . . . 31
3.2.2 Run Time Side . . . 31
3.2.3 Context Capture and Representation . . . 33
3.3 A Framework for Communicating Systems . . . 33
3.3.1 Design Time Side . . . 34
3.3.2 Run Time Side . . . 37
3.4 Graph Models for Architectural Transformation . . . 37
3.4.1 Basic approach for Graph Transformation . . . 37
3.4.2 The Double PushOut Approach . . . 38
3.4.3 The Node Label Controlled Mechanism . . . 39
3.4.4 The Neighbourhood Controlled Embedding Mechanism . . . 40
3.4.5 Our Approach: Combining DPO and edNCE . . . 42
3.5 Refinement and Selection Implementation . . . 44
3.5.1 Application–Collaboration Refinement and Selection . . . 44
3.5.2 Collaboration–Middleware Refinement and Selection . . . 45
3.5.3 Deployment Service . . . 50
3.5.4 Refinement and Selection Illustration . . . 50
3.6 Conclusion . . . 52

4 Case Studies and Performance Study **53**
4.1 Graph Matching and Transformation Engine . . . 53
4.2 ERCMS Case Study . . . 56
4.2.1 ERCMS Case Study Description . . . 58
4.2.2 Context Description . . . 61
4.2.3 Initial Refinement and Adaptation Process Examples . . . 65
4.3 Web Service-Based Applications Case Study . . . 71
4.3.1 The Graph Grammar rule-oriented reconfiguration . . . 72
4.3.2 The FoodShop Case Study: A Web Application . . . 75
4.3.3 Application of our Approach . . . 76
4.4 Evaluation Experiments and Simulations . . . 78
4.4.1 Adapting Architecture to Organization Changes . . . 80
4.4.2 Adapting Architecture to Resources Changes . . . 82

4.5 Conclusion 85

5 Conclusion **87**

6 Extended French Abstract/Résumé **91**
6.1 Introduction 91
6.2 Approche générique de modélisation 93
6.2.1 Modélisation Architecturale Multi-niveau 93
6.2.2 Procédures génériques de Raffinement et de Sélection 95
6.3 Application aux Systèmes Communicants 96
6.3.1 Niveau application 97
6.3.2 Niveau collaboration 99
6.3.3 Niveau middleware 100
6.3.4 Niveau infrastructure 100
6.3.5 Implantation du raffinement et de la sélection 100
6.4 Cas d'étude 102
6.5 Expérimentations et évaluations 105
6.6 Conclusion 107

Author's Publications **113**

Bibliography **117**

List of Figures

3.1 Multi-level Architecture Modelling . . . 28
3.2 Layered Architecture Modelling . . . 32
3.3 Model-Oriented Reconfiguration Framework. . . . 35
3.4 Generic Collaboration Ontology . . . 36
3.5 Dangling Edges Problem . . . 38
3.6 DPO Approach Example . . . 40
3.7 NLC Approach Example . . . 42
3.8 Combining DPO and edNCE . . . 43

4.1 The Graphical User Interface . . . 55
4.2 The XML Graph File . . . 57
4.3 The XML Rule File . . . 58
4.4 ERCMS Mission Description . . . 59
4.5 The ERCMS Ontology Description . . . 62
4.6 General Structure of our Context Ontology . . . 63
4.7 Instance Example from our Context Ontology . . . 64
4.8 Instance Example of a Device Hardware Profile . . . 65
4.9 Adaptation for Connectivity and QoS Preservation . . . 70
4.10 Graphical Representation of $GG1$ Production $p1$ . . . 73
4.11 Graphical representation of $GG2$ production $p1$ . . . 74
4.12 Graphical Representation of $GG2$ Production $p2$ . . . 74
4.13 Proposed Architecture Applied to The FoodShop Case Study . . . 75
4.14 Initial Architectural Configuration . . . 76
4.15 Monolithic Service Substitution . . . 77
4.16 Composite Service Substitution According to WS Cost . . . 79
4.17 Execution Time of Applying SWRL (Table 4.1) . . . 80
4.18 Execution Time of Applying $GG_{COLLAB \rightarrow EBC}$ . . . 81
4.19 Results of the Connectivity Adaptation . . . 83
4.20 Energy Evolution on Nodes . . . 84

6.1 Approche de modélisation multi-niveau. . . . 94
6.2 Cadre de modélisation pour les SCC et technologies d'implantation. . . . 99

List of Tables

2.1 Examples of Contextual Data 17

3.1 Generic Refinement Procedure 29
3.2 Generic Selection Procedure. 30
3.3 A SWRL Rule for Establishing DataFlows 44
3.4 The Graph Refinement Procedure 46
3.5 Refinement Graph Grammar $GG_{COLLAB \rightarrow EBC}$ 47
3.6 Dispersion Function 48
3.7 Relative Cost Function 48
3.8 Context Aware Function 49

4.1 Establishing a Cooperation Flow SWRL Rule 68
4.2 Critical Situation Discovery SWRL Rule 68
4.3 Adding a New Investigator SWRL Rule 69
4.4 GG1: The Monolithic Substitution of a WS 73
4.5 GG2: Service Composite Substitution Policy 74

6.1 Procédure générique de raffinement 94
6.2 Procédure générique de sélection. 95
6.3 Procédure d'adaptation au contexte 97
6.4 Procédure de calcul de la dispersion 98
6.5 Procédure de calcul du coût relatif 98
6.6 Procédure de raffinement basée grammaire de graphes 102

Chapter 1

Introduction

The design of Self-configuring Systems that adapt their service to dynamically changing environments is among the main research directions of autonomic computing and communication. Providing solutions for distributed software systems supporting group communication requires dynamic management of evolving group membership, and dynamic connections of deployment nodes. It also requires dynamically distributing software entities on remotely interconnected deployment nodes. For a number of group communication-based applications, reconfiguration anticipation is important. In order to be applicable in different situations, designers of such applications have to ensure tractability and scalability of elaborated solutions when changing from several to thousands users, nodes, components and services. The particular class of collaborative software applications can especially benefit from communicating systems' advanced capabilities. Collaborative applications are distributed systems where several users act in a coordinated manner in order to achieve a common goal. In such applications, users are organized into structured groups where each participant may play a role. Several tools are provided to users in order to communicate, share documents, and interact in an efficient manner. Many collaborative applications have been successfully developed within the desktop paradigm in different domains such as e-learning, engineering, design, etc. Exploiting features such as context information, mobility, etc. may enhance the functionality and usability of collaborative applications. The fact that users are located in the same environment, using personal mobile devices, and are connected through wireless connections instead of using desktop computers remotely connected through the Internet represents a very different approach for the design and use of such

applications. Contextual information can be exploited to detect potential collaboration situations between several users and then spontaneously propose them to join a session where they can interact together. This implicit way of collaboration differs from the classic desktop collaboration where activities are initiated explicitly. Therefore, collaboration and even more context-aware collaboration raises several promising challenges. The concept of *context* is crucial in applications that support collaboration. Indeed, if a system is able to correctly detect context changes and react to them in an intelligent way, it will be able to deliver a much richer user experience. In order to correctly exploit contextual information, context has to be represented with models that formalize the considered contextual data. Also, dynamic context data have to be acquired and introduced into instances of the available context models. Once context is acquired, the system has to react to context changes; this is called adaptation. In the case of collaborative applications, the main objects of adaptation are the collaborative sessions which are established to support the users' activities, and the components that manage such sessions. Providing generic and scalable solutions for automated self-reconfiguration in group collaboration support systems can be driven by rule-based reconfiguration policies. To achieve this goal, we elaborate a dynamic graph-based modelling approach and we develop structural models that can represent the different interaction dependencies from different configuration-related point of views: communication flows between the distributed machines, the networked deployment nodes, and the service composition. Our solution is based on graph grammars rewriting. We define architectures as graphs where vertices correspond to deployment nodes, software services and their internal components. Moreover, we provide graph transformation to specify rules for changing deployment architecture while being in conformance to an architectural style. Dealing with dynamically evolving architectures requires at least describing the set of consistent architecture instances. This is mandatory to validate the management models and verify architectural constraint preservation [GDD04]. An architecture instance is considered consistent, if its corresponding description graph can be generated by a sequence of graph grammar productions. Our approach supports formal verification for correctness and safety proofs. We implemented a graph rewriting system that ensures automating our approach with a high performance making it tractable for large scale configurations. In order to handle the complex design of communicating collaborative system architectures and the related adaptation issues, we propose a multi-layer modelling approach. This simplifies the problem by separating concerns in different

abstraction layers. This approach assures generic solutions for automatic context aware adaptation.

Our approach is based on the observation that semantic data analysis can be exploited to manage priorities and more generally to manage communications. This allows us to represent, in a richer way, the semantics of the managed systems. Consequently, these semantics offer a vector for ensuring interoperability between interactive and autonomous heterogeneous management entities. They can contribute to enforce adaptability where they support, by capturing semantics from organizational, environmental and operational viewpoints, the automation of the accomplishment of management actions on the overall distributed environment. We propose a combined use of both ontologies and policies to enforce, in a context-aware approach, the adaptive behavior of autonomous management entities. Modelled and ontological data are combined at run time to determine the current context, which policies are applicable to that context, and what services and resources should be offered to users and applications. Ontologies have demonstrated their benefits to support non-functional properties, and to manage QoS in distributed systems. Our ontology for communications not only describes devices and communications in terms of functional and non-functional properties, but also allows to manage priorities of exchanged messages. This work focuses on the presentation of the developed ontologies and emphasizes the beneficial effects of semantic for self-adaptation behaviors. To illustrate the proposed models and their transformations, we consider a case study of Emergency Response and Crisis Management Systems (ERCMS) involving several cooperating participants which have different roles and functions. We consider also the case of a Web Service-Based Applications: the Foodhop. Using graph grammars, We handle architectural reconfiguration by defining architectures as graphs where vertices correspond to software services and their operations. We use graph transformation to specify rules for deployment architecture changes (evolutions) while being in accordance with the architectural style

This document is organized as follows. Chapter 2, details research activities dealing with adaptation and focus on autonomic systems. The concepts of collaboration and context are detailed. A new approach that focuses on managing reconfiguration complexity in a runtime environment: “model@runtime” is presented. In the last part, We focus on context adaptation in collaborative communicating systems.

Chapter 3, details our modelling approach for collaborative communicating architectures. This approach defines abstraction levels to tackle level related problems indepen-

dently. This separation in levels favours the realization of a multi-level adaptation, that takes into account both high level requirements and low level constraints. Generic procedures for refinement and selection are presented in this chapter. Refinement ensures that lower level models actually implement associated higher level models. Selection allows choosing a model among a set of candidates at a given level. Ontologies, SWRL rules, graphs, and graph's grammars are presented. These rules handle both transforming a given architecture within the same layer, and architectural mappings between different layers.

Chapter 4 presents the graph matching and transformation engine that we developed to implement the graph grammars approach. The ERCMS scenario and the FoodShop application scenario are presented.

In ERCMS scenario, mobile actors collaborate to manage emergency situations. We propose three ontologies to model different aspects related to communication as the activity, collaboration and context. Adaptation actions are also proposed for adaptation in communicating architecture. We presented refinement and selection procedures as well as examples of external context and of resources context changes adaptations.

In the FoodShop scenario, we studied the dynamic reconfiguration of Service Oriented Architectures for maintaining the Quality of Service in perturbation-prone environments. Our approach uses graph grammar theories to implement rules that characterize the set of the different configurations candidate to solve composite or monolithic reconfiguration.

To validate our approach, we conducted an experimental evaluation using our graph matching and transformation engine, GMTE, and the SWRL rule execution engine, Jess. Our results show that ERCMS participant's resources remain in service as required for mission achievement thanks to our adaptation approach.

Finally, conclusions to this work are presented and several research perspectives extending specific aspects of the present work are identified.

Chapter 2

Adaptation for Context-Aware Collaborative Communicating Systems

Designing and implementing self-adaptive communicating systems are a complex task. To handle this complexity, several studies showed the need to lay on model-based design approaches associated with automated management techniques. In the first part of this chapter, we detail research activities dealing with adaptation. In the second part, we focus on autonomic system and we explain how the reconfiguration issue is addressed in autonomic computing. After that, we detail the concepts of collaboration and context. In next part, we present research activities that present a new approach: "model@runtime". These approaches focus on managing reconfiguration complexity in a runtime environment. Finally, We focus on adaptation in collaborative communicating systems and we emphasis the context adaption solutions.

Adaptation is the operation of making changes to a program or an information system to maintain its functionalities and, if possible, to improve its performance in a certain execution environment. In the area of communicating systems and context-awareness, adaptation is extended by the concept of adaptability that characterizes system's capability to change its behavior to improve its performance or to continue its role in different environments.

2.1 Adaptation Goals

After developing an information system, different reasons can lead to adaptation. These reasons can be for corrective, evolutional or perfective purposes [KBC02]:

2.1.1 Corrective Adaptation

In some cases, we can notice that the application does not behave properly or as expected. The corrective adaptation is a solution to identify the application module that causes the problem and to replace it by a new correct module. This new module provides the same functionality as the former.

2.1.2 Evolutional Adaptation

When developing an application, some features are not taken into account. With the changing needs of the user, the application must be extended with new features. This extension can be achieved by adding one or more modules to provide new features or modifying existing modules to enrich their functionality while keeping the same application architecture.

2.1.3 Perfective Adaptation

The objective of this kind of adaptation is to improve application performance. For example, we can have a module that receives a lot of requests and fails to meet them. To avoid system performance degradation, we can duplicate this module to share the requests with the existing one.

2.2 Objects of Adaptation

Adaptation approaches target many levels of information systems: User interfaces, Content, Services, Middleware and Transport.

2.2.1 User Interface Adaptation

User interface adaptation deals with producing Human-Computer means that can be deployed and used on different types of terminals while meeting user's preferences. Major existing work related to user interface adaptation is based on models that describe the different aspects of interaction between humans and machines. These models are implemented in different XML or UML languages like UMLi [PP00] and XIML [PE02]. These models are used to produce the adequate user interface code corresponding to the given XML or UML description. In the existing user interface adaptations, we distinguish two techniques: User Interface transformations and User Interface generation. In the first technique, the adaptation process starts from a description language which is very close to the user interface code that must be generated. This solution is adopted to produce adapted Web pages to different terminals starting from an XML description. In this kind of adaptation, style sheets (like XSLT) are used to specify replacement rules of XML tags by scripts that can be directly used by the target device. The second approach generates user interfaces code starting from a high level description which is completely independent from the target programming language of the user interface. SEFAGI [CL04] is an example of a platform using the generation technique to ensure user interface adaptation.

2.2.2 Service Adaptation

Service-Oriented Architecture (SOA) paradigm is based on dynamically publishing and discovering services. This kind of architectures provides the possibilities to dynamically compose services for adapting applications to contexts. Service descriptions are published, via the registry, by service providers and dynamically discovered by service requesters. There are various implementation technologies like COM/ DCOM (Component Object Model/ Distributed Component Object Model [Rog97, GG97] of Microsoft, the EJB (Enterprise Java Beans) [MH00, Tho98] of Sun Microsystems, and CCM (CORBA Component Model) [OMG99] of OMG (Object Management Group). We can also consider JXTA [STS03] the peer to peer framework or .NET [Mic01] of Microsoft.

2.2.3 Middleware Adaptation

Other frameworks are proposed to provide adaptability at the middleware level. In [NH04], an adaptive framework for supporting multiple classes of multimedia services with different QoS requirements in wireless cellular networks is proposed. [STS03] proposes CME, a middleware architecture for service adaptation based on network awareness. CME is structured in the form of software platform which provide network awareness to applications, and manage network resources in an adaptive fashion.

2.2.4 Transport Adaptation

At the transport level, [ESD03] provides frameworks for designing transport protocols whose internal structure can be modified according to the application requirements and network constraints. Adaptation actions correspond to the replacement of a processing module or micro-protocol by another following a plug and play approach.

2.2.5 Content Adaptation

The adaptation of multimedia content has been the subject of considerable research. Several techniques for adapting the delivered data to the user have been proposed. These techniques are based on textual transformations [NSS01], [HL96], image transcoding [WA03], or processing video and audio. One of the major issues in content adaptation is where the decision-making and transformations are made. In the literature, three general approaches have been proposed according to the location of adaptation processing between the source that hosts the content and destination that requests it: (i) on the content provider side, (ii) on the requester side, and (iii) at an intermediary (proxy) between the data source and the client. The content provider-side solutions have some drawbacks. Indeed, the changes made on the content induce a calculation load and consequent resource consumption on the server. However, this approach is very suitable for situations with low variability and low adaptation frequency regarding the simplicity of its implementation. But it is not reliable for cases where the adaptation is triggered frequently. The content requester-side approach is suitable when the transmission characteristics are less critical than the display limits of the user device [NKSB99]. However, the usual complexity of adaptation processing hampers the wide adoption of this approach [PAC+01]. Moreover, the client's terminal usually have lim-

ited computing capacity, power and storage. When considering the proxy solution, the flexibility of positioning the adaptation mechanisms on the best content distribution point is a major advantage compared to the other approaches (provider and content sides). However, the proxy must be a trusted party by the provider and the requester of the content. In addition, the third party may charge for the service it provides and the resources it employs to perform the adaptation for the receiver. Therefore, accounting mechanisms should be incorporated in the proxy solution in order to keep track of the amount of resources utilized and the usage of data.

2.3 Model Based Adaptation Approaches

There are many relevant contributions concerning system architecture adaptation. This kind of approaches uses model-based strategies to apply the necessary transformations on the systems architecture to adapt it to environment and requirement changes. These strategies define or reuse models describing the system software architectures. These models are also known as ADLs (Architecture description languages). We distinguish between three general ADL types: formal ADLs like graph grammars [HIM98] and Petri nets [Mur89], semantic ADLs using ontologies [ZPM+07] and technical ADLs using XML deployment languages [DvdHT02]. The technical ADLs can be proprietary or implementing the formal and the semantic ADLs. These ADLs are used to guarantee the architectural evolution and correctness during the different predictable and unpredictable changes in the system's environment. The necessary actions to achieve such adaptations are specified using rules according to the application runtime context. [CLC07] is an example of these approaches defining a complete model based architecture adaptation at the service, content and user interface levels. [CGD+06] presents another model based method using graph grammars to adapt cooperative information systems to situation changes at the communication level.

2.4 Autonomic Computing

Managing autonomic systems should consider abstraction levels. More precisely , it has to be managed in a coordinated manner both within and between these abstraction levels. Distinguishing these abstraction levels allows designers and developers to master

specification and implementation of adaptation rules. The autonomic concept and systems suggested in the literature are defined in various ways. In our study, we target the solutions given within and between different levels.

2.4.1 Basic concepts of autonomic computing

In this section, we present research activities that focus on the need of autonomic computing; and the first concepts are introduced. They present the autonomic control loop in detail. For IBM in [KC03], Autonomic computing systems are those systems that automatically manage themselves by carrying out tasks that have been traditionally performed by computer specialists. The self-management tasks are well defined. Self-optimization is the ability of the system to optimize the use of resources. Self-healing is the ability of the system to detect faulty behavior, and perform self-repair. Self-configuration is the capacity of the system to change its structure and behavior. Finally, self-protection is the ability of the system to detect intrusions, policy violation, etc. and recover from them.

In [IBM06], the internal functional architecture of an Autonomic Element was introduced. This architecture is composed of a number of functional modules that enable the expected autonomic behavior through a set of autonomic operations. The autonomic operations are achieved using a self-adjusting control loop. Inputs to the control loop consist of various status signals from the system or component being controlled, along with policy-driven management rules that orchestrate the behavior of the system or component. Outputs are commands to the system or components to adjust its operation, along with messages to other autonomic elements.

In [DDF+06], the authors present a survey of the state of research in autonomic communications and present the autonomic control loop for network communication. They address the five interlinked perspectives of the design and analysis of decentralized algorithms; the modelling, handling and use of context, novel and extended programming approaches; issues and approaches for addressing security and trust; and systems evaluation and testing. Several challenges are presented in this work, such as interaction with stranger, information reflection and collection, lack of centralized goals and control, meaningful adaptation, cooperative behavior in the face of competition, heterogeneous services and semantics. They match these challenges against the cross-cutting issues and show the technical ideas emerging from each issue when addressing each

challenge.

2.4.2 Network level

In this section, we present some research activities that consider the network's point of view of autonomic computing. The work presented in [vdMDS+06], explains Autonomic Network definition in more detail, links it to the foundational principles of architecture for Autonomic Network Management, and provides guidance on how to develop specifications and best practices for building Autonomic Communication Systems. The purposes of this paper are: firstly, to define required terminology necessary to support the realization of an Autonomic Communications Framework. Secondly, to define a flexible framework that can be used as the foundation of autonomic network management. Finally, and to define how this framework can be used to build Autonomic Communications Environments. This paper focuses on four research areas that, in combination, define the foundations of autonomic network management: Modelling and Knowledge Engineering for Autonomic Network Management; Automating Network Configuration via Model-Centred Policy Analysis and Deployment; Network Algorithms and Processes; Architecture and Methodology for Autonomic Network Management.

The work presented in [ABB+06] gives an overview of the different architectures that support the design, implementation and deployment of autonomic systems. This paper presents the issues related to the design, implementation and deployment of autonomic networks. It focuses on the autonomic-management approaches. Authors present the motivation behind the emergence of autonomic, self-managed systems and the required features of such architectures. Then, they propose different architectures. In addition, they discuss the complexity related to the autonomic information modelling and the autonomic behavior. Moreover, they present the potential of bio-inspired techniques and compare it with autonomic concepts.

The work presented in [MHSC04], examines the trends in next-generation wireless access networks that will lead to a significant increase of the costs associated with the deployment and configuration of such networks. The authors propose, the concept of a self-deploying, self-configuring radio access network to resolve these issues. They propose algorithms from economic theory, ecology/population growth models, or cellular automata. An example, taken from the field of cellular automata for a radio network

capable of self-adaptation to achieve universal coverage in a simplified environment was examined.

In [XB05], the authors present a mechanism for QoS-aware service composition and adaptation of end-to-end network service for autonomic communication. They introduce a service provisioning framework based on the autonomic communication principle, covering a number of essential functions: domain discovery, domain reacheability, composition, cross-domain contracting, intra-domain provisioning, domain-wide monitoring, and adaptation. Through domain graph abstraction, they reduce the domain composition and adaptation problem to the classic-multiconstrained optimal path problem. They develop a set of new algorithms for QoS-aware service composition and adaptation. Their composition algorithm finds a series of consecutive domains spanning end-to-end and select appropriate service class in each domain such that the overall QoS requirements are satisfied. The algorithm also minimizes the overall cost of the path. As the network condition changes over time or as the user roams across domains, the adaptation algorithm ensures that the QoS requirements of the communication path are respected as long as it is feasible to do so, while minimizing the cost of such adjustments. Together, these algorithms are designed to support self-configuration, self-optimization, and self-adaptation of network communication services. They address the service provisioning problem at the domain level; the algorithms can function over heterogeneous intra-domain provisioning mechanisms, and more importantly, provide hard end-to-end QoS guarantees over "soft" intra-domain QoS schemes.

2.4.3 Application level with Agents

In this section we focus on research activities that consider a Multi-Agent approach for autonomic systems. The work presented in [LV07b], describes a Multi-Agent approach to the modelling and design of Collaborative Ubiquitous Environments. These environments support collaboration among persons in a context of ubiquitous computing. In particular, the paper shows how research in the topic of Multi-Agent Systems environment has provided both modelling abstractions and concrete computational supports for the analysis, design and engineering of Collaborative Ubiquitous Environments. In particular, the Multilayered Multi-Agent Situated System model was applied to represent and to manage several types of awareness information (both physical and logical contextual information) which is an essential part of a Collaborative Ubiquitous En-

vironment. This work differs from other existing proposals that employ agents and agent-based infrastructures simply as a middleware for the design and implementation of pervasive computing systems. The authors also present a selection of the available platforms developed in this context and discuss their suitability to support the development of Collaborative Ubiquitous Environments.

The work presented in [JI07] begins by discussing some of the general issues of complex systems and explains why the agent-based approach is attractive. The article investigates the application of multilevel hypernetworks in team robotics as an example of a complex interaction-based system. The authors show how hypernetworks can represent multilevel relational dynamics by the in-depth analysis of a robot soccer simulation game. They have sketched a mathematical formalism for representing, the relational structure between agents.

2.4.4 Frameworks and architectural proposals

In [LP06], the authors present the Accord programming framework that extends existing programming models/frameworks to support the development of autonomic applications in wide area distributed environments. The framework was built on the separation of the composition aspects (e.g., organization, interaction, and coordination) of elements from their computational behaviors that underlies the component and service based paradigm, and extends it to enable the computational behaviors of objects/components/services as well as their organizations, interactions, and coordination to be managed at runtime using high level rules. The operation of the proposed framework is illustrated using a forest fire management application.

In [YYF00], the authors present The NESTOR system. The NESTOR system addresses the needs of network management automation and of minimizing the management of small home networks due to limited resources. The NESTOR system combines several techniques from object modelling, constraint systems, active databases, and distributed systems. In the NESTOR system, managers operate on an unified object-relationship model of the network using a rich set of operations that support rollback and/or recovery of operational configuration states. Declarative constraints prevent the known configuration inconsistencies, and in conjunction with policy scripts may automatically propagate changes to maintain consistency. Protocol proxies are used to provide much of this functionality with little or no changes in the network clients. A

protocol for replication and distribution of the directory assures availability and operational efficiency. Other research activities try to analysis Autonomic Computing Systems.

The work presented in [Lit07], investigates performance analysis techniques used by an autonomic manager. It looks at the complexity of the workloads and presents algorithms for computing the bounds of performance metrics for distributed systems under asymptotic and nonasymptotic conditions, with saturated and nonsaturated resources. The techniques used are hybrid, making use of performance evaluation and linear and nonlinear programming models. The authors treat autonomic transactional distributed systems. The system is modelled with a Queuing Network Model.

2.5 "Models@runtime"

Other research activities address problems related to the management of huge information associated with runtime situations in self-adaptive software. A new approach to manage complexity in runtime environments is referred as "models@runtime". A "model@runtime is a causally connected self-representation of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective." as defined in [BBF09].

Authors in [MBJ+09], present an approach for specifying and executing dynamically adaptive software systems. This approach combines model-driven and aspect-oriented techniques. This work, which is a part of the EU-ICT DiVA project (Dynamic Variability in complex, Adaptive systems), addresses two drawbacks related to adaptation and evolution management by using software models at runtime as well as at design time. The authors intend to tram the explosion in the number of artefacts considered.

The authors in [Mao09] present model-based traces as runtime models and traces analysis methods. They focus on the scenario-based trace as a runtime model and on the metrics and operators to analyze it. They illustrate their proposed methods using different application scenarios. The syntax and semantics of various types of the model-based traces in this work are not formally defined.

Authors in [GvdHT09] provide operations control center through an adaptive vision in which human users can understand and manage software systems at runtime. Their approach, entitled "architectural runtime configuration management", creates a model

that captures an adaptive system's configurations and corresponding behaviors and organizes them in a historical graph of configurations.

2.6 Collaboration

Collaborative activities belong to the Computer Supported Collaborative Work (CSCW) domain [CS99] that started in the 90s. CSCW emerges from four main domains:

- Social sciences (sociology, organization theories) take into account the people, the organizational structures, the group efficiency, and their benefits or disadvantages;
- Cognitive sciences (distributed artificial intelligence) with interpretation of data semantics, planning and assistance for realizing common tasks;
- Human-machine interfaces, for designing multiuser interfaces;
- Distributed computing science (distributed systems and networking) for information storage, transfer and exchange.

Collaborative activities involve users organized within groups that communicate and act in a coordinated manner for achieving a common goal [EGR91, KK88]. This class of applications is the most general one that can be considered. It generalizes the particular case of a single user interacting with a set of pervasive devices and services, classically considered in communicating systems. In our work, more general interactions are considered: user-environment, user-user and/or user-group interactions.

2.7 Context

It is well-known that the concept of context plays a central role in communicating systems. Since it is a complex concept, there are almost as many definitions of context as research projects dealing with it.

2.7.1 Definition

From the adaptation point of view, a definition of context must take into account any information that, in case of change, is likely to change any aspect of the application.

This information should consider a variety of elements such as technical and communicational data, and social aspects of the user's behavior. It should also enhance the development of context-aware applications. Mobility and collaboration modes of entities implied in the applications are specially important in communicating systems environments. Indeed, users can move as well in space as in time, and the concept of context must take these changes into account. Moreover, we must consider that devices may be embedded and able to move geographically. Let us consider the definition of Dey et al. [ADB+99]: *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

This definition simplifies the task of enumerating the context parameters of a given application scenario for application developers. If a piece of information can be used to characterize the situation of a participant in an interaction, then that information is a part of the context. Although, this definition is missing a generalization of the notion of user that we will call *M*. Where *M* can be a user, a machine or device (including smart sensors, laptops, etc.) or any entity including a service or software component. Moreover, mobility of entities is also missing from the definition above. Physical location is a parameter that can characterize the context of an entity. When the entity is moving, for example a user travelling in a car or in a bus, this parameter is not static. Moreover the motion itself is also a parameter of the context, and it can be defined by its origin, destination and speed (and in some cases its orientation). This leads us to a new definition of context that we propose: *"Context is any information that can be used to characterize the situation of an entity M. M can be a person, a machine or any object including a service, a software component, or data, that is considered relevant to the interaction between any entities. Any entity M can be in motion within space and time, and can be transformed to become a new entity."* The transformation of entities means that for example an architecture of an application can change to be adapted to a new set of context parameters, in a manner that the application is not the same after change.

Data	Category	Examples
Identity	user	
Spacial	physical	location, orientation, speed
Temporal	physical	date, time of day, season
Environmental	physical	temperature, light, noise
Social	user	people nearby, activity, calendar
Resources	computing	CPU, RAM, battery
Physiological	user	blood pressure, heart rate
Resources	network	bandwidth, throughput

Table 2.1: Examples of Contextual Data

2.7.2 Taxonomy of the Contextual Data

Taxonomies of context provide a classification of contextual data. They may be useful for application designers in order to decide the most relevant types of contextual data that will be considered in their applications. General taxonomies divide context into two levels. The first level is composed of a set of four primary parameters: location, identity, time, and activity. These parameters characterize the situation of a particular entity. The secondary level of context parameters has a common characteristic: they can be indexed by the first level parameters. For instance, a user's music preferences is a piece of secondary context because it can be obtained by using the user's identity as an index into an information space like a musical database.

Schilit [SAW94] divides primary context into three main categories: computing context, user context and physical context. Some examples of contextual data of each category can be found in Table 2.1. In the rest of this work, we will use a similar classification: *external context* (i.e. groups, user, and physical context) and *resources context* (equivalent to computing and network context). External context contains information about the users, their location, time, the collaborative activity itself, etc. It is application-dependant since each application will be interested in different aspects of these external parameters. This context is captured with sensors that detect physical information and translate it into exploitable data. Resources context describes the processing and communication resources available on devices and communication links, e.g., memory, CPU load, battery level, bandwidth between two nodes, etc. The level of each resource can be directly captured on each device through the use of internal

monitors.

2.7.3 Context-Awareness

The main motivation of Context-Awareness is making computers more aware of the physical and social worlds we live in and of the available resources and breaking computers out of the box. This is based on several assumptions:

- explicit input/output is rather slow, intrusive, and requiring user attention;
- the input/output loop between the box and the user is sequential.

The main objectives of Context-Awareness are to move away from the black box model into context-sensitivity where human is out-of-the-loop (as much as possible) and to reduce explicit interaction as much as possible. The main principle is to let computer systems sense automatically, remember history of interaction, adapt to changing situations, and reduce explicit interaction. One needs to draw a boundary around the system under consideration and to define *explicit* and *implicit* interactions. The concept of Context-Awareness characterizes the capacity of a system to be adapted to the changes of the context. According to Dey and Abowd, a system is sensitive to the context if it uses the context to provide relevant information and services for the user, where the relevance depends on the task required by the user [ADB+99]. Context-aware applications are classified into three categories according to whether a context change implies a simple presentation of the changes, it implies the execution of adapted services or it implies specific storage of adapted data [Cha07].

2.7.4 Challenges

As seen in subsection 2.7.2 and subsection 2.7.3, Context and Context-Awareness research fields may enable the transition towards real implementations of communicating systems. However, there are still many issues that need to be solved. We cite here some of them:

- *Existence of general frameworks*: instead of using ad-hoc, application-specific context models and implementations, it would be desirable to have common frameworks that can be used by any application. However, as pointed by Edwards [Edw05], building such a framework is a very difficult task, because the

use and meaning of context is evolving, fluid and ambiguous and it is very difficult to predict all of the facets of context that will be meaningful to people and applications.

- *Reasoning about context*: it is necessary to have high-level abstract information about context that aggregates raw context data, so reasoning is necessary in order to correctly interpret this data. For example, if the system wants to know if two persons are at the same place, should it consider their GPS coordinates, although they are in the same city, in the same building, in the same room?

- *Acquisition of contextual data*: some context parameters are easy to acquire, e.g., position or temperature. But how to acquire information about physiological or social context?

- *Scalability* of context acquisition and reasoning infrastructures, either in terms of the volume of processed data and/or in terms of the number of supported client applications.

2.8 Adaptation in Collaborative Communicating Systems

In this section, some of the challenges in context adaptation for collaborative communicating systems are highlighted. Adaptation is the operation of making run-time changes to a software component or system in order to optimize its functionalities and performance in a certain execution environment. About adaptability, which is defined as the capability of a system to perform adaptation actions, Satyanarayanan says [Sat01]: *"adaptability is necessary when there is a significant disparity between a resource's supply and demand."*

2.8.1 Classification

The adaptation solutions suggested in the literature can be classified as follows.

2.8.1.1 Design vs Runtime

Two different adaptability views may be distinguished: the design time adaptability [DK04, FH00, EBP01] and the runtime adaptability [CK00, BLNS06]. For the first view, we can find design support tools which handle the application development cycle and optimizes the resources for example. On the other hand, the runtime adaptability [FDBC00] presents several adaptation techniques among which use proxy services, change model of interaction and reorganize application structure.

2.8.1.2 Local vs Distributed

Adaptation may have a local or a distributed scope. Adaptive components can be deployed on a single machine or distributed on several machines. In the first case, the adaptation is local and only local changes are performed. In the second case, it is distributed and synchronization problems between peer adaptive entities have to be managed [Bri01].

2.8.1.3 Behavioral vs Architectural

The adaptation solutions suggested in the literature distinguish behavioral and architectural aspects. The adaptation is behavioral (or algorithmic) when the behavior of the adaptive service can be modified, without modifying its structure. Standard protocols such as TCP and specific protocols such as those presented in [WHZ+01, OBAA04] provide behavior-based adaptation mechanisms. Behavioral adaptation is easy to implement but limits the adaptability properties. The adaptation is architectural when the service composition can be modified [IEE00, GP95, EHP+96] dynamically. In self-adaptive applications, components are created and connected, or removed and disconnected during execution. The architectural changes respond to constraints related to the execution context involving, for example, variations of communication networks and processing resources. Moreover, they may also respond to requirement evolution in the supported activities involving, for example, mobility of users and cooperation structure modification.

We focus on *architectural adaptation*, as it is the most common way of achieving software adaptation. As it has been pointed above, in the case of architectural adaptation the *object of the adaptation* (what is adapted) is the system's software: the set of components present in the system and the way they are interconnected. Since communicating

systems are a special case of distributed systems, the components may be deployed on different *nodes* (devices) that are interconnected over a network. Therefore, the component deployment and interconnection schema becomes the main object of adaptation in such systems. In the particular case of collaborative systems, this deployment is needed for establishing multimedia sessions between the users in order to support their collaboration. Therefore, the main adaptation actions in a collaborative communicating systems involve component (re)deployments and/or flow (re)configurations. For example, if a new user enters a group and audio is needed for him to collaborate with the other members of the group, then an *audio component* must be deployed on his machine and an *audio flow* must be set up between this component and the components deployed on the devices of the other participants, so they can communicate. The *trigger of adaptation* is the evolution of the context in which the application is executed. Adaptation to the resources context must be performed in order to optimize the utilization of resources in a constrained environment. For example, the user's personal devices have limited computation power so applications have to reduce their CPU consumption in order to allow the parallel execution of other applications. Battery is a critical resource in a mobile environment, so the deployment of software architectures must be adapted to the level of available battery on each device (e.g., deploy the most battery-consuming components on devices having the higher battery level). Adaptation to external context is crucial in order to provide users with a satisfactory service. It represents the main difference of communicating systems with respect to classic desktop systems.

In the case of collaborative activities, it may be very useful in order to detect potential collaboration situations and then spontaneously propose a collaborative session interesting for users. Once the collaboration is initiated, external contextual information may be also useful in order to organize the collaboration inside the group e.g., it can help to automatically decide in which group a user should participate, assign roles to users, etc. This twofold context can be viewed as a mix of high-level requirements and low-level constraints. It may be difficult to conciliate these two aspects of adaptation because they may be more or less orthogonal or even contradictory. Layered software architectures may be able to take into account adaptation actions needed in different layers. However, as pointed by Satyanarayanan [Mah04], this conciliation of layering and adaptation is very difficult to achieve and remains an open question. Some modelling effort is required in this area in order to provide guidelines helping to structure software communicating systems. Therefore, one of the main challenges of

collaborative communicating systems is to correctly handle adaptation with respect to changes in several levels of context.

2.9 Survey of Solutions for Context Adaptation in Collaborative Communicating Systems

This section describes several major research activities about context adaptation for communicating systems. The majority of these solutions also deal with some aspects of collaboration. However, in our opinion, very few research activities treat specifically the problem of providing tools for building context-aware collaborative applications for communicating systems environments. The survey focuses on handling context adaptation and the role that collaboration plays in the system. Related work has been classified into three subsections in accordance with the facets highlighted in the description: model, architecture or platform.

2.9.1 Model

Becker and Giese [BG08] present an approach based on graph transformation techniques coupled with UML stereotypes in order to model self-adaptive systems. Adaptation, which is performed at run-time, is decomposed into three levels: goal management, change management and component control. However, in this approach, context and collaboration are not explicitly modelled. Edwards [Edw05] presents a system named Intermezzo that enables the construction of applications making use of rich, layered interpretations of context. It provides a context data store and an integrated notification service. This system is collaboration-oriented; contextual information is structured through *activities*, that represent the use of resources (e.g., documents) made by users of the system. The underlying context model is very rich and deals with problems such as *ambiguity*, *identity*, *evolution* and *equality* of contextual data, providing solutions for each one of them. However, the case of low-level resources context is not considered. This work represents a great effort in providing a context information service for collaborative communicating applications. Adaptation to context changes is not considered within the framework: it is up to applications using the framework to react to these changes in order to provide a better service.

Padovitz et al. [PLZ08] present a context model and reasoning approach developed with concepts from the state-space model, which describes context and situations as geometrical structures in a multidimensional space. A context algebra based on this model is also presented. This work shows how merging different points of view over context enhances the global context reasoning process. The authors provide a model (named Context Spaces) that unifies the context views of different entities into a single context representation. Based on this model, they present an approach to reason about situations under uncertainty. They expand this model by developing Context Spaces algebra, which comprises operators that allow merging the perspectives of different entities in the pervasive environment. A multiagent system is used to enable distributed reasoning about the context. The cooperative aspect on this work focuses on migration, modelling and reasoning, partitioning, and merging context descriptions between agents for attaining optimal reasoning and context awareness. Therefore, collaboration is a tool for performing these tasks, but as in the previous case, high-level human collaboration is not considered.

2.9.2 Architecture

Zhang et al. [ZLL07] propose an adaptive model infrastructure for pervasive computing environments. They propose three adaptive layers: adaptive collaboration, adaptive middleware and adaptive services layers. Adaptive collaboration layer provides a service cooperation platform in dynamic environments. Adaptive middleware layer is a self-reconfiguring layer that provides an optimized uniform high-level interface for implementation of distributed applications. The adaptive services layer provides an adaptive contents service and adaptive user interfaces. Using this model, the approach makes environmental changes invisible to collaboration among applications and users. From the architectural point of view, this work is similar to ours, as it defines three layers. However, collaboration in this work is limited to the collaboration between services to adapt contents to the user interface. This adaptation is performed according to user preferences and environment; it does not take into account user collaboration. The middleware is implemented on top of CORBA. It uses reflection techniques to benefit from the Object Repository Broker of CORBA to make this layer self adaptive.

2.9.3 Platform

Ejigu et al. [ESB07] propose a collaborative context-aware service platform named CoCA. This platform is data-independent and it may be used for context-aware application development in pervasive computing. It performs reasoning and decisions based on context data and domain-based policies using ontologies. Data are organized into a generic context management model. The platform introduces a neighborhood collaboration mechanism to facilitate peer collaboration between the pervasive devices in order to share their resources. The generic context management modelling deals with the way the context data are collected, organized, represented, stored, and presented. The collaborative context-aware service interprets and aggregates the level of context values and performs reasoning about the context. Depending on this reasoning, decisions are taken about the actions to be triggered. In this work, collaboration is used between devices in order to enhance the achievement context data acquisition. It is quite different of our work because the collaborative aspect is used as a means of acquiring context data and reasoning about it, but human, user-to-user collaboration is not considered.

Lee et al. [LJP+06] present the project Celadon in order to establish an infrastructure enabling on-demand collaboration between heterogeneous mobile devices and environmental devices. Celadon project is a middleware architecture for ubiquitous device collaboration. Collaborative environments are organized into Celadon zones, which are public areas equipped with wireless access points for technologies such as Bluetooth or 802.11, and with environmental devices, such as displays, printers and servers. Using SOA, OSGi and Web Services, this infrastructure provides service discovery, description and binding functions, collaborative session management, and association management. Collaboration in this work is limited to sharing hardware or software resources. Sessions are defined as a group of devices, and they are used to manage the sharing of resources. The context considered in this work is the user context.

The Conami middleware [FNBS07] is a Collaboration-based Content Adaptation Middleware for dynamic pervasive computing environments. The authors consider the case of Mobile Ad hoc Networks (MANETs). The middleware allows devices in MANETs to collaborate with each other to perform content adaptation. Content adaptation is derived from the context of the user and the user environment (e.g., user's preferences, device's characteristics) and is done by composing available nearby ser-

vices. The middleware implements a content adaptation tree construction algorithm to consider the dynamicity of services. In this work, collaboration is considered as a technique to adapt the content according to the devices' capabilities. The considered contents include user preferences and device characteristics. The context is used by an Adaptation Decision Engine that chooses the services which will perform content adaptation. This middleware only handles content adaptation; architectural adaptation and component deployment are not considered.

Perich et al. [PJYF05] present the design and implementation of a Collaborative Query Processing protocol that enables devices in pervasive computing environments to locate data sources, and obtain data that match their queries. The features of the protocol enable devices, regardless their limited computing, memory, and battery resources, to collaborate with other devices in order to obtain an answer for their queries. The presented approach deals mainly with low level context parameters. Information describing location, time, identity and the current user activity are also used. This allows a device to predict what information the user will need. Based on that prediction, a device adapts its strategies for querying its neighbor devices, caching of data, and collaborative processing. Collaboration in this work is limited to sharing information and resources to deal with the lack of devices' capacity; user collaboration is not considered.

2.10 Discussion

The majority of the presented research activities consider collaboration as a way to provide a better use of resources (hardware or software). This collaboration is performed among several components of the system. Therefore, high-level, end-user collaboration, and the tools that support it are not considered. In our work, collaboration is considered as a first-class activity that requires special support and that may benefit from the potentialities of communicating systems. The presented research activities tackle the adaptation to the context in different ways. In general, they focus either on the resources context (low-level) or the user context (high-level). In our work, both contexts, and their mutual influence, are considered. Each one is processed at a different level, and the transformations between levels merge requirements from the upper level with constraints from the lower level. The variety of the solutions in the literature underline the need to provide tools, models and techniques to build adaptation in collaborative

communicating systems. In this chapter, we synthesize the trends and outcomes of collaborative computing, with respect to the evolution of computing devices. From desktop-based group-aware applications, collaborative computing has evolved to be built over numerous hidden devices, fixed and/or nomadic, and highly heterogeneous in terms of computing and communicating capabilities. Design approaches have evolved to take into account these important changes. The complexity increase has been tackled by introducing context adaptation approaches, able to automatically integrate the presence of numerous, dynamic, highly heterogeneous, and nomadic devices. Group sessions have evolved by introducing higher-level modelling approaches, able to handle group intention, and to instantiate implicit sessions through high-level rules. Hence, in the near future, the daily environment will transparently react to contextual information and needs. New interactive collaborative activities will emerge from these rich communicating environments. However, several research directions have to be pursued. Handling context remains a central and complex problem. In our opinion, a unified framework has to be proposed to use more effectively this context information. This framework should lay on semantic technologies that are able to capture high-level intentions. All these propositions define important challenges to be explored in the coming years.

Chapter 3

Proposed Approach

In this chapter, our modelling approach for collaborative communicating architectures is presented. This approach separates different concerns into associated abstraction levels to enable a clear encapsulation of problems in different levels to tackle them independently. Low level details are abstracted for higher levels, and therefore complexity is reduced, thus achieving a clear and coherent design. Moreover, this separation in levels favours the realization of a multi-level adaptation, that takes into account both high level requirements (related to human activities) and low level constraints (related to real implementations). Adaptation actions are more effective if they take into account data from several levels. Generic procedures for refinement and selection (that enable transitions between levels) are presented in this chapter. Refinement ensures that lower level models actually implement associated higher level models. Selection allows choosing a model among a set of candidates at a given level. The chosen model is optimal with regard to the current context and to a given policy.

3.1 Generic Modelling Approach

In the following, we present the model that we use in our approach. We provide generic level models and generic procedures for refinement and selection. Then, we mention the implementation in detail using OWL and graphs for models and rule oriented techniques, such as SWRL and graph grammar productions, for refinement and selection.

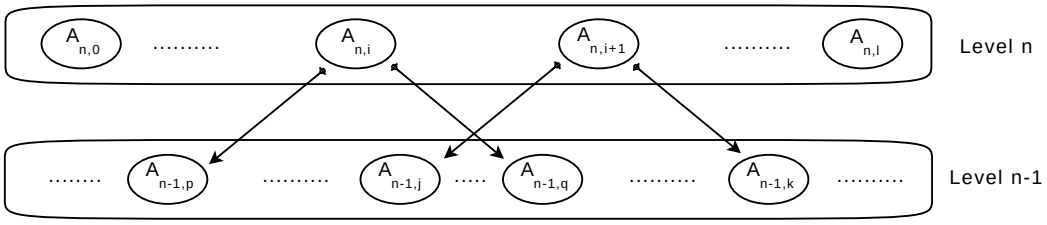


Figure 3.1: Multi-level Architecture Modelling

3.1.1 Multi-level Architecture Modelling

In our approach, models represent architectural configurations, i.e. sets of linked software entities. An architectural configuration is denoted $A_{n,i}$, where n is the considered abstraction level and i is the sequence order (i.e. an architecture $A_{n,i}$ evolves to $A_{n,i+1}$ when it is reconfigured). For a given architectural configuration $A_{n,i}$ at level n, multiple architectural configurations $(A_{n-1,1}, \ldots, A_{n-1,p})$ may be implemented at level $n-1$ (see Figure 6.1).

Adapting the architecture to constraint changes at level $n-1$ by switching among these multiple architectural configurations maintains the n-level architectural configuration unchanged. Moreover, when adaptation requires changes at level n, this may need no changes at level $n-1$. This happens if the initial and the new architectural configurations of level n (e.g. $A_{n,i}$ and $A_{n,i+1}$ in the Figure 6.1) share common implementations (e.g. $A_{n-1,j}...A_{n-1,q}$) at level $n-1$.

This adaptation technique requires two different actions: *refinement* and *selection*. Refinement determines the set $\mathbb{A}_{n-1} = \{A_{n-1,1}, \ldots, A_{n-1,p}\}$ of $(n-1)$-level architectural configurations that implement a given n-level architectural configuration $A_{n,i}$. Selection chooses the retained appropriate configuration among all possible architectural configurations at a given level. These two actions are explained in the following subsection.

3.1.2 Generic Refinement and Selection Procedures

A refinement associates a high level architectural configuration with a set of lower level architectural configurations. A generic refinement procedure called `Refine()` (see Table 3.1) is considered. For a given architectural configuration $A_{n,i}$ at level n, the procedure computes the set $\mathbb{A}^i_{n-1}$. This set represents all possible architectural config-

urations at level $n-1$ that implement $A_{n,i}$.

```
1 Refine()
2 {
3 Let 𝔸_n be the set of configurations at level n
4 Let 𝔸_{n-1} be the set of configurations at level n-1.
6 Let A_{n,i} ∈ 𝔸_n, i ∈ ℕ be a given configuration
7 Compute 𝔸^i_{n-1} = {A_{n-1,j} ∈ 𝔸_{n-1}
        such that:  A_{n-1,j} implements A_{n,i}, j ∈ ℕ}
8 }
```

Table 3.1: Generic Refinement Procedure

Given a set of possible architectures, it is necessary to choose an architecture to be effectively deployed. We present here a procedure, `Select()` (see Table 3.2), that allows choosing an architecture depending on several parameters. This procedure uses the resources context (e.g. variations of communication networks and processing resources) to eliminate the architectural configurations that can not be deployed within the current resources levels. Among the set of selected architectures, the best configuration with respect to architectural characteristics (e.g. number of components) is selected. The choice of an architecture must take into account the resources context at first. The `Context_Adaptation()` function (see Table 3.2, line 5) is a generic function that depends on the resources context. For example, it can express the availability level of a given resource (bandwidth, memory, energy, etc.). This function is used for two purposes: first, it allows discarding architectures that cannot be deployed within the current resources context. Second, it allows selecting the architectures best adapted to that context. This function assigns a value to a given architecture depending on its degree of adaptation to the current context. If the architecture is not compatible with the current context, its value will be -1. Otherwise, it will receive a positive value. Best suited architectures will receive higher values. When several architectures have the same value of `Context_Adaptation()`, a policy (indicated by the parameter `Policy`) is used by the `Select()` procedure in order to retain the optimal configuration. If the chosen policy is `Weight`, the selection is based on minimizing the function `Dispersion()` (see Table 3.2, line 8). This generic function corresponds to the cost or the efficiency/performance of an architecture. For instance, it may be defined as the number of software components deployed per node. If the chosen policy is `Distance`, the selection

```
1 Select(Policy)
2 {
3 Let A_{n,p} ∈ 𝔸_n , p ∈ ℕ
4 Let C denote the context attributes
5 Select S_1 = {A_{n-1,k} ∈ 𝔸^p_{n-1}, k ∈ ℕ such that:
      Context_Adaptation(A_{n-1,k}, C) ≥
        Context_Adaptation(X, C), ∀X ∈ 𝔸^p_{n-1}}
6 if card(S_1) ≠ 1
7    if Policy = Dispersion
8       Select S_2 = {A_{n-1,k} ∈ S_1 , k ∈ ℕ such that:
        Dispersion(A_{n-1,k}) ≥ Dispersion(X), ∀X ∈ S_1}
9    if Policy = Distance
10      Let A_{n,p} and A_{n,q} ∈ 𝔸_n , p, q ∈ ℕ
11      Let A_{n-1,p} the current mapping
         at level n − 1 of A_{n,p}
12      Select S_2 = {A_{n-1,k} ∈ 𝔸^q_{n-1}, k ∈ ℕ such that:
13      Relative_Cost(A_{n-1,p}, A_{n-1,k}) ≤
            Relative_Cost(A_{n-1,p}, X), ∀X ∈ S_1}
14   if card(S_2) ≠ 1
15      Select any configuration from S_2
16 }
```

Table 3.2: Generic Selection Procedure.

minimizes the distance between two architectural configurations at level $n-1$, both implementing the corresponding n-level architectural configuration. This is performed using the function `Relative_Cost()` (Table 3.2, line 13). `Select()` can be extended with other possible policies.

3.2 Multi-layer Modelling of Collaborative Communicating Architectures

This section presents how we apply the multi-layer modelling approach presented above to collaborative communicating systems in order to build a comprehensive generic framework for such systems. Therefore, relevant abstraction layers have to be identified. Adaptation at the highest layers should be guided by the evolution of activity requirements. Adaptation at the lowest layers should be driven by execution context

constraint changes. The retained layers are represented in Figure 3.2. These layers are numbered 3, 2 and 1 respectively and are presented detailed in the following paragraphs. Context capture and representation is discussed after that.

3.2.1 Design Time Side

The application layer represents applications needing collaboration inside groups of users and/or devices (which are generically called *entities*) in communicating environments. It contains software elements implementing the application's business, as well as user interfaces, security modules, etc. Among these elements, (at least) those that are relevant with respect to collaboration are represented in the architectural model corresponding to this abstraction layer, $A_{3,i}$. Hence, this model is a business view of the collaborating entities and the business relations between them. As this model is highly application-dependant, it must be built by the designers of each application and instantiated at runtime by the application itself. Application designers also have to implement the refinement procedure allowing to obtain the set $\mathbb{A}_2^i$ of collaboration layer models that implement a given $A_{3,i}$ model. Only collaboration-related elements of the application layer model will be taken into account in the refinement process[1]. The collaboration layer provides a session level abstraction. It describes how the members of a group are organized within sessions where they can send and receive data flows. The main issue addressed by this layer is the determination of a high-level collaboration schema that responds to the application's collaboration needs. Hence, it is able to manage collaborative sessions and to determine the elements needed to implement such sessions.

The architectural model produced by this layer, $A_{2,i}$, consists of a graph containing the following elements: *nodes*, *components* and *data flows* (which are organized within one or more *sessions*). This model is inspired by classic graph-based session description formalisms such as dynamic coordination diagrams [BDVT04].

3.2.2 Run Time Side

The middleware layer provides a communication model that masks low-level details (like TCP sockets, UDP datagrams and IP addresses) in order to simplify the repre-

[1]Nevertheless, the application layer model can contain other business elements (non collaboration-related) and thus be used in order to represent the whole application.

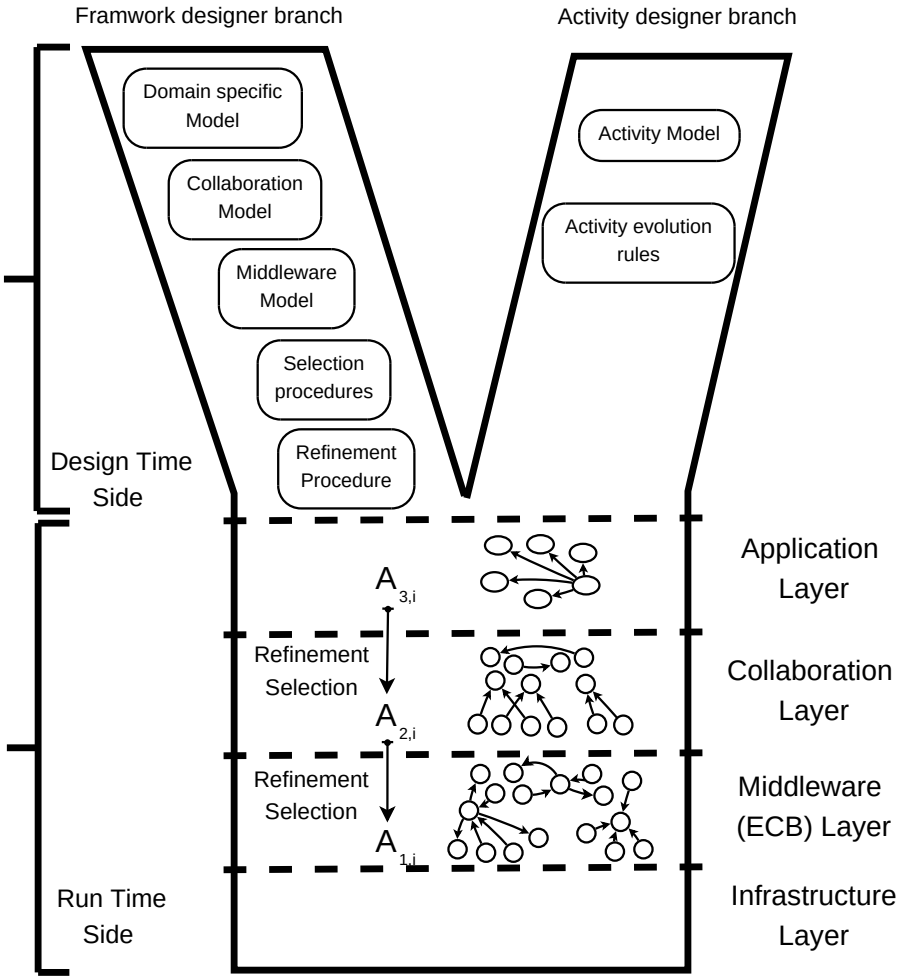


Figure 3.2: Layered Architecture Modelling

sentation of communication channels. This layer abstracts distributed systems, so they are transparent for upper layers. For instance, this model may be based on abstractions like Event Based Communications, Peer to Peer, Remote Procedure Calls or Remote ethod Invocation. The architectural model produced by this layer, $A_{1,i}$ represents a detailed deployment descriptor containing the elements needed to implement the sessions defined by the collaboration layer. For instance, if this layer is implemented with an Event Based Communication technology, the $A_{1,i}$ model contains the event producers, event consumers and channel managers to be deployed on each node, as well as the pull/push links between these elements. If a Peer-to-Peer implementation is considered, the $A_{1,i}$ model contains peers, super-peers and pipes to link peers. The infrastruc-

ture layer contains hardware devices and software components (e.g., OS, drivers) which are needed to run applications on the user's device(s). They also enable such devices to communicate with other devices existing in the user's environment. In our work, we assume that the services and components of this layer are available and correctly configured, and hence this layer is not handled by adaptation models.

3.2.3 Context Capture and Representation

In our work, we target the adaptation of cooperative applications due to different context changes. These changes may be related to resource constraints (like connectivity, energy level, available memory, etc.) or the evolution of the collaborative activity and the environment where participants can join and leave, change their roles, active a "do not disturb" mode in their devices, etc. We respectively call these two sets of parameters *resources context* and *external context*. Context parameters are captured by external modules. Relevant events, which are produced in the external context are taken into account by the application layer (i.e. they are detected and *translated* into modifications on the $A_{3,i}$ model, thus producing a new model $A_{3,i+1}$). Relevant events, that are produced, related to run-time resource changes are taken into account by the middleware layer. If it is possible, changes are handled by reconfiguring the middleware architecture model $A_{1,i+1}$ (i.e. by selecting a different model among the possible refinements of the collaboration model). If this is not possible, a message is sent to the upper layer in order to inform it that the current architecture can not be implemented.

3.3 A Collaborative Framework for Communicating Systems Reconfiguration

The proposed multi-layer modelling approach for collaborative communicating systems remains generic with respect to implementation. Indeed, it neither states the formalism of the presented models nor the means through which the refinement and selection procedures are implemented. As we have separated the approach and its implementation, this generic approach may be implemented in different ways by different designers. In this subsection, we propose an implementation that can be used by application designers as a collaboration framework for communicating systems. Figure 6.2 illustrates our

choices. We present the models used in each layer and then the refinement and selection procedures that enable the transitions between layers. We distinguish three branches the "Framework designer branch" the "Activity designer branch" that corresponds to the Design Time side of our approach and the "Run Time branch" corresponds to Run Time side of our approach.

3.3.1 Design Time Side

For description of the application layer architecture models ($A_{3,i}$), we have chosen an ontology based model based representation which constitutes a standard knowledge representation technique, allowing reasoning and inference. Moreover, ontologies facilitate knowledge reuse, and sharing through formal and real world semantics. Ontologies are high-level representations of business concepts and relations. Such representations are close to developers' way of thinking; therefore they are well suited to represent application layer models. We have describe these models in OWL [SWM04], the Semantic Web standard for metadata and ontologies.

In general, ontologies are divided in two levels: a generic ontology and a specific ontology. The former is a domain-wide ontology, but is independent of applications. The latter ontology extends the generic one with terms specific to an application-specific category. We have followed the same pattern in our implementation: we distinguish a generic collaboration ontology (that describes sessions, users, roles, data flows, nodes, etc.) and an application-specific ontology that extends the collaboration ontology with business-specific concepts and relations.

The generic collaboration ontology is common to all applications, and therefore it is provided within the framework. This ontology[2] is represented in Figure 3.4. The main concept in this ontology is `Session`. A session contains one or more `Flows`, which have a source `Node` and a destination `Node`. `Nodes` are hosted on `Devices`. Each `Node` has one or more associated `Roles`. `Flows` are processed by `Tools`, which are composed of several `Components` (e.g. `SenderComponents` and `ReceiverComponents`). Related `Flows`, `Tools` and `Components` share the same `DataType` (e.g. `Audio`, `Text` or `Video`). Further explanations about this ontology and the associated choices can be found in the work of Sancho [STV08]. Application designers can extend this generic ontology with specific ontologies describing the business logic of their applications. The collaboration

[2] http://homepages.laas.fr/gsancho/ontologies/sessions.owl

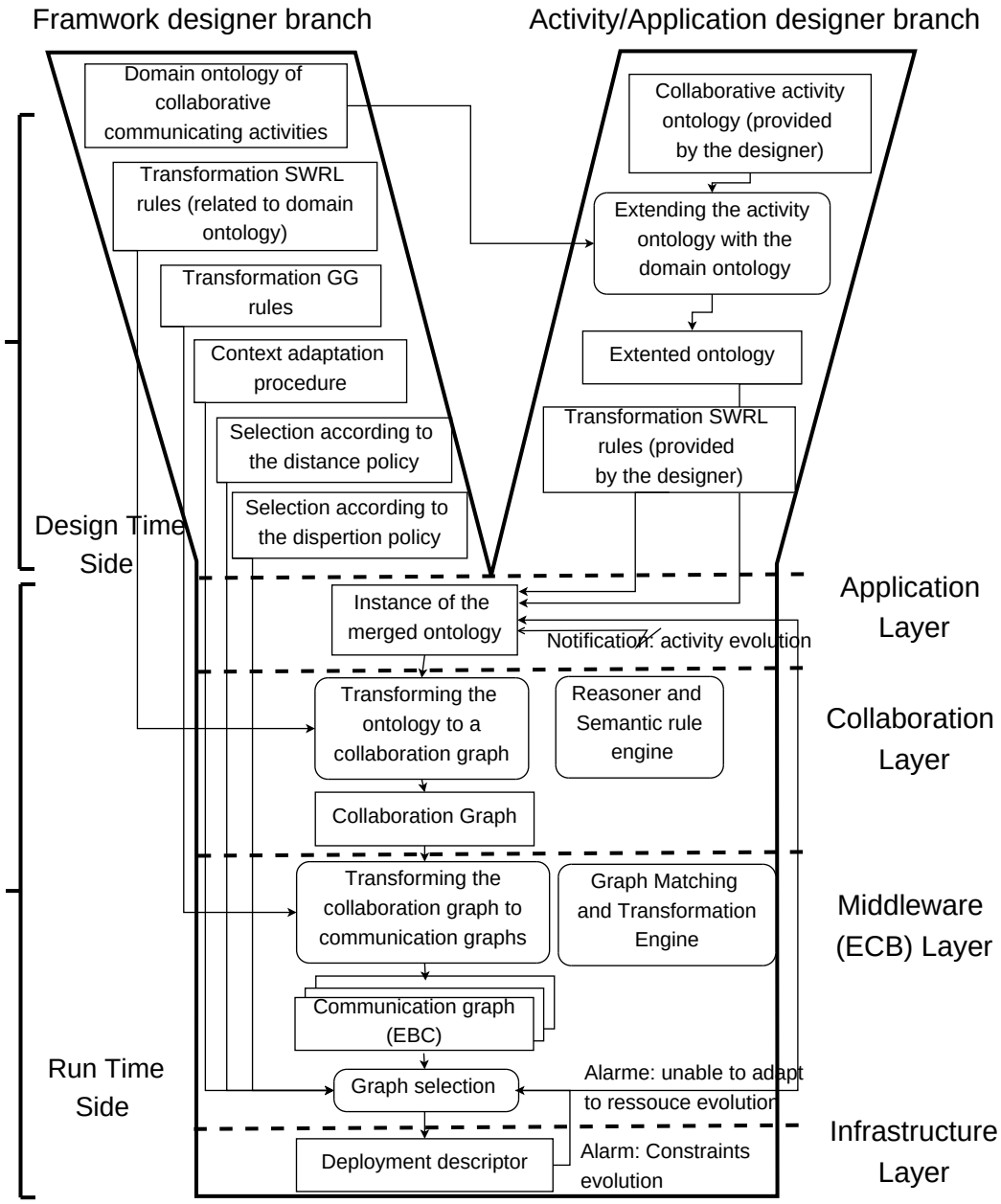


Figure 3.3: Model-Oriented Reconfiguration Framework.

layer model ($A_{2,i}$) is a graph, inspired by dynamic collaboration diagrams [BDVT04], shows the detailed structure of one or more session. A *session* is a set of *data flows*. Each *data flow* goes from a *sender component* to a *receiver component* (components are

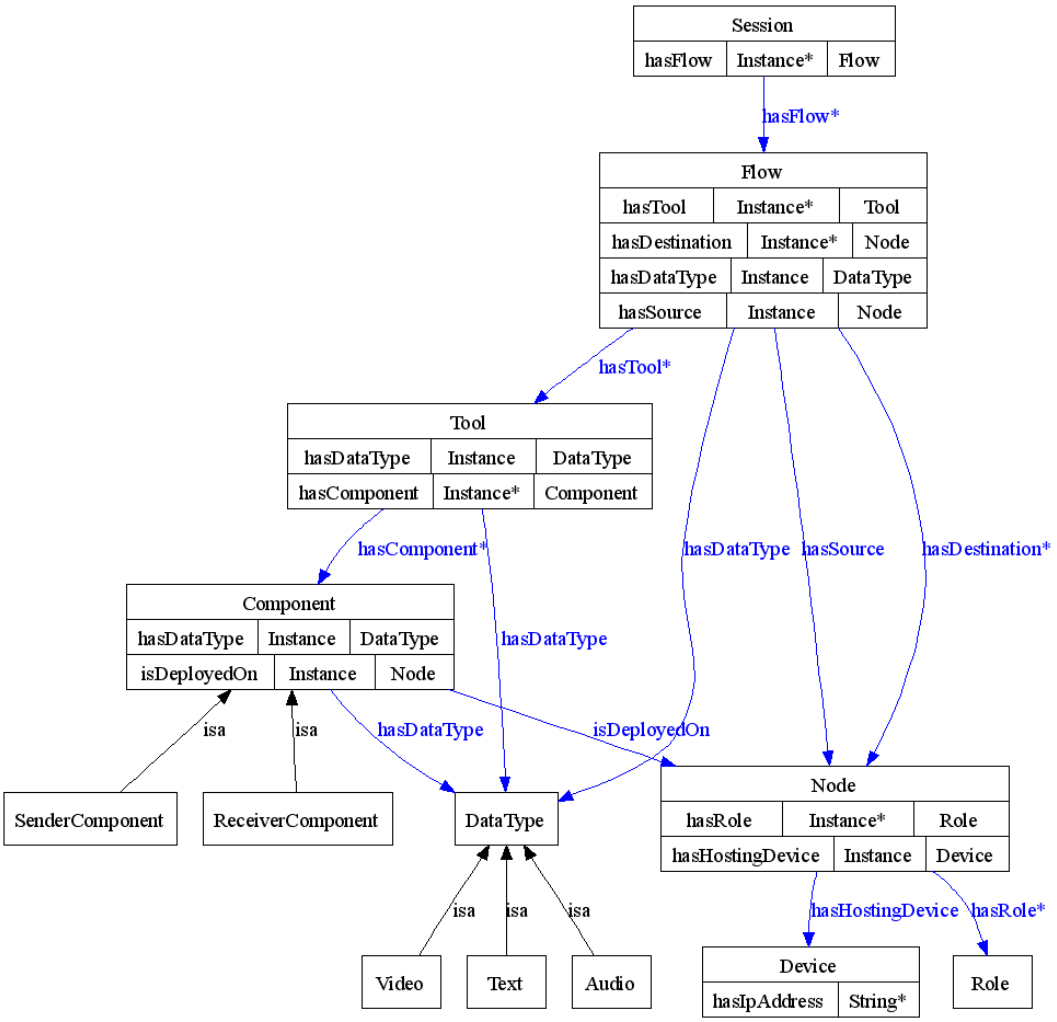


Figure 3.4: Generic Collaboration Ontology

deployed on *nodes*)[3]. Sender and receiver components may have text, audio or video as types. Flows are labelled with data types (audio, text and video) and the session to which they belong. This graph is expressed in the GraphML language (an XML dialect for representing graphs [BEH+01]).

[3]As the reader may have noticed, these same elements have already been presented in the collaboration ontology. This redundancy is exploited in order to refine models from application to collaboration layers, as it will be explained later.

3.3.2 Run Time Side

For the middleware layer ($A_{1,i}$), we consider the Event Based Communication (EBC) paradigm [MC02]. The EBC model represents a well established paradigm for interconnecting loosely coupled components and it provides one-to-many or many-to-many communication patterns. EBC entities are represented in the middleware layer model. This model is a detailed graph containing a set of *event producers* (EP), *event consumers* (EC), and *channel managers* (CM) connected with *push* and *pull* links. Multiple producers and consumers may be associated through the same CM. Since this model is also a graph, it is also expressed in the GraphML language.

3.4 Graph-Based Models for Architectural Transformation

We introduce here both graph grammar foundations and the graph grammar based approach that we use for the collaboration layer and the middleware layer models. Graph grammars constitute an expressive formalism dynamic structure description. Moreover, theoretical work on this field provides formal means to specify and check structural constraints and properties [Roz97, EK91]. Inspired from Chomsky's generative grammars [Cho56], graph grammars are defined, in general, as a classical system $< AX; NT; T; P >$, where AX is the axiom, NT is the set of the non-terminal vertices, T is the set of terminal vertices, and P is the set of transformation rules, also called grammar productions. An instance belonging to the graph grammar is a graph containing only terminal vertices and is obtained starting from axiom AX by applying a sequence of productions in P.

There are different approaches for the definition of a graph grammar production structure and the mechanisms used for the specification of its execution.

3.4.1 Basic approach for Graph Transformation

The basic approach to transform a graph G into a graph G' is to replace a sub-graph m of G by a graph d. G' is the graph resulting from this operation. G is called the host graph, m is called mother graph and d is called daughter graph. In this approach, a grammar production is described in the basic model by a pair of graphs $< L; R >$. This

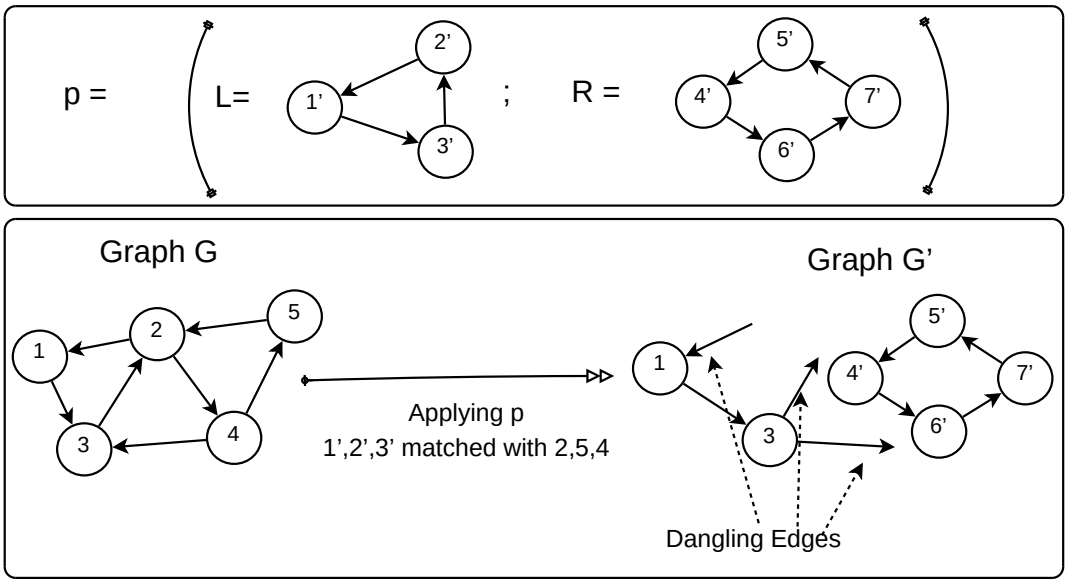


Figure 3.5: Dangling Edges Problem

rule is applicable to a graph G if there is an occurrence of L in G. As a consequence, its application leads to removing the occurrence of L from the graph G and substitute it by a copy (isomorphous) of R. This basic definition introduces the problem of dangling edges as shown in Figure 3.5 where the grammar production P is applied on graph G.

In this example, different matching of pattern L are possible in graph G (for instance, nodes 2, 5, 4 respectively match nodes 1', 2', 3'). In this case, the removal of the pattern corresponding to L will lead to the situation where edges that connect nodes 2 to 1, 3 to 2 and 4 to 3 will become dangling edges. Moreover, in this basic approach, it is not possible to specify the gluing of the introduced pattern (i.e. gluing nodes 4', 5', 6', 7'). To address the dangling edges problem, different approaches were introduced with different choices concerning productions specification and dangling edge management.

3.4.2 The Double PushOut Approach

The Double PushOut (DPO) [Ehr87] considers a richer structure for grammar productions. These productions are specified by a 3-uplet $< L; K; R >$ where L and R keep the same significance as in the basic structure. K is a sub-pattern of L specifying a part to be maintained after applying the rule. The application of this production requires an additional condition called the dangling condition. This condition states that the

production is applicable only if its application will not lead to dangling edges. If the two conditions (i.e. existence of an occurrence of L and absence of dangling edges) hold, the application of the production implies the removal of the occurrence $Del = (L \setminus K)$ and the insertion of a copy of $Add = (R \setminus K)$. The DPO approach introduces a more sophisticated structure for grammar productions. Considering pattern K allows the specification of graph transformations by considering nodes in the L pattern that will not be removed after applying the related productions. However, DPO manages the dangling issue in a very basic way. To overcome this limitation, we will consider a richer handling for dangling edges and daughter graph gluing.

3.4.3 The Node Label Controlled Mechanism

The Node Label Controlled (NLC) mechanism [Roz97] is based on the specification of the so-called connection instructions to allow the gluing of the daughter graph to the neighbours of the mother graph nodes. These instructions consider node type/labels to carry out this gluing. A production NLC is of the form $< L; R >$ where L and R are graphs containing labelled terminal and non-terminal nodes. The application of such a production implies, the basic replacement of the mother graph L by the daughter graph R. The connection instructions are of the form (μ, δ) where μ and δ are two labels that can be assigned to terminal or non-terminal nodes. Their execution implies the introduction of an edge between each node of the daughter graph labelled with μ and each neighbour of the mother graph nodes that are labelled with δ [4]. A grammar NLC is thus described by a quadruplet $< AX; NT; T; P; C >$ where AX, NT, T, P respectively represent the traditional axiom, the set of non-terminals, the set of terminals and the set of grammar productions. C constitutes the set of grammar connection instructions. These instructions are common to all the productions and after each application of one of the productions belonging to P all the applicable instructions in C will be executed.

A simple example of the DPO transformation approach is given in Figure 3.6. We notice that the host graph of this example G is different from the one given for the basic approach (graph G of Figure 3.5). The difference lies in the fact that G does not contain any more the edge connecting nodes 4 and 3 in G. This is due to the fact that if

[4]The existence of neighbours of the graph mother nodes that are labelled by δ is not a condition for the applicability of the production. If such nodes do not exist in the host graph, the connection instruction is simply ignored.

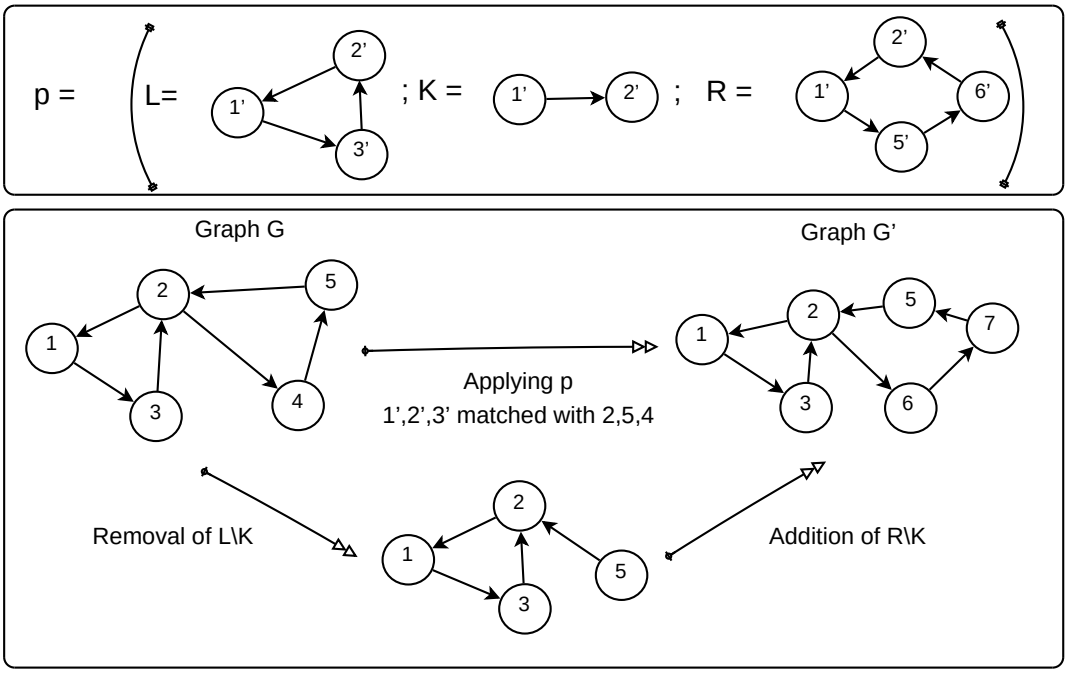


Figure 3.6: DPO Approach Example

we maintain this edge in G, P would not be applicable any more because its application would violate the dangling condition.

3.4.4 The Neighbourhood Controlled Embedding Mechanism and its Extensions

The main weakness of the NLC approach is that nodes are distinguished only by their labels. The Neighbourhood Controlled Embedding (NCE) mechanism [Roz97] addresses this problem by making it possible to describe connection instructions referring directly to the daughter graph nodes instead of referring to their labels. The instructions are of the form (n, δ) where δ is a label of a terminal or non-terminal node, and where n identifies a node belonging to the daughter graph. The execution of this instruction implies the introduction of an edge between node n and all nodes that are neighbours of the mother graph labelled by δ. An NCE graph grammar is defined by the quadruplet $< AX; NT; T; P >$ where AX, NT and T keep the same significance as in NLC grammars. P Specifies the set of grammar productions that are of the form $(L; R; C)$ where (L, R) is NLC production and where C is a set of NCE instructions. Compared to the

NLC approach, the instructions are not applied to the whole grammar productions set, but are related to a specific production. The extension of NCE for directed graphs is quite easy. Within the framework of this extension called directed NCE (dNCE), connection instructions are described by a triplet (n, δ, d) where $d \in \{in, out\}$ making it possible to take into account edge direction:

- an instruction (n, δ, in) implies the introduction of an edge from the node n to all n' which are in-neighbours of the mother graph and that are labelled by δ.

- an instruction (n, δ, out) implies the introduction of an edge from the node n to all n' that are out-neighbours of the mother graph and that are labelled by δ.

This approach preserves the direction of the edges. Another approach makes it possible modify edge direction. In this case, the instructions are specified by a quadruplet (n, δ, d, d') where n, δ and d keep the same significance as in the preceding approach d' indicates the direction of the edge to be introduced. Thus, the instruction (n, δ, in, out) (respectively (n, δ, out, in)) implies the introduction of an edge from node n and all nodes n' that are in-neighbours (respectively out-neighbours) of the mother graph and who are labelled δ. The direction of the edge is, this time, from n to n' (respectively from n' to n)[5]. To consider labelled edges, an additional extension of the NCE approach can be introduced. This approach called edge label NCE (eNCE); it takes into account the edges' labels and their updating. Connection instructions are of the form $(n, \delta, p/q)$ where n and δ indicate the same concepts as in the case of the classical NCE approach while p and q are edge labels. The execution of this type of instructions implies the introduction of an edge with the label q between n and all nodes labelled by δ and that are p-neighbours[6] of the mother graph. edNCE approach combines the two approaches eNLC and dNLC. The grammar productions are of the form $(X; D; C)$ such that C is a set of instructions of the form $(n, \delta, p/q, d, d')$. The execution of this instruction implies the introduction of an edge in the direction indicated by d' between the node n and all nodes n' that are p-neighbours and d-neighbours (i.e. in-neighbours if d=in and out-neighbours otherwise) of the mother graph.

[5]Instructions (n, δ, in, in) and (n, δ, out, out) are respectively equivalent to the instructions (n, δ, in) and $(n, delta, out)$ that preserve edge direction.

[6]p-neighbours of a node n are all nodes n' such that there exists an edge labelled by p which connects n and n'.

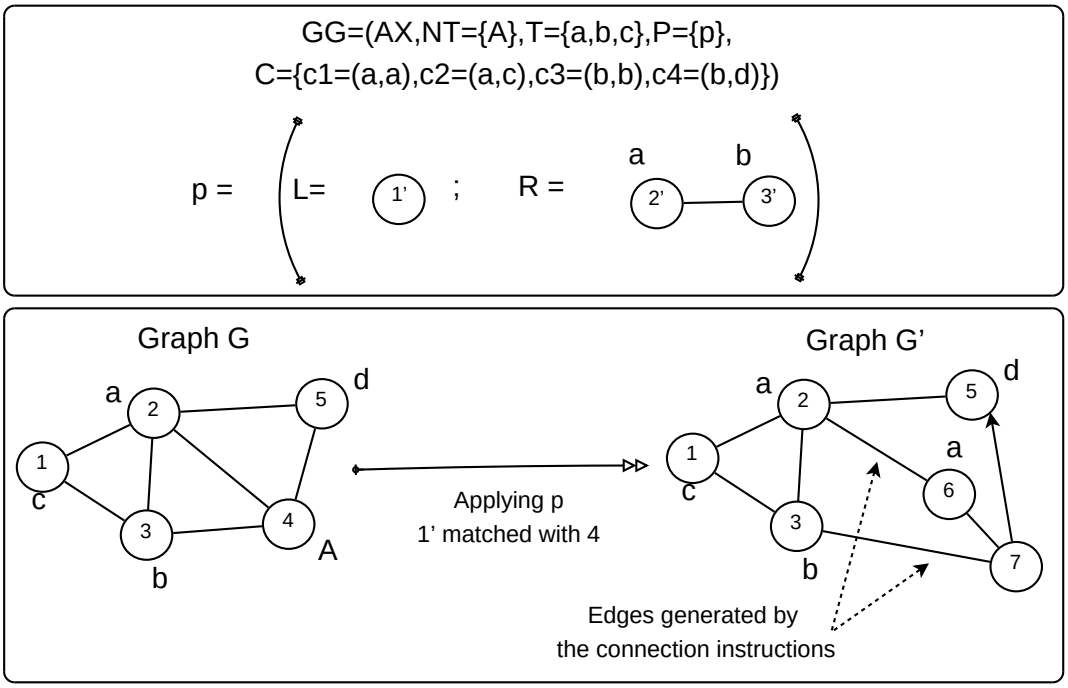


Figure 3.7: NLC Approach Example

Figure 3.7 gives an example of graph transformation using the NLC mechanism. Non-terminal node 4 having the label A is replaced by a copy of the graph R, i.e. nodes 6 and 7 and the edge that connects them. The application of the instruction c1 has as a consequence the introduction of an edge between nodes 6 and 2; c3 implies the introduction of an edge between nodes 7 and 3 while c4 introduces the edge between 7 and 5. Instruction c2 is ignored since none of the neighbours of node 4 is labelled by label c.

3.4.5 Our Approach: Combining DPO and edNCE

In our approach, a graph grammar is of the form $< AX; NT; T; P >$ where AX, NT and T keep the same significance as before. We use productions of type $(L; K; R; C)$ where $(L; K; R)$ corresponds to the structure of a Double PushOut (DPO) production and where C is a set of connection instructions. The instructions belonging to C are of the edNCE type. They are specified by a system (n, δ, d, d') where n corresponds to a vertex belonging to the daughter graph R, p and q are two edge labels, δ is a vertex label, and d and d' are elements of the set in, out. For example, a production defined by

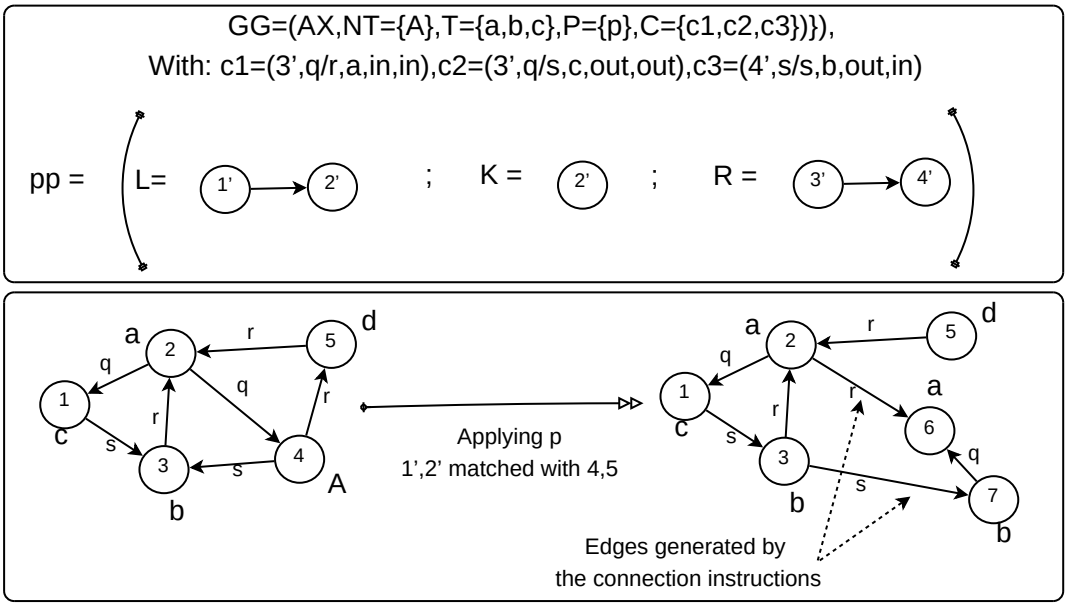


Figure 3.8: Combining DPO and edNCE

the system $(L; K; R; (n, \delta, d, d'))$ is applicable to a graph G if it contains an occurrence of the mother graph L. The application of this production involves transforming G by deleting the subgraph $(Del = L \setminus K)$ and adding the subgraph $(Add = R \setminus K)$ while the subgraph K remains unchanged. All dangling edges will be removed.

The execution of the connection instruction implies the introduction of an edge between the vertex n belonging to the daughter graph R and all vertices n' that are p-neighbours[7] of and d-neighbours[8]. This edge is introduced following the direction indicated by d' and labelled by q. An example is given in Figure 3.8.

Production $p3$ has the same structure and transformation logic as in the DPO approach: node 5 is not removed even if it is matched with node 2' belonging to the L pattern because it also belongs to the K pattern. The example also considers the edNCE connection instructions c1, c2 and c3 allowing the correct addition of edges connecting nodes 2 and 6, and connecting nodes 3 and 7.

Following the commonly used conventions for graphs describing architertures, we consider that vertices represent communicating entities (e.g. services, components)

[7]p-neighbours of a vertex n are all vertices n' such that there exists an edge labelled by p which connects n and n'.

[8]In-neighbours if $d = in$ and out-neighbours otherwise.

and edges correspond to their related interdependencies (e.g. communication links, composition dependencies).

3.5 Refinement and Selection Implementation

In this section, we present how the application layer architecture models are refined into collaboration layer architecture models, and how collaboration layer architecture models are refined into middleware layer models. The application layer models are ontologies represented in OWL and we use SWRL rules to implement the refinement rules. Collaboration layer models and middleware layer models are graphs and we use the graph grammar approach presented in subsection 3.4.5 to implement the refinement rules between the architectural models of two layers.

3.5.1 Application–Collaboration Refinement and Selection

As the application layer model is represented in OWL, we use SWRL rules [HPSB+04b] in order to implement its refinement to a collaboration architecture. Such rules constitute a natural and flexible way for expressing designers' choices for transformations (in comparison, e.g. to hard-coding such transformations). SWRL rules have the following form: $b_1, ..., b_n \rightarrow a$, where $b_1, ...b_n$ is the body of the rule and a is the head of the rule. The terms $a, b_1, ..., b_n$ are SWRL atoms, i.e. *concept* assertions, *relation* assertions or *built-ins*. The semantics of the rule are: whenever the interpretation of the body holds, then the interpretation of the head also holds. Some rules have been included along with the generic collaboration ontology. For example, let us consider the rule shown in Table 3.3. This rule states that, whenever an instance of `DataFlow` is found in the ontology, two components have to be instantiated[9]: a `ReceiverComponent` having the flow's destination node as its deployment node, and a `SenderComponent` having the source node as its deployment node. Similar rules are used for text and video flows, thus generating text and video sender and receiver components. The rules implementing the transition from the application-specific ontology to the generic collaboration ontology are application-dependant, and therefore they have to be specified by the application designers along with the application ontology. Thus, designers specify the application

[9]The SWRL *built-in* `CreateOWLThing()` creates new instances of existing concepts within a SWRL rule.

```
DataFlow(?af) ∧ hasSource(?af,?src) ∧
hasDestination(?af,?dst) ∧
swrlx:createOWLThing(?asc,?src) ∧
swrlx:createOWLThing(?arc,?dst)
→ AudioSenderComponent(?asc) ∧
isDeployedOn(?asc,?src) ∧
AudioReceiverComponent(?arc) ∧
isDeployedOn(?arc,?dst)
```

Table 3.3: A SWRL Rule for Establishing DataFlows

layer model and part of the refinement from application to collaboration layers. The processing[10] of the SWRL rules along with the application ontology produces a new ontology instance that describes the collaboration layer graph in OWL language. This graph is translated into GraphML in order to be shared with the middleware layer. The refined model produced by the rule processing is unique, i.e. each application layer model corresponds to a unique collaboration layer model. Therefore, the selection procedure at this layer is straightforward.

3.5.2 Collaboration–Middleware Refinement and Selection

As the collaboration layer and the middleware layer models are represented by graphs, graph grammar theories, presented in section 3.4, represent an appropriate formalism to handle the refinement process. We provide a graph grammar-based implementation[11] of the generic refinement procedure presented in Table 3.1. This implementation, called `Grefine()` (see Table 3.4), corresponds to the application of a set of graph grammar productions $p_1 \dots p_k$ that implement the refinement of an architectural configuration from level n to level $n-1$. We use a graph grammar, that addresses the refinement of a given collaboration level architecture to all possible EBC level architectures. The productions of this graph grammar consider collaboration components (e.g. `RevceiverComponent` denoted R and `SenderComponent` denoted S)) as non-terminal nodes and EBC entities (EPs, ECs and CMs) as terminal nodes. A session involving

[10] This processing is done with a rules engine such as Jess or a SWRL-enabled reasoning engine such as Pellet.

[11] This implementation is done with a Graph Matching Transformation Engine (GMTE), available at http://homepages.laas.fr/khalil/GMTE

1 G_Refine()
2 {
3 Let $\mathbb{A}_n$, $\mathbb{A}_{n-1}$ be the set of configurations at level n and level $n-1$.
4 Let $A_{n,i} \in \mathbb{A}_n, i \in \mathbb{N}$, be a given configuration
5 Compute $\mathbb{A}_{n-1,i} = \{A_{n-1,j} \in \mathbb{A}_{n-1}$ such that:
$\exists p_1 \dots p_k \in P : A_{n,i} \xrightarrow{p_l \dots p_k} A_{n-1,j}, j \in \mathbb{N}\}$
6 }

Table 3.4: The Graph Refinement Procedure

several senders and receivers is refined as a CM connected to several EPs and ECs. In order to refine a given collaboration architecture into a set of EBC architectures, the graph grammar $GG_{COLLAB \rightarrow EBC}$, detailed in the Table 3.5, is used[12]. In this graph grammar, non-terminal nodes are collaboration entities while terminals nodes are EBC entities. Therefore, the productions of this graph grammar refine `ReceiverComponent` and `SenderComponent` (R and S) into EPs, ECs and CMs. Similar grammar productions have been developed for text and video components. The production p_1 refines the pattern consisting of a *SenderComponent* (denoted as as) connected to a *ReceiverComponent* (denoted as ar) by the introduction of an event consumer, an event producer and a channel manager for a specific session (denoted here by x). Connection instructions $ic1$ and $ic2$ consider the push options. Other *ReceiverComponent*s (resp. *SenderComponent*s) linked to ar (resp. as) are connected to the created channel manager. The production p_2 refines the pattern consisting of a *SenderComponent* connected to a channel manager. The production p_3 refines the pattern consisting of a *ReceiverComponent* connected to a channel manager for a specific session (denoted here by x). The production p_4 guarantees that only one channel manager is kept for each session.

In order to choose the architecture to be effectively deployed, the selection procedure `Select()` (see Table 3.2) is used. This procedure allows choosing an architecture depending on several parameters. First, it uses the captured resources context to eliminate the architectural configurations that cannot be deployed within the current resources levels. Then, the best configuration with respect to architectural characteristics (e.g. number of components) is selected. In order to select the optimal architecture among those built by the refinement process, the generic selection procedure presented in

[12]This graph grammar is processed with a Graph Matching Transformation Engine (GMTE), available at http://homepages.laas.fr/khalil/GMTE.

$GG_{COLLAB \rightarrow EBC} = (AX, NT, T, P)$ with: $T = \{CM(cm, s, m), EC(ec, m), EP(ep, m)\}$, $NT = \{R(ar, m), S(as, m)\}$ and $P = \{p_1, \ldots, p_4\}$
$p_1 = ($ $L = \{R(ar, m1), S(as, m2), S \xrightarrow{data,s} R\}$; $K = \{\ \}$; $R = \{EC(ec1, m1), EP(ep1, m2), CM(cm1, s, m1),$ $CM \xrightarrow{push} EC, EP \xrightarrow{push} CM\}$; $C = \{$ $ic2 = (CM(cm1, s, m1), data, s/push, S, out/out),$ $ic1 = (CM(cm1, s, m1), data, s/push, R, in/in)\})$
$p_2 = ($ $L = \{S(as, m2), CM(cm1, x, m1), S \xrightarrow{data,s} CM\}$; $K = \{CM(cm1, s, m1)\}$; $R = \{EP(ep1, m2), CM(cm1, s, m1), EP \xrightarrow{push} CM\}$; $C = \{\ \})$
$p_3 = ($ $L = \{R(ar, m2), CM(cm1, s, m1), CM \xrightarrow{data,s} R\}$; $K = \{CM(cm1, s, m1)\}$; $R = \{EC(ec1, m2), CM(cm1, s, m1), CM \xrightarrow{push} EC\}$; $C = \{\ \})$
$p_4 = ($ $L = \{CM(cm1, s, m1), CM(cm2, s, m2)\}$; $K = \{\ \}$; $R = \{CM(cm1, s, m1)\}$; $C = \{\ \})$

Table 3.5: Refinement Graph Grammar $GG_{COLLAB \rightarrow EBC}$

Table 3.2 is used. Here, we present in detail our choices for the functions `Dispersion()`, `Relative_Cost()`, and `Context_Adaptation()`. The function `Dispersion()` is used to select architectures having fewer Channel Managers (CM) deployed on the same device. The goal is to efficiently balance resource consumption and to be more robust. This function is detailed in Table 3.6. It associates an architecture $A_{1,q}$ with the number of nodes containing at least one CM. This definition gives higher values to architectures having CMs dispersed in more nodes. The function `Relative_Cost()` (see Table 3.7) is

```
1 Dispersion()
2 {
3 Let A_{1,q} be an architecture at level 1
4 Let node_i^q be a deployment node of A_{1,q}
5 weight=0
6 For each node_i^q
7  if ∃CM deployed on node_i^q then weight=weight+1
8 return weight
9 }
```

Table 3.6: Dispersion Function

used to select the *closest* architecture to the currently deployed architecture. We choose as a criterion of selection the number of redeployments needed to switch from a given architecture to another. The function `Context_Adaptation()` associates a given archi-

```
1 Relative_Cost()
2 {
3 Let A_{1,q} and A_{1,k} be two architectures at level 1
4 Let node_i^q(component_j) be the deployment node
    of component_j ∈ A_{1,q}
5 rcost=0
6 For each component_j ∈ A_{1,q} ∪ A_{1,k}
7 if node_i^q(component_j) ≠ node_i^k(component_j) then
    rcost=rcost+1
8 return rcost 9 }
```

Table 3.7: Relative Cost Function

tecture to a value that reflects its degree of adaptation to the current resources context. Our criterion for this function is as follows: well adapted architectures are those which have fewer nodes in a *critical situation*. A node is in a critical situation when its level for a certain resource is close to the threshold defined for that resource. This function is detailed in Table 3.8. In this function, a set of resources $Resource_1 \ldots Resource_R$ (e.g. $Resource_1$=energy, $Resource_2$=CPU load and $Resource_3$=available RAM). L_r^i represents the available level of the resource $Resource_r$ for the node $node_i$. It is calculated as the resource's level before deployment (given by the resources context module) minus the amount of resource consumed by each deployed component; considering that it is expressed as a percentage. The value T_r is a threshold that indicates the criti-

```
1  Context_Adaptation()
2  {
3  Let A_{1,q} be an architectural configuration at level 1
4  Let N = card(A_{1,q})
5  Let Resource_1 ... Resource_R the set of considered resources
6  Let R = card({Resource_1 ... Resource_R})
7  Let node_i^q be deployment node i of A_{1,q}
8  Let L_r^i the level of the resource r for node_i^q
9  Let T_r be the threshold associated with the resource r
10 Let α_r^i  r ∈ [1..R] be weights associated with each resource
         r for node_i^q
11 Let β_r  r ∈ [1..R] be weights associated with each resource
         r for A_{1,q}
12 Let cadapt=0
13 for each i ∈ 1..N
14     for each r ∈ 1..R
15           P_r^i = α_r^i L_r^i − T_r
16           if P_r ≤ 0 then return -1
17     end for
18 end for
19 for each r ∈ 1..R
20     cadapt=cadapt+β_r min_i(P_r^i)
21 end for
22 return cadapt
23 }
```

Table 3.8: Context Aware Function

cal percentage of the resource r, for a given node, under which the deployment is not possible. The coefficients α_r^i represent the importance assigned to each level L_r^i with respect to the characteristics of $node_i$. For instance, the CPU load is more critical for a smartphone than for a laptop, because the smartphone needs CPU processing for answering calls, etc. Therefore, α_{CPU}^i is 1 for a laptop nodes and 0.5 for a smartphone (i.e. a smartphone will be considered as critical when its CPU level is lower than $2T_{CPU}$). P_r^i is calculated for every node as the difference between the level of the $Resource_r$ (pondered by the correspondent α_r^i) and the threshold T_r. If this difference is negative for a node, this means that the node is in a critical state with respect to $Resource_r$, and hence the considered architecture cannot be deployed. Therefore, -1

is returned and the considered architecture will not be selected. If no node was found to be in a critical situation, then the *degree of adaptation to the context* (`cadapt`) of the considered architecture is calculated, as shown in Table 3.8, line 20. First, for every resource, the minimum value of P_r^i found on any deployment node of the architecture is retained. Second, `cadapt` is calculated as an average of these minima (pondered by β_r coefficients). The β_r coefficients represent the global importance degree given to each resource ($\sum_1^R \beta_r = 1$). Resources with higher β_r are considered more important than other resources. These coefficients can be defined by the administrator or the business logic. This definition of the function `Context_Adaptation` leads to the selection of the architectures having the highest values of resources for their more critical nodes.

3.5.3 Deployment Service

The $A_{1,i}$ model produced by the middleware layer is the detailed deployment descriptor that implements the low-level elements of the required architecture: producers, consumers, channel managers, and links. In order to effectively deploy such elements into real devices, a *Deployment Service* is needed. This service takes a deployment descriptor $A_{1,i}$ as input and then it downloads, installs, and starts the required components on each device. From our point of view, the implementation of this deployment service may be based on the OSGi technology [OSG07]. Indeed, OSGi offers very promising functionalities such as dynamic code loading and execution. Within this approach, deployable components may be packaged as OSGi bundles that are easily handled by the proposed deployment service.

3.5.4 Refinement and Selection Illustration

This subsection provides a global view of the top-down refinement process from application level to middleware level that is initiated at the application startup. Similarly, the adaptation process triggered by context changes is explained. Once the application is started, the first step is the creation of the application level model $A_{3,0}$ (which is an instance of the application ontology) by the application. This model represents the state of the application itself and its external context. The second step is to start the refinement process to refine this model into a collaboration model $A_{2,0}$. This is done by the processing of the associated SWRL rules transformation. The produced

graph $A_{2,0}$ is used as an input to the `Grefine()` procedure, which returns a set $\mathbb{A}_1^0$ of middleware level models (graphs) that implement the given collaboration model. In order to retain a single model to be deployed, the `Select(Dispersion)` procedure is used. The retained model $A_{1,0}$ is hence the optimal model taking into account the current resources context first, and then the model dispersion, if necessary. Finally, $A_{1,0}$ is used by the deployment service as a deployment descriptor for the effective deployment of components. The system keeps the same configuration until the arrival of context events. Changes in the external context are translated into changes at the application level, thus producing a new instance $A_{3,1}$ of the application model. These high-level changes imply the triggering of the refinement process in order to obtain a new deployment descriptor. Therefore, external context changes are handled by an inter-level adaptation, because they trigger modifications at several levels. First, this model is refined into a new collaboration model $A_{2,1}$. If this model is equal to the previous $A_{2,0}$, no reconfiguration is needed at collaboration and middleware levels, i.e. the current low-level configurations implement the previous high-level configuration. Otherwise, the procedure `Grefine()` is used in order to refine the collaboration model $A_{2,1}$ into a set $\mathbb{A}_1^1$ of middleware level configurations. Then, the `Select(Distance)` is used to find $A_{1,1}$ which is the most adapted configuration for the current context. As previously explained, the use of the `Distance` policy implies that, if several choices are possible, the chosen configuration will be the *closest* to the previous configuration $A_{1,0}$. If $A_{1,1}$ and $A_{1,0}$ are equal, then no redeployment is needed. Otherwise, a new deployment is performed according to the new deployment descriptor $A_{1,1}$. Changes in the resources context are handled at the middleware level. Therefore, the response to resources context changes is an intra-level adaptation. This adaptation is performed by the `Select(Distance)` procedure. This procedure scans the set $\mathbb{A}_0^1$ (which was built in the initial refinement) in order to find a new model $A_{1,p}$ that is better adapted to the new resources context than the previous model $A_{1,0}$ to be used for redeployment. On the other hand, Application and collaboration models remain unchanged. For all the considered cases (initial refinement, external context adaptation, and resources context adaptation), the `Select()` procedure may find none of the candidate configurations can be deployed within the current resources context. This means that the high level requirements can not be implemented with the current resources, so the middleware level sends an alert to the application level. Therefore, the application level produces a new model which considers this limitation, and the refinement process is triggered in

order to obtain a new collaboration model followed by a new middleware model, which is used for redeployment.

3.6 Conclusion

In this chapter, we presented a multi-layer architecture modelling approach for collaborative communicating systems. Architectural models for application, collaboration and middleware layers have been detailed. Ontologies, SWRL rules, graphs, and graph's grammars have been used for implementing a rule-based refinement process. These rules handle both transforming a given architecture within the same layer, and architectural mappings between different layers. This implementation forms a framework for building collaborative communicating systems. Using such rule-based approach allows correct architectural reconfigurations to be characterized, and to be used either *offline* to help implementing the decision process, or *online* to handle architectural adaptations. We presented also, different approaches for the definition of graph grammar production structure, and the mechanisms used for the specification of their execution. In the next chapter, we will present the case studies that we used to illustrate and evaluate our proposed framework.

Chapter 4

Implementations, Case Studies and Performance Study

In this chapter, we present the graph matching and transformation engine (GMTE), the tool that we developed to implement the graph grammars approach we proposed here (presented in subsection 3.4.5). We conducted an experimental study of our graph rewriting system using the rules of the architecture adaptation models presented earlier. We show a study related to the reconfiguration on the application layer that are based on SWRL rules. We show the efficiency of our graph grammar reconfiguration used on the middleware layer. We show in this chapter how graph grammars can be used to design policy-driven reconfiguration mechanisms of architectures and to rule running applications using reconfiguration laws for Web Service-Based Applications. We conducted a simulation of ERCMS use cases. This simulation shows the manner in which ERCMS entities collaborate to maintain connectivity and enhance the quality of communication.

4.1 Graph Matching and Transformation Engine

The GMTE is implementied in C++. It is an efficient implementation of an extension of Messmer's algorithm ([Mes95]). This tool is capable of searching small and medium graph patterns in huge graph in a short time. A computational complexity analysis of the algorithm has been conducted [GUE06] and performant experimental results have been obtained. It has also been proved that, when only constant labels are considered,

this complexity is similar to the complexity of Ullmann's algorithm ([Ull76]). Both the pattern graph (called rule graph) and the host graph have labelled nodes and edges. The rule graph labels may be totally or partially instantiated. Unification is conducted for non-instantiated labels. The tool can be used non-interactively as a C++ library providing a function that can be invoked from either a C++ or a Java program. The tool can be used through as a C++ executable that reads the rule graph and the host graph description from input TXT or XML files. The XML standard used is GraphML (Graph Markup Language). GraphML [BEH+01] is an XML-based file format for graphs. It consists of a language core to describe the structural properties of a graph and a flexible extension mechanism to add application-specific data. Its main features include supporting directed, undirected, and mixed graphs, hypergraphs, hierarchical graphs, graphical representations, references to external data, application-specific attribute data, and light-weight parsers. Unlike many other file formats for graphs, GraphML does not use a custom syntax. Instead, it is based on XML and hence ideally suited as an interoperability format for all kinds of services generating, archiving, or processing graphs.

A GraphML document is composed of a GraphML element and a variety of subelements: graph, node, edge. A graph is denoted by a graph element. Nested inside a graph element are the declarations of nodes and edges. A node is declared with a node element, and an edge with an edge element. Our C++ tool has been associated with a graphical user interface (see Figure 4.1) composed of the following zones and components:

- A menu bar offering to the user items to manipulate the interface contents such as creating, deleting, saving projects, graphs and rules.
- A tool bar that the user can use to edit graphs and rules (saving, undo, redo...).
- Project explorer giving the user a tree representing the list of opened projects, graphs and rules.
- A component panel containing a list of buttons for creating nodes and edges.
- A graph representing zone which offers to the user the possibility to open and show graphs she/he is manipulating.

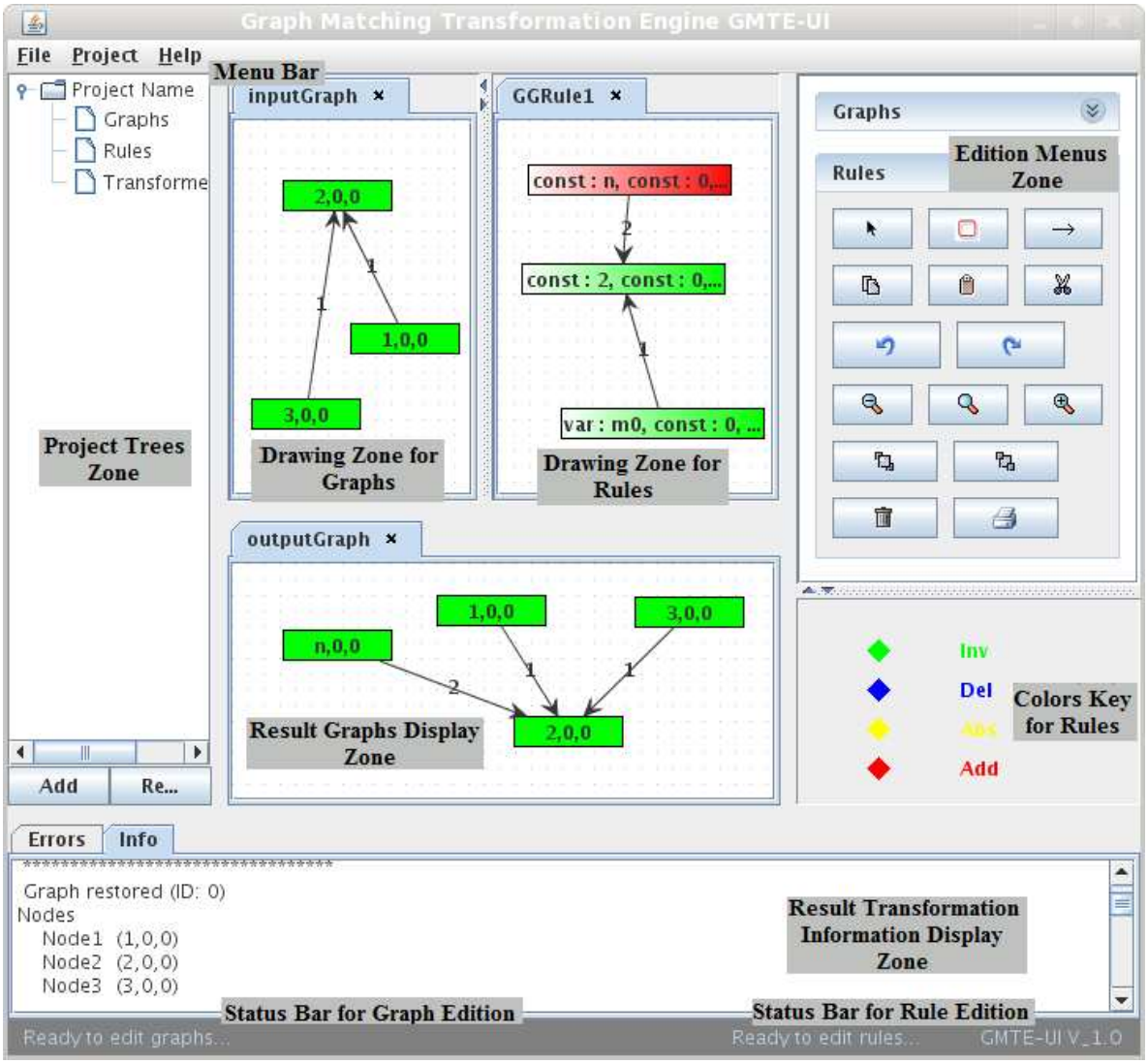


Figure 4.1: The Graphical User Interface

- A rule representing zone which offers the user the possibility to open and show rules she/he is manipulating.
- Transformed graph zone which offers to the user has the possibility to open and show graphs she/he had transformed.
- A rule legend with which the user can distinguish between rule zones *(Inv, Del, Abs, Add)*.

  - The *Inv* zone: representing a fragment of the graph rule which must be identified (by homomorphism) in the input graph, if several subgraphs are homomorphic to this area, a subgraph is chosen randomly. This fragment remains invariant after the execution of the rule;
  - The *Abs* zone: this zone represents a fragment of graph rule (containing the fragment *Inv*) which must not exit (by homomorphism) in the graph. If this fragment exists, the rule can not be executed;
  - The *Del* zone: is under a fragment of the *Inv* zone that will be deleted when executing the rule;
  - The *Add* zone: is the fragment that will be added after executing the rule.

- Two tabs showing the information and the errors when transforming a graph.

The user can export graphs and rules from the tool to TXT format or XML format according to the standard GraphML format as well as an image. The interface offers an export wizard which gives the user the possibility to specify the export parameters, such as file name, storage directory and the export format. The exported XML graph file (see Figure 4.2) is composed of a graph element containing a list of nodes and edges elements with different attributes describing each element. The exported XML rule file (see Figure 4.3) is composed of a rule element containing a list of nodes representing the different zones of the rule *(Inv, Del, Abs, Add)*. Each zone element is composed of a graph containing a list of node and edge elements with different attributes.

4.2 Emergency Response and Crisis Management Systems Case Study

We consider the example of Emergency Response and Crisis Management Systems (ERCMS). ERCMS-like activities involve structured groups of participants who are communicating to achieve a common mission (e.g. save human lives, fight against a huge fire, etc). Communication relationships between participants are subject to evolve throughout the mission. By nature, ERCMS-like activities cannot be deployed over a wired/static infrastructure which are subject to destruction or non existence. This kind of activities need ad-hoc networks with mobile devices for communications. To

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:schemaLocation="http://graphml.graphdrawing.org/xmlns
         http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="0" edgedefault="directed">
    <node id="SWRLInjected_20">
      <data key="attribute1">EP</data>
      <data key="attribute2">audio</data>
      <data key="attribute3">red_RS_session</data>
      <data key="attribute4">dev5</data>
    </node>
    <node id="red_RS_session">
      <data key="attribute1">CM</data>
      <data key="attribute2">audio</data>
      <data key="attribute3">red_RS_session</data>
      <data key="attribute4">dev5</data>
    </node>
    <node id="SWRLInjected_21">
      <data key="attribute1">EC</data>
      <data key="attribute2">audio</data>
      <data key="attribute3">red_RS_session</data>
      <data key="attribute4">dev4</data>
    </node>
    <edge source="SWRLInjected_20" target="red_RS_session">
      <data key="attribute1">push</data>
      <data key="attribute2">-</data>
    </edge>
    <edge source="red_RS_session" target="SWRLInjected_21">
      <data key="attribute1">push</data>
      <data key="attribute2">-</data>
    </edge>
  </graph>
</graphml>
```

Figure 4.2: The XML Graph File

support such group activities, future applications or network-oriented services should be dynamically activated in response to implicit or explicit requests. These services should be accessible independently of the users' location and access point, wired or wireless. They should take into account different time-varying requirements depending on the targeted activity, users' mobility, exchanged data flows (e.g. audio, video), and

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns:xsi="http://www.w3.org.2001/XMLSchema-instance"
         xmlns:schemLocation="http://graphml.graphdrawing.org/xmlns
         http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd"
         xmlns:y="http://www.yworks.com/xml/graphml">
  <rule id="1">
    <zone  zoneName="inv">
        <node nodeId="CM" nodeTypeId="var">
                <attributNode1  typeAttributNode1="const">audio</attributNode1>
                <attributNode2  typeAttributNode2="var">dev5</attributNode2>
        </node>
    </zone>
    <zone  zoneName="del">
          <node nodeId="SC" nodeTypeId="var">
                <attributNode1  typeAttributNode1="const">audio</attributNode1>
                <attributNode2  typeAttributNode2="var" >dev2</attributNode2>
          </node>
    </zone>
    <zone  zoneName="add">
    </zone>
    <zone  zoneName="add">
          <node nodeId="EP" nodeTypeId="const">
                <attributNode1  typeAttributNode1="const">audio</attributNode1>
                <attributNode2  typeAttributNode2="const">dev1</attributNode2>
          </node>
    </zone>
    <edge value="red_RS_session" source="EP,audio,dev" target="CM,audio,dev5"/>
  </rule>
</graphml>
```

Figure 4.3: The XML Rule File

different time varying constraints such as variable communication and device resources. Moreover, in ERCMS-like group activities, changes in the cooperation structure between users should also be operated in response to different events such as decisions and instructions from the mission's coordinator or information acquired by the participants.

4.2.1 ERCMS Case Study Description

The scenario involves different types of mobile actors which carry different types of communication devices. We distinguish human actors who may be professional actors with special communication devices or occasional actors who carry a mobile device (e.g PDAs, Phones). We distinguish also, robot actors like planes, helicopters and ground robots. For all actors, the communication system must deal with unexpected

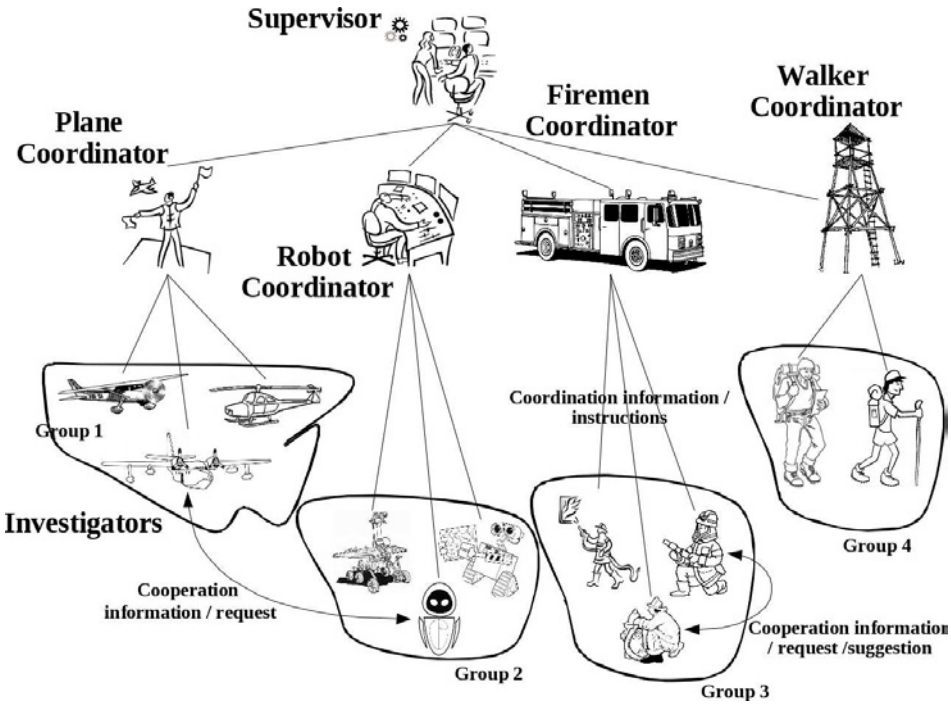


Figure 4.4: ERCMS Mission Description

or expected evolution of user needs or the changes due to device/network constraints. We can distinguish different steps during the mission. We give describe the two most representative execution steps: "Investigation step" (for the localization and the identification of the crisis situation), and "Action step"(after the identification). ERCMS-like activities are based on information exchange between mobile participants collaborating to achieve a common mission. To be more generic, we define the different participants roles: The supervisor of the mission, the coordinators, and the field investigators. Each group of investigators is supervised by a coordinator (see Figure 4.4). Each participant is associated with an identifier, a role and the devices he/she/it uses. Each participant performs different functions:

- The supervisor's functions include monitoring and authorizing/managing actions to be done by coordinators and investigators. The supervisor is the entity which supervises the whole mission. He waits for data from his coordinators who synthesize the current situation of the mission, and is characterized by having permanent energy resources, and high communication and CPU capabilities.

- Coordinators have to manage an evolving group of investigators during the mission and to assign tasks to each of them. The coordinator has also to collect, interpret, summarize and diffuse information from and towards investigators. The coordinator has high software capabilities and hardware resources. In Figure 4.4, we can distinguish 4 coordinators:
  - The robot coordinator.
  - The plane coordinator.
  - The firemen coordinator that manages professional actors.
  - The walkers coordinator (located in a watch tower) that manages occasional actors.
- The investigator's functions include exploring the operational field, observing, analyzing, and reporting the situation. Investigators also act for helping, rescuing, and repairing.

In Figure 4.4, we can distinguish coordination flows and cooperation flows. The coordination flows are between investigators and their corresponding coordinator as well as between the coordinators and the supervisor. The investigators transmit coordination information to the corresponding coordinator such as: feedbacks "D" that are Descriptive data, and feedbacks "P" that are Produced data; they express the analysis of the situation by an investigator. The supervisor function's includes in supervising the whole mission, i.e. deciding actions to be performed, and sending coordination instructions. Cooperation flows are between investigators of the same group (A2A type: fireman2fireman, robt2robot, etc) or between investigators of different groups (A2B type: robot2fireman, plane2firman, walker2fireman, etc). In the case of A2A cooperation flow, we can distinguish:

- cooperation information: a robot can inform another about a field state (There is a water there).
- cooperation requests: a fireman can ask another for help (Help me to bring this heavy object).
- cooperation suggestions : a plane can warn another (Do not go backward).

In case of A2B cooperation flow, we can distinguish

- cooperation information: a robot can inform a plane about a field state (there is a robot in the area).
- cooperation requests: a fireman can warn a plane (do not dump water now/here).

We associate priorities to flows according to the mission's structure. Different priorities could be associated with different flows or terminals, according to the importance of the participant role, and also to the resource communication state or the flow type (cooperation, coordination, A2A, A2B, information, instructions, requests, and suggestions). Figure 4.5 depicts the ontology that models the business concepts and its relations to the ERCMS activity. The main concept is the `Participant` which has several properties. The different types of participant (`Supervisor`, `Coordinator`, and `Investigator`) are modelled as sub-concepts where each of them has its own additional properties. A `Participant` belongs to a `Group` that is lead by a `Manager`. A `Manager` can be a `Supervisor` who manages a `CoordinatorGroup` or a `Coordinator` who manages an `InvestigatorGroup`. The other important idea of this ontology is the `Entity` concept. In fact, a `Participant` is also an `Entity` with two sub-concepts: `Artificial` and `Human`. Various human participants are represented as concepts e.g. `Fireman`, `Pilot`, or `Walker` and also the artificial entities e.g. robots or vehicles, for instance. The different types of robots (`Drone`, and `GroundRobot`) are modelled as sub-concepts of `Robot`, each one with its own additional properties. This is related to the Generic Collaboration Ontology (`GCO`) [STV08], because, as shown in Figure 4.5, `GCO:CommunicationFlow` is defined as a sub-concept of `Flow` and `Entity` is defined as a sub-concept of `GCO:Node`. This means that participants' roles are defined in the collaboration ontology, and thus they inherit all their properties. For example, they have a related `GCO:Node` that is deployed on a `GCO:Device`, etc.

4.2.2 Context Description

In our work, we target the adaptation of distributed cooperative applications due to different changing parameters. These parameters may include environmental constraints (like connectivity, energy level, available memory, etc) or the evolution of the cooperation environment where participants can change roles, memberships, and functions. These parameters constitute the context of the cooperative application. We have defined a context ontology (see Figure 4.6) to structure and organize these parameters.

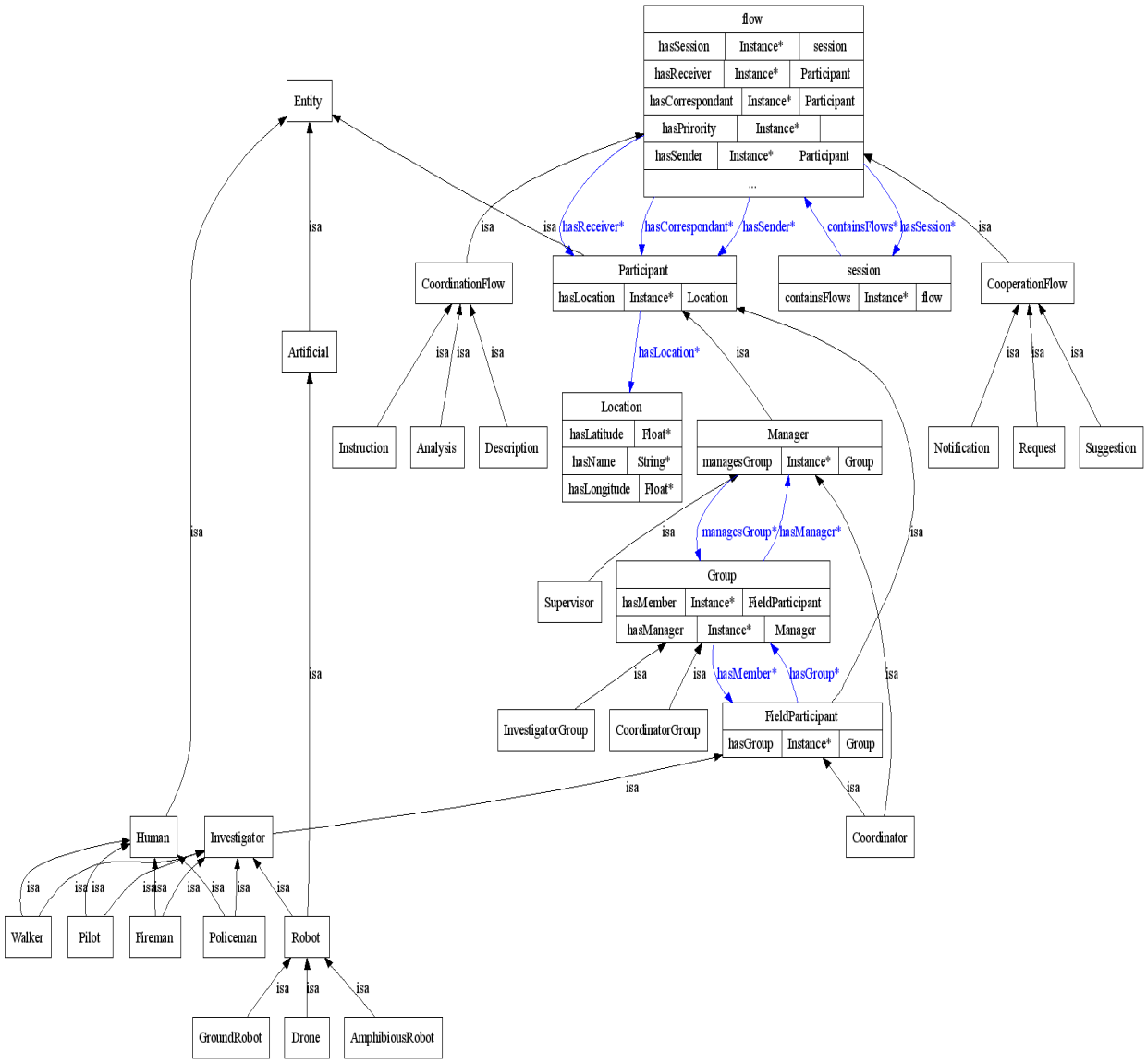


Figure 4.5: The ERCMS Ontology Description

Our context ontology describes the cooperation activity, the involved participants and the environment constraints (machine constraints). It contains two levels: the generic level and the specific level. The first level is independent of the application and the activity types. The second level populates the generic one by specific terms to an application category (like ERCMS applications). The generic level contains a concept

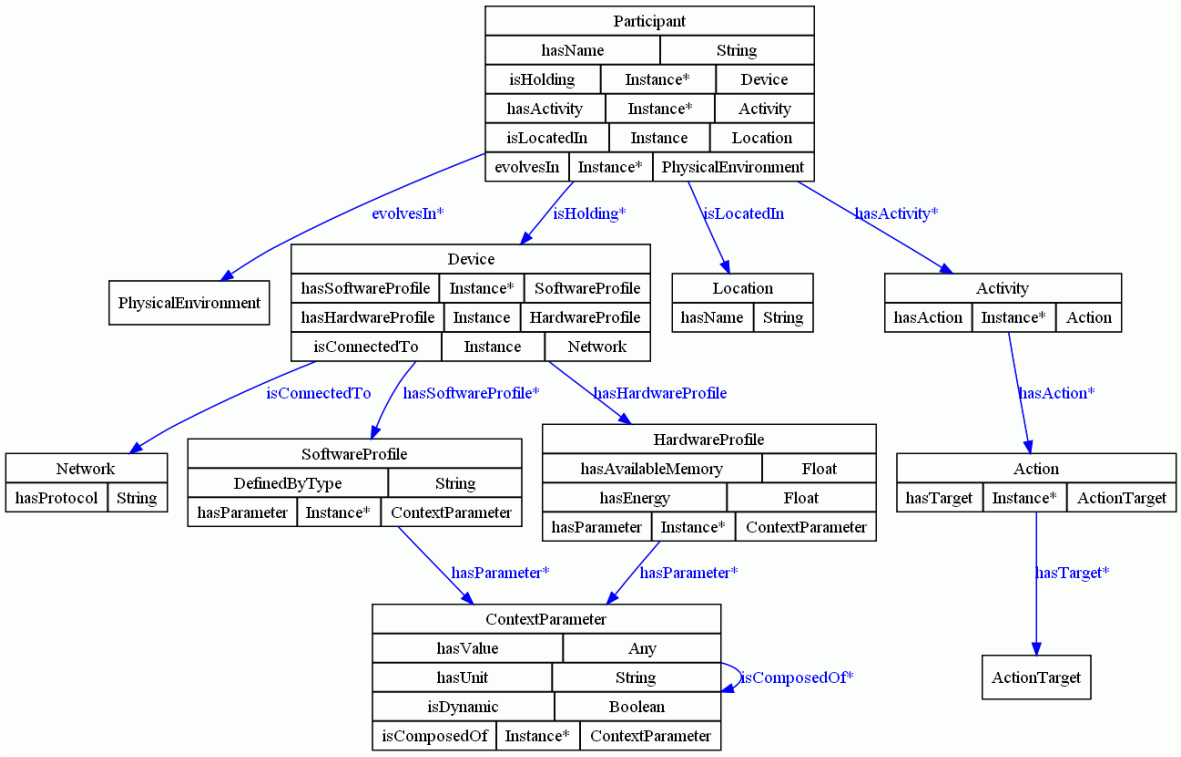


Figure 4.6: General Structure of our Context Ontology

`Participant` that specifies the basic profile (name, login, etc) of a participant in the cooperative activity and her/his current function. The concept `Location` describes the physical space, and spatial relations of involved devices and users in the system. Locations can be specified by physical representations (GPS coordinates) and symbolic high level representations (like places or cells identified by their names). The concept `Device` describes the hardware (like available memory and CPU usage), and the software (like operating system and installed applications) capabilities of the user device. The concept `Activity` specifies the properties of the user's activity: in progress, starting dates, ending dates, etc. The concept `Network` describes the characteristics of the network, such as the transport protocol, the connectivity, and the QoS of the available connections. Finally, the concept `PhysicalEnvironnement` provides physical properties of the user space (like ambient temperature) and her/his relation with her/his external environment. The specific level ontology depends on the application type. It populates the generic level (with *isa* relationships) to add domain specific concepts that

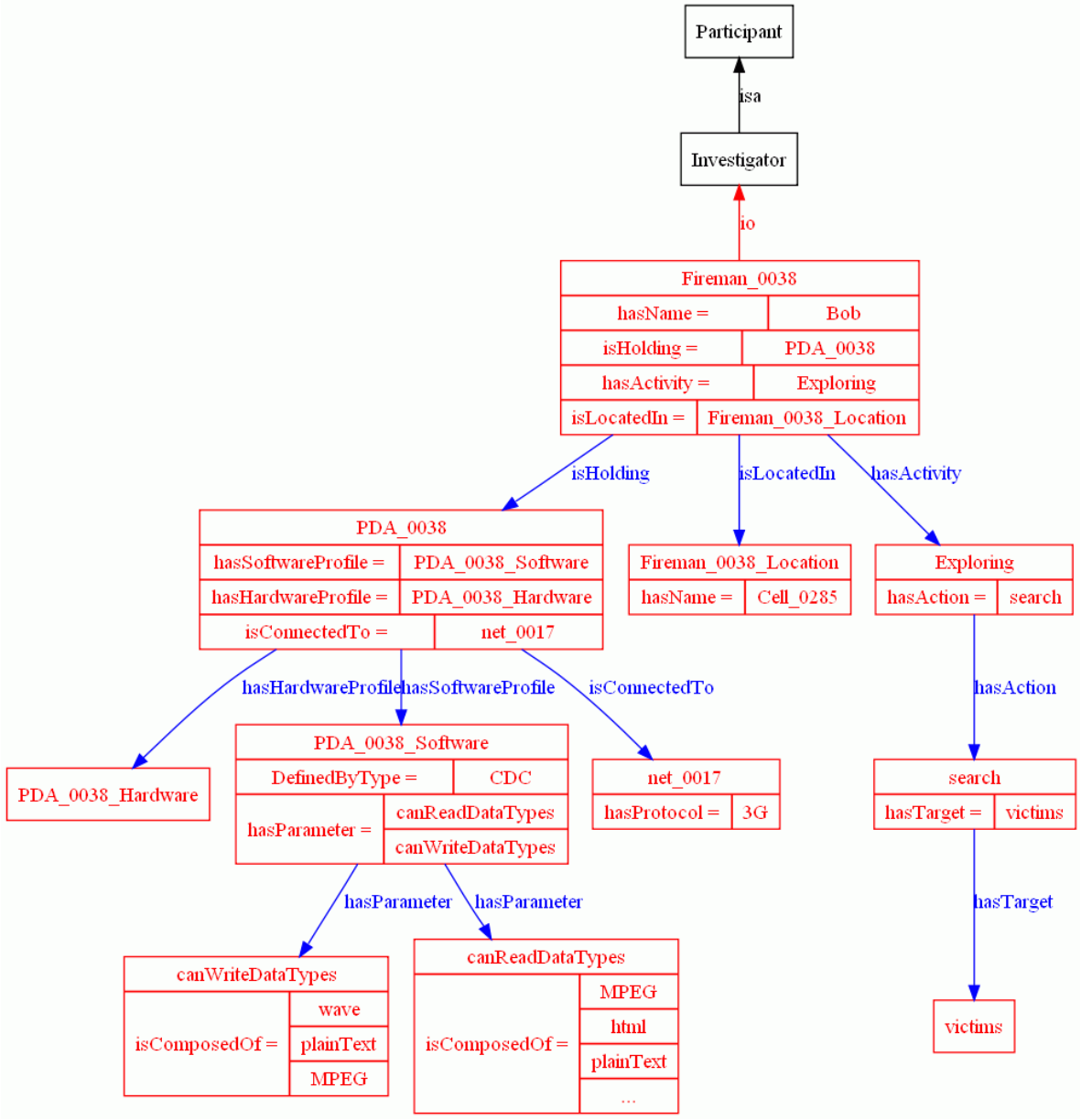


Figure 4.7: Instance Example from our Context Ontology

represent the application logic. For example, in our case study we defined, the concept `Participant` from the generic level is populated in three subclasses: `Supervisor`, `Coordinator`, and `Investigator`. This specific level also contains the real-world instances of the defined concepts. Let us quote our illustration example where a fireman

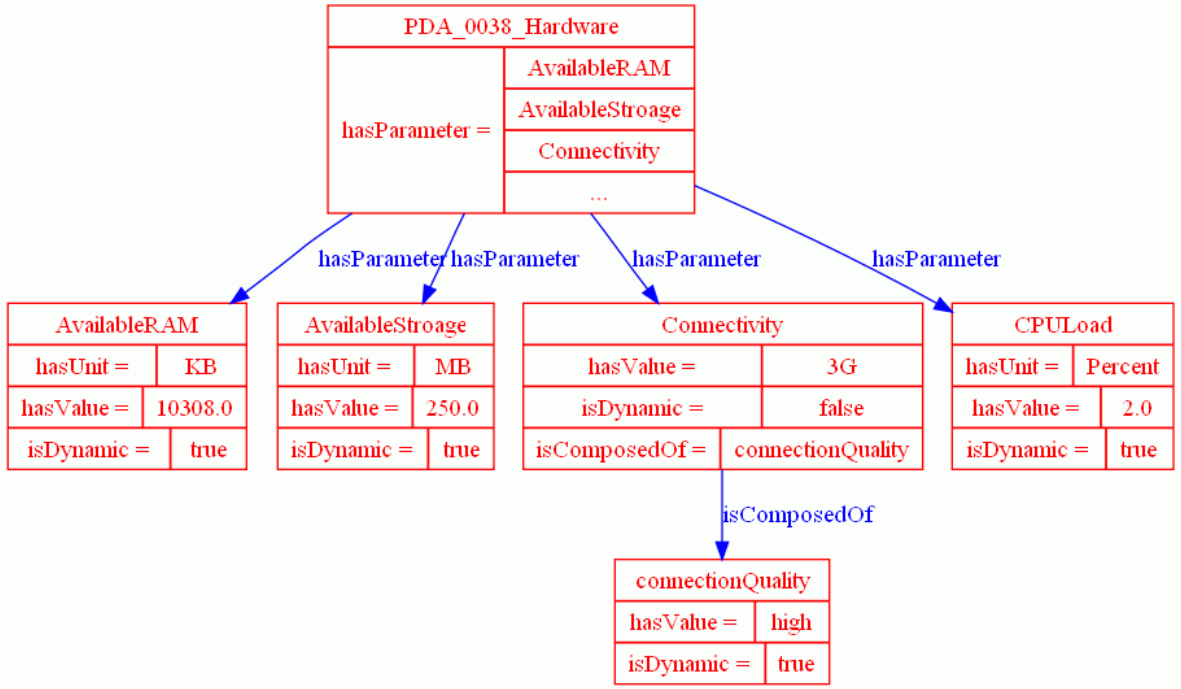


Figure 4.8: Instance Example of a Device Hardware Profile

(`Investigator`) holding a smart phone (`Device`) equipped with a camera connected to the firemen coordinator (`Coordinator`) through a 3G communication (`Network`). He is exploring (`Activity`) the field at a specific zone (`InterpretedLocation`) identified by his 3G signal power (raw Location information). Figure 4.7 presents a simplified view of this case. Figure 4.8 presents a simplified view of a smart phone (`Device`) equipped with an IP-enabled camera.

4.2.3 Initial Refinement and Adaptation Process Examples

Our adaptation policy associates the suitable adaptation transformations with the corresponding context changes. We use SWRL rules [HPSB+04a] to define our adaptation policy. The application designer defines these rules according to context changes that he wants to handle. The header part (swrl:head) of these rules references the adaptation transformations. The body part (swrl:body) references the context ontology elements. These rules are executed when the context changes. They react to different events concerning the cooperation environment and the context values.

4.2.3.1 Top-Down Refinement Example

This subsection presents a top-down example showing the architecture models of the three considered layers as well as the refinement processes between layers. The adaptation process is also illustrated. In this example, we focus on group3 of Figure 4.4 in order to illustrate the initial refinement. Group3 has three investigators Fireman1, Fireman2, and Fireman3. There is a cooperation flow between Fireman1 and Fireman2. The firemen coordinator manages group3 and has a coordination flow with each fireman and the supervisor. This business level architectural configuration is captured by the game application and represented in the activity ontology $A_{3,0}$. Concepts and relations from the activity-specific ontology and from the generic collaboration ontology are instantiated together. Generic collaboration rules are applied. In this case, the rule presented in Table 3.3 is processed for each instance of `DataFlow` found, thus creating the corresponding `SenderComponent` and `ReceiverComponent` at the endpoints of the audio flow. The resulting set of ontology instances represents a collaboration layer graph. This graph is translated into GraphML language. The resulting collaboration graph contains 5 senders (S) and 3 receivers (R). The graph edges correspond to data flows, and are labelled by data type (*audio*), and by the session to which they belong. Each component has three attributes: the identifier, the type (sender or receiver), and the deployment machine identifier. In order to refine this collaboration layer graph, the $GG_{COLLAB \rightarrow EBC}$ graph grammar, detailed in Table 3.5, is used. This produces a set of valid configurations $\mathbb{A}_1^0$. The procedure `Selection()` is used in order to find the optimal configuration. The retained configuration, $A_{1,0}$, is presented in Figure 2, depicting the middleware layer. This configuration contains only terminal nodes (i.e. nodes belonging to the EBC layer), and is obtained by the application of the sequence $Sp = p_1; p_1; p_1; p_2; p_2; p_4$. Production p_1 refines the pattern consisting of a S and R linked. Productions p_2 and p_3 refine patterns for other Rs and Ss. Production p_4 eliminates redundant Channel Managers. This refinement creates a detailed deployment descriptor which is used by the deployment service in order to deploy the indicated components on each device, thus implementing the required application layer session.

4.2.3.2 External Context Changes Adaptation

In this subsection, we show examples of external context changes adaptation which are due to evolution on the mission. The first situation is where a cooperation flow is

needed between two investigators to avoid a conflict. The second case correspond to the discovery, by an investigator, of a critical situation of an injured person. The situation requires adaptation of the communication schema. The third case corresponds to an addition of a new investigator to the mission.

4.2.3.2.1 Investigators Cooperation We consider a situation where an investigator from group1 (e.g a plane in Figure 4.4) wants to drop water over an area in which another investigator from group3 (e.g Fireman1 in Figure 4.4) is already in action. In this case, the plane has to be notified as soon as possible, not to drop water on that area. Another investigator from group 3 (e.g Fireman2), after seeing the plane, establishes a coordination flow to its coordinator and subsequently the coordinator indicates the situation to the supervisor. In practice, we can suppose that the supervisor knows already the position of the approaching plane. Then the supervisor notifies the plane not to drop the water on that particular area via the plane coordinator. As there is no connection between the plane and the Fireman2, Fireman2 needs to obtain the supervisor's decision that takes several steps. The other simpler solution could be to make a connection between the plane and Fireman2 by establishing a new cooperation flow by running the SWRL rule (see Table 4.1). This rule is executed when there is a need to establish a cooperation flow between the investigators among different groups. By this, Fireman2 can tell the plane "Do not drop the water". After the establishment of the new cooperation flow between the Fireman2 and the plane, a channel manager has to be installed in the device of Fireman1. Thus, the corresponding event producer and event consumer need to be deployed in the plane.

4.2.3.2.2 Critical Situation Discovery We consider a situation where an investigator from group3 (e.g Fireman1) discovers an injured person. Fireman1 needs more cooperation flows with other firemen (Fireman2 in our case) to help this person and with a physician to check the status of this person. We show the SWRL rule (see Table 4.2) that expresses a critical situation discovery by an investigator. This rule is executed when the value of *hasStatus* attribute in the context ontology changes. If an investigator detects a critical situation, a transformation is performed to establish the cooperation flows needed.

```
Supervisor(?s)∧Coordinatorgroup(?CG)∧managesGroup(s,CG)∧
Coordinator(?co1)∧Coordinator(?co2)∧hasMember(CG,co1)∧
hasMember(CG,co2)∧InvestigatorGroup(?IG1)∧
InvestigatorGroup(?IG2)∧managesGroup(co1,IG1)∧
managesGroup(co2,IG2)∧Investigator(?I1)∧
Investigator(?I2)∧hasMember(IG1,I1)∧
hasSender(IG2,I2)∧CoordinationFlow(?cf1)∧
hasReceiver(cf1,co1)∧ hasSender(cf1,s)∧
CoordinationFlow(?cf4)∧hasReceiver(cf4,co2)∧
hasSender(cf4,s)∧CoordinationFlow(?cf3)∧
hasReceiver(cf3,I1)∧hasSender(cf3,co1)∧
CoordinationFlow(?cf5)∧hasReceiver(cf5,I3)∧
hasSender(cf5,co2)
→SWRLb:createOWLThing(CooperationFlow(?cpf1))∧
hasSender(cpf1,I2) ∧ hasReceiver(cpf1,I1)
```

Table 4.1: Establishing a Cooperation Flow SWRL Rule

```
Investigator(?inv2) ∧ Investigator(?inv1)∧
hasID(?inv1,?i)∧ hasActivity(?inv1,?activity)∧
sameAs(?activity,exploring)∧hasAction(?activity,?a)∧
hasResult(?a,?result)∧ swrlb:equal(?result,true)
→ SWRLb:createOWLThing(CooperationFlow(?cpf1))∧
hasSender(cpf1,inv1) ∧ hasReceiver(cpf1,in2) ∧
SWRLb:createOWLThing(CooperationFlow(?cpf2))∧
hasSender(cpf2,inv1) ∧ hasReceiver(cpf,inv3)
```

Table 4.2: Critical Situation Discovery SWRL Rule

4.2.3.2.3 Adding a New Investigator We consider a situation where an investigator (e.g Fireman4) wants to join group3 (firemen group). Fireman4 needs to establish a coordination flow with the firemen coordinator. We show the SWRL rule (see Table 4.3) that allows the addition of a new investigator (Fireman4 in our case). This rule is executed and thus a coordination flow is established with the firemen coordinator.

```
Supervisor(?s)∧Coordinatorgroup(?CG)∧
managesGroup(s,CG)∧ Coordinator(?co1)∧
hasMember(CG,co1)∧InvestigatorGroup(?IG1)∧
managesGroup(co1,IG1)∧Investigator(?I1)∧
Investigator(?I2)∧hasMember(IG1,I1)∧
CoordinationFlow(?cf1)∧hasReceiver(cf1,co1)∧
hasSender(cf1,s)∧CoordinationFlow(?cf3)∧
hasReceiver(cf3,I1)∧hasSender(cf3,co1)∧
→ SWRLb:createOWLThing(CoordinationFlow(?cf5)∧
hasReceiver(cf5,I2)∧hasSender(cf5,co1)
```

Table 4.3: Adding a New Investigator SWRL Rule

4.2.3.3 Resources Context Changes Adaptation

When the resource context changes (like a RAM saturation, a CPU overload or an energy level decrease) occurs, the adaptation process explained is triggered. In this subsection, we show examples of resources context change adaptation that are due to evolution on the mission or due to the constraints of the environment (e.g fire) or the nodes (e.g not enough energy). The first situation, illustrates the case of an important diminution of the energy level of the device of the firemen coordinator. The second situation corresponds to the case of communication degradation due to mission needs. The middleware layer, in this case, reacts and provides solutions in the EBC layer and if it is impossible, it provides suggestions to the mission supervisor.

4.2.3.3.1 Energy level Diminution For instance, if there is an important diminution of the energy level of the device of the firemen coordinator, the EBC layer launches the `Select()` procedure. This procedure chooses a new configuration (e.g. $A_{1,5}$) which is adapted to the new context parameters. In the configuration $A_{1,5}$, the two CMs previously deployed on the firemen coordinator device are moved to its neighbours: the firemen coordinator, Fireman2, and Fireman3. Within this new configuration, the firemen coordinator device has fewer components deployed on it, and thus its energy consumption will decrease. Later on, the energy level of the firemen coordinator device will fall below a threshold for which the firemen coordinator's device can no more accept components hosting. In this case, the triggered selection process will be unable to find a valid middleware layer configuration suitable for the new context. This means that

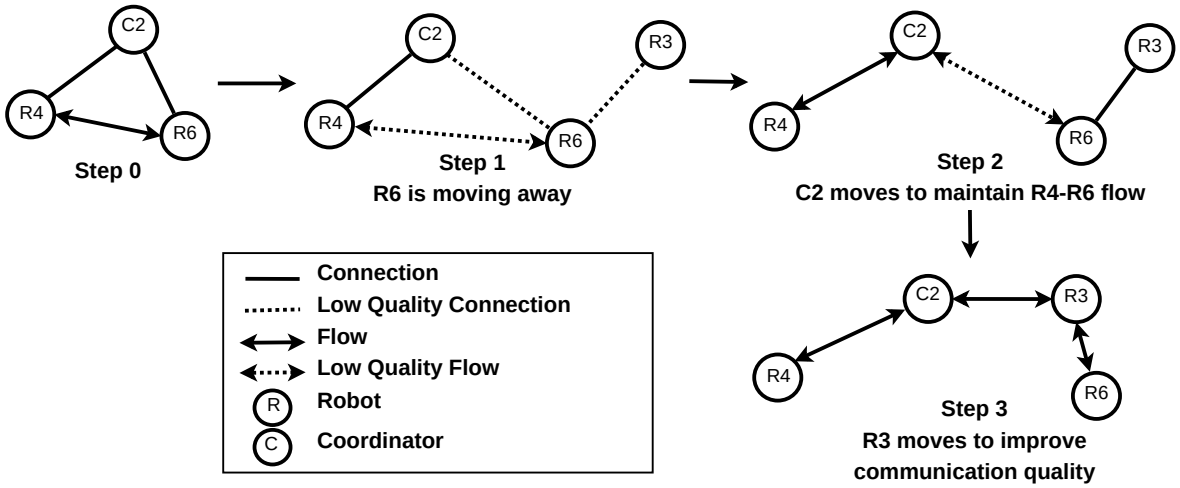


Figure 4.9: Adaptation for Connectivity and QoS Preservation

the required application layer configuration can not be implemented within this context. Therefore, the middleware layer sends a reconfiguration event to the application layer. This event is processed in the application layer in order to produce a new better adapted configuration. For instance, we consider that the application layer decides to permute the roles of the firemen coordinator, and Fireman3: Fireman3 becomes a firemen coordinator and the firemen coordinator becomes a regular fireman: Fireman4. This reconfiguration leads to a refinement similar to the one previously presented. This minimizes the number of components deployed on the device of Fireman4.

4.2.3.3.2 Adaptation to Preserve Connectivity and to Improve QoS We consider a group of three robots with their corresponding robot coordinator. This corresponds to group3 of Figure 4.4. Figure 4.9 gives a summary of the sequence of events of the scenario.

- Step 0. Initially two ground robots (Robot4 and Robot6 in see Figure 4.9) are communicating. The communication flow between Robot4 and Robot6 is considered important for their cooperation.

- Step 1. In relation with mission objectives (rescue), Robot6 has to move away. This step could be initiated if a robot receives a recommendation from the supervisor of the mission to find an injured person in an area near to its position.

Therefore, Robot6 and Robot4 detect a loss rate increase. In this case, the triggered selection process will be unable to find a valid middleware layer configuration suitable for the new situation. This means that the required application layer configuration can not be implemented within this context. Therefore, the middleware layer sends a reconfiguration event to the application layer. Thus the problem (connectivity loss: threshold of adaptability is reached) is reported. A suggestion of adaptation is decided and sent to each robot ("Coordinator2 should move to position X to maintain connectivity").

- Step 2. We consider that Robot4 can't move from his position due to the mission constraints. To maintain the communication between Robot4 and Robot6, Coordinator2 has to move near Robot6 (step3). Coordinator2 is able to move because the application layer allows him to do so.

- Step 3. Robot6 arrives to the required position, Coordinator2 maintains the communication between Robot4 and Robot6. Unfortunately, this communication is not efficient (high loss rate). Coordinator2 detects this problem. An adaptation policy requires to improve this existing link (if possible) when a threshold of comfort is reached. A fortuitous discovery robot (Robot3) can be used as a relay for the communication. Robot3 comes close to Coordinator2 and Robot6 and thus, the communication link quality is improved.

4.3 Application of Graph Grammars Models on Web Service-Based Applications

In web service-based applications, a set of distributed services collaborate in order to respond to user's requirements. Such a composition is built while describing the whole process required by the end user, and querying a repository about the available web services (The *WS Discovery Service*). Here, we consider a generic template of architecture where a Virtual Intermediate Connector (VIC) is used to bind service requesters. The VIC can route the requests to effective service implementations depending on the operation being requested and provided, and the global or operation-specific performances of the requested service. The VIC asks the *WS Discovery Service* for required web services in order to achieve building the composed application. The research process is based

on functionalities and qualities of the desired web service. The architecture graph to which our approach applies may be characterized by any successive compositions of the elementary patterns: $N_i(ServiceRequester) \rightarrow N_j(VICType, Interface))$ and $N_j(VIC) \rightarrow N_k(ServiceProvider, performanceAttributes)$, where N_i, N_j, N_k denote the graph nodes associated, respectively, with the "Service Requester", the "Virtual Intermediate Connector", and the "Service Provider". Once the composition process is achieved, the application is ready to be deployed and used. During runtime, QoS offered by each service may be degraded and hence, needs to be substituted it by rerouting the requests for one or more operations to different service providers. A substitution is considered as the elementary architectural reconfiguration action after a QoS degradation detection or prediction for the currently blinded service. Searching for an equivalent service using the *WS Discovery Service* is required to find, partially or totally, equivalent services. Our representation approach of architectural configurations relies on graphs and graph grammars. Graph grammar theories represent an appropriate formalism to handle the reconfiguration process. Moreover, graph grammars are tractable and powerful way of handling complex transformations and characterizing the set of configurations without its explicit enumeration. Following the common representation, a node refers to a web service and a directed edge denotes an invocation link.

4.3.1 The Graph Grammar-based rule-oriented reconfiguration

We elaborate here graph grammars that implement reconfiguration policies and minimize unnecessary reconfigurations. We use the following notations in the sequel: graph nodes are represented by $N_i(att_1, \cdots att_n)$ where "i" allows a node to be identified in the graph and where $att_1, \cdots att_n$ are attributes of the node. Attributes may represent properties of the service associated with a given graph node such as: the $role : (Requester, Provider)$, the $status : (Deficient, Non_Deficient)$, etc.

4.3.1.1 Monolithic Service Substitution Policy

Table 4.4 shows the graph grammar for substituting a single WS by another equivalent. Applying the production $p1$ leads to the unbinding of the deficient WS described by the sub-graph containing the node N_2 and the edge $N_1 \rightarrow N_2$, and the binding the

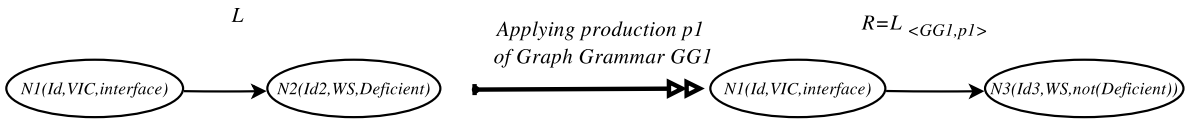


Figure 4.10: Graphical Representation of $GG1$ Production $p1$

substitute WS described by the sub-graph containing the node N_3 and the edge $N_1 \rightarrow N_3$. Figure 4.10 shows a visual representation of this production that unbinds the

$GG1 = (AX, NT, T, P)$ with: $T = \{N_1(Id_1, Type, interface), N_2(Id_2, SW, State)\}$, $NT = \{\ \}$, and $P = \{p1\}$
$p1 = ($ $L = \{N_1(Id_1, Type, interface), N_2(Id_2, WS, Deficient),$ $N_1 \rightarrow N_2\})$; $K = \{N_1(Id_1, Type, interface)\}$; $R = \{N_1(Id_1, Type, interface), N_3(Id_3, WS, not(Deficient)),$ $N_1 \rightarrow N_3\})$

Table 4.4: GG1: The Monolithic Substitution of a WS

deficient service N_1 and substitutes it by a $Non_Deficient$ service N_3.

4.3.1.2 Cost-Aware Composite Service Substitution Policy

The graph grammar described in Table 4.5 describes a composite substitution policy where requests related to a given operation Op_i of the interface of a WS are routed to a different WS implementing the same operation. This may be useful in different situations where a new service is discovered and which implements, more efficiently or less costly, the given operation. This can also be applied when a particular operation is over-requested and a where load balancing is necessary. This can also apply to a situation where an operation is implemented in a way that does not free resources and its repeated invocation leads to an increasing response time. All these situations will be summarized as an "*availability*" property of a service with respect to a given operation. Figure 4.11 and figure 4.12 show a visual representation of the productions $p1(Op_i)$ and $p2(Op_i)$ of $GG2$. Applying the production $p1$ leads to the removal of the edge

$GG2 = (AX, NT, T, P)$ with: $T = \{N_1(Id_1, Type, interface), N_2(Id_2, WS, State)\}$, $NT = \{\ \}$ and $P = \{p1, p2\}$
$p1(Op_i) = ($ $L = \{N_1(Id_1, Type, interface), N_2(Id_2, WS, Deficient),$ $N_1 \xrightarrow{\{Op_1,...,Op_n\}} N_2\}$; $K = \{N_1(Id_1, Type, interface), N_2(Id_2, WS, Deficient)\}$; $R = \{N_1(Id_1, Type, interface), N_3(Id_3, WS, not(Deficient)),$ $N_1 \xrightarrow{\{Op_1,...,Op_n\}\backslash Op_i} N_2, N1 \xrightarrow{Op_i} N_3\};)$
$p2(Op_i) = ($ $L = \{N_1(Id_1, Type, interface), N_2(Id_2, WS, Deficient),$ $N_1 \xrightarrow{\{Op_i\}} N_2\}$; $K = \{N_1(Id_1, Type, interface))\}$; $R = \{N_1(Id_1, Type, interface), N_3(Id_3, WS, not(Deficient)),$ $N1 \xrightarrow{Op_i} N_3\});)$

Table 4.5: GG2: Service Composite Substitution Policy

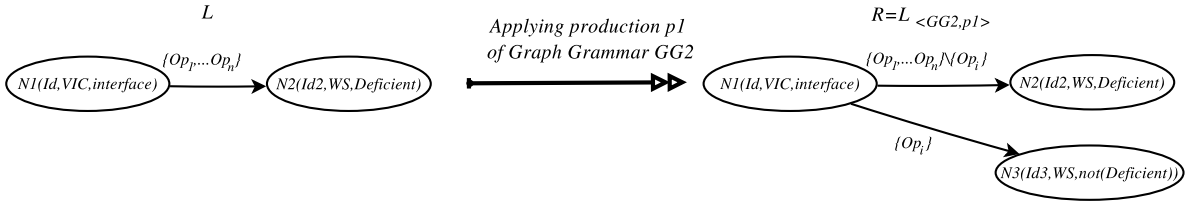


Figure 4.11: Graphical representation of $GG2$ production $p1$

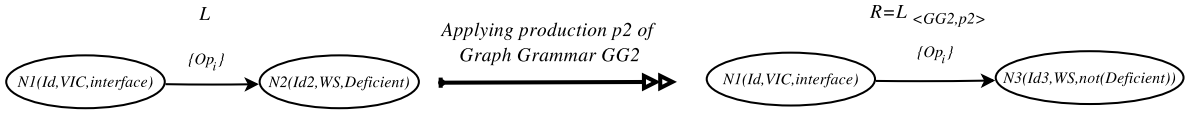


Figure 4.12: Graphical Representation of $GG2$ Production $p2$

$N_1 \xrightarrow{\{Op_1,...,Op_n\}} N_2$, and the addition of the substitute WS as described by the subgraph containing the node N_3 and the edges $N_1 \xrightarrow{\{Op_1,...,Op_n\}\backslash Op_i} N_2$ and $N_1 \xrightarrow{Op_i} N_3$. Applying the production $p2$ leads to the removal of the deficient WS as described by

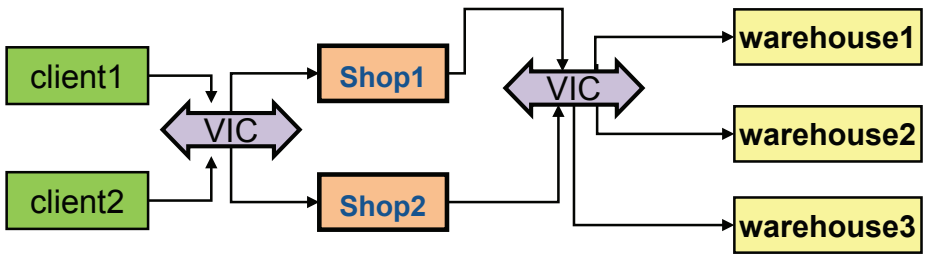


Figure 4.13: Proposed Architecture Applied to The FoodShop Case Study

the sub-graph containing the node N_2 and the edge $N_1 \xrightarrow{Op_i} N_2$, and the addition of the substitute WS described by the sub-graph containing the node N_3 and the edge $N_1 \xrightarrow{Op_i} N_3$.

4.3.2 The FoodShop Case Study: A Web Application

The FoodShop application is concerned with a web service-based application representing a company that sells and delivers food. The company has online Shops and several warehouses ($warehouse_1$, ..., $warehouse_n$) located in different areas that are responsible for stocking imperishable and perishable goods and delivering items to customers. The *VIC* is deployed within the FoodShop application between each pair of provider/requester as shown in Figure 4.13. The *Diagnosis* modules exchange information, in order to coordinate the healing actions. For instance, for the two linked services $shop_1$ and $warehouse_1$, the QoS degradation of $warehouse_1$ may propagate to $shop_1$ from the *client* point of view. This triggers a healing process within the two *VIC* instances. If not in coordination, each *VIC* substitutes its provider. However, the global reasoning about the degradation deduces that the $shop_1$ QoS degradation is due to the propagation and only $warehouse_1$ has to be substituted. The deployment of the *FoodShop* within the *VIC* enables the monitoring at the HTTP level. It extracts parameters like IP address, the deployment host, the communicating WS names, the invoked operations, the execution time and the communication type (synchronous or asynchronous). These information allow the dynamic discovery of involved parties in the application and the automatic building of the application profile. We have developed a monitoring graphical interface with the *VIC*, in addition to the self-healing features. The collected monitoring data enable us to draw up dynamically a visualization window of WS hosts

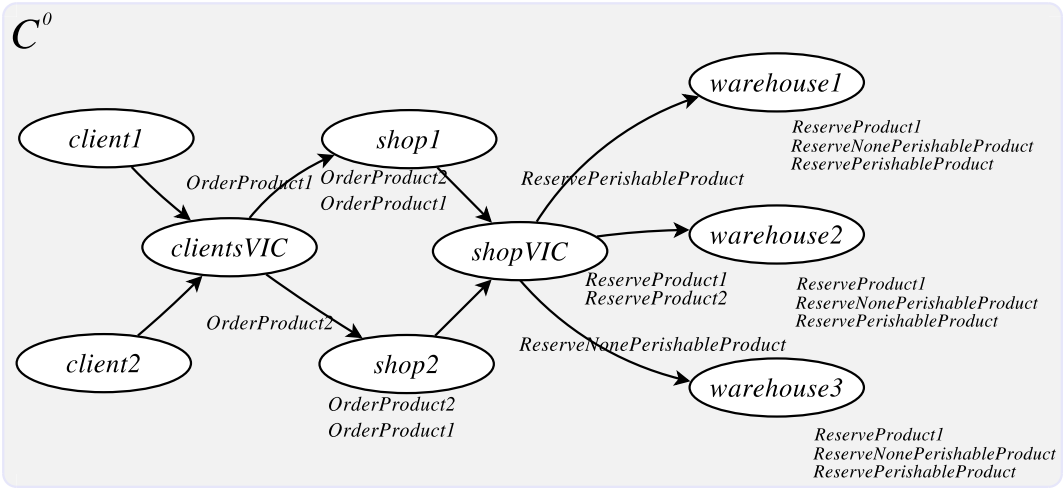


Figure 4.14: Initial Architectural Configuration

and invoked operations.

4.3.3 Application of our Approach

To illustrate our approach, we propose the following scenario. Figure 4.14 shows the initial configuration C^0. We consider two clients, two shops and three warehouses. Each shop implements two operations: Op_1 and Op_2. These operations are $OrderProduct1$ and $OrderProduct2$. Each warehouse implements four operations: Op_1, Op_2, Op_3 and Op_4. These operations are $ReserveProduct1$,$ReserveProduct2$, $ReservePerishableProduct$ and $ReserveNonePerishableProduct$. $Client_1$ invokes the operation $shop_1/OrderProduct1$ that invokes the operation $warehouse_1/ReserveProduct1$. $Client_2$ invokes the operation $shop_1/OrderProduct2$ that invokes the operation $warehouse_2/ReservePerishableProduct$. $Client_2$ invokes $shop_2/OrderProduct2$ that invokes the operation $warehouse_1/ReserveNonePerishableProduct$. To illustrate the reconfiguration policies, two events are considered. The events are "$shop_1/OrderProduct1$ degradation", and "$warehouse_2/ReserveProduct1$ degradation". In the following, we show the reconfiguration of the architecture when the first policy is used. The architecture is reconfigured according to the reconfiguration algorithm that implements, in this case, the graph grammar GG1 (see Table 4.4).

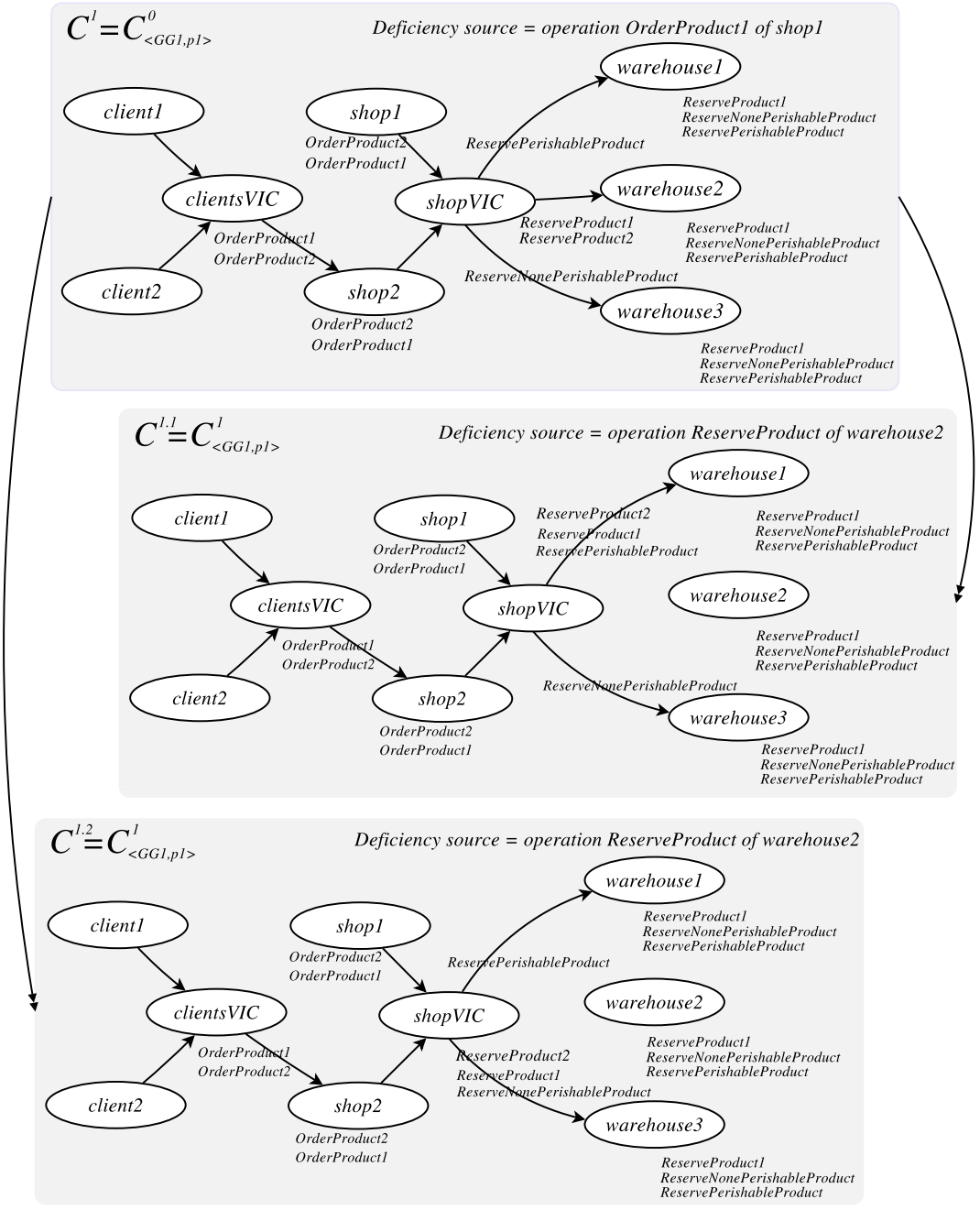


Figure 4.15: Monolithic Service Substitution

4.3.3.1 Monolithic Service Substitution Policy

When the $shop_1/OrderProduct1$ performance is considered as degraded, the WS $shop_1$ is disconnected and WS $shop_2$ is binded to the requester , in order to provide the service requested by the client as shown in $C^1 = C^0_{<GG_1,p_1>}$ (see Figure 4.15). When the $warehouse_2/ReserveProduct1$ interaction performance is considered as degraded, the WS $warehouse_2$ is disconnected and WS $warehouse_1$ provides the service requested by the $shop_2$ as shown in $C^{1.1} = C^1_{<GG_1,p_1>}$ (see figure 4.15). We can also have a WS $warehouse_3$ that provides the service requested by the $shop_2$ as it is shown in $C^{1.2} = C^1_{<GG_1,p_1>}$ instead of $warehouse_1$. This policy represents a basic solution that is efficient in case of a limited service number but in some cases generates unnecessary reconfiguration. This is the case, for example, when $warehouse_2$ is unbound with only one operation ($ReserveProduct1$) at the origin of the QoS degradation. This triggers the routing request to the equivalent operation $ReserveProduct2$ of $warehouse_2$ that is unnecessary ($ReserveProduct2$ is working).

4.3.3.2 Composite Service Substitution According to WS Cost Policy

To avoid unnecessary configuration enumeration, we propose a policy that replaces the degraded operation based on its cost. The architecture is reconfigured according the graph grammar GG2 (see Table 4.5). When the $shop_1/OrderProduct1$ interaction performance is considered as degraded, the $OrderProduct1$ of WS $shop_2$ provides the service requested by the client as shown in $C^1 = C^0_{<GG_2,p_2>}$ (see Figure 4.16). When the $warehouse_2/ReserveProduct1$ interaction performance is considered as degraded, the $ReserveProduct1$ of WS $warehouse_1$ provides the service requested by $shop_2$ as shown in $C^{1.1} = C^1_{<GG_2,p_2>}$ (see Figure 4.16). Here, there is another alternative that using $ReserveProduct1$ of $warehouse_3$, but $ReserveProduct1$ of WS $warehouse_1$ costs less. Hence, the use of this policy avoids the generation of unnecessary reconfigurations, and $warehouse_2$ is still providing the operation $ReserveProduct2$.

4.4 Evaluation Experiments and Simulations

We contacted evaluation experiments useing our engine GMTE for executing the graph grammar transformations, and the rule engine, Jess, for executing the SWRL rules. The experiments have been achieved under grid *Grid'5000* [CCD+05], an experimental grid

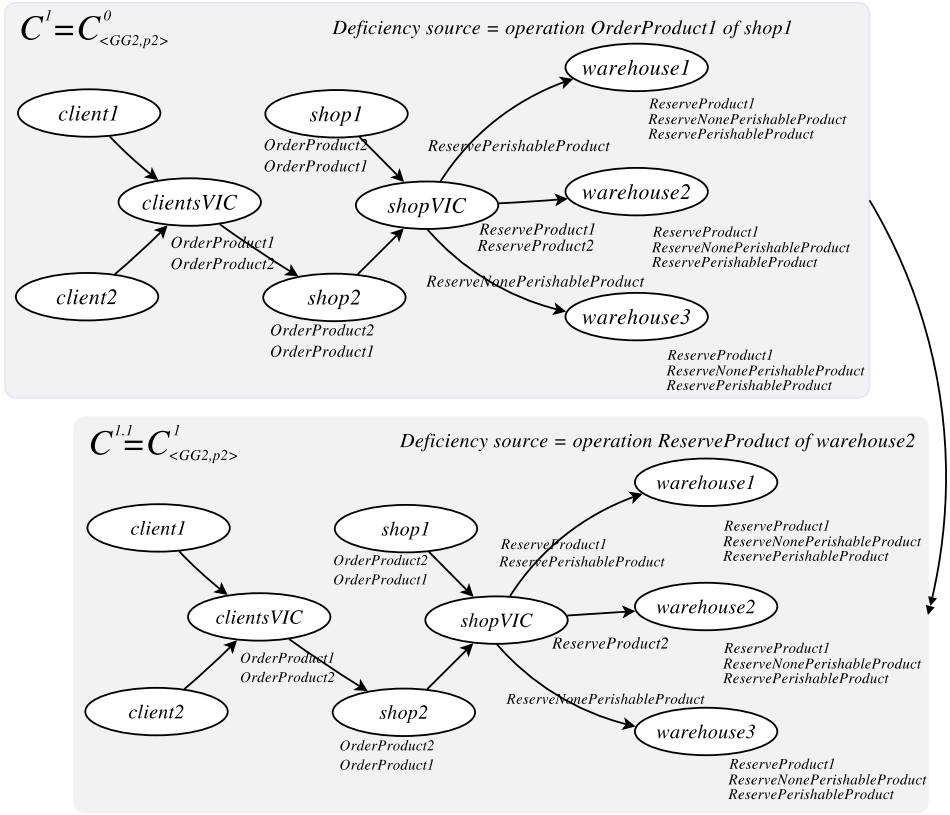


Figure 4.16: Composite Service Substitution According to WS Cost

platform that interconnects clusters geographically distributed in France. Considered grid nodes are single core machines (Sun Fire V20z) with an AMD Opteron 248 processor with 2.2 GHz of CPU speed and 2 GB of RAM. The provided network bandwidth is up to 10GB/s. The results will help us studying scalability and limits of the current models to investigate the tractability of an anticipation approach for adaptability. The experimental performance evaluation focuses on execution time. The experiment uses 30 machines in parallel. We limited our experiments to 30 machines in one site (Toulouse). This number can be assumed to be representative for commonly available computing infrastructure of a great number of companies and labs. Using such number of machines gives a realistic idea about the tractability of our approach in real situation for design-time purposes.

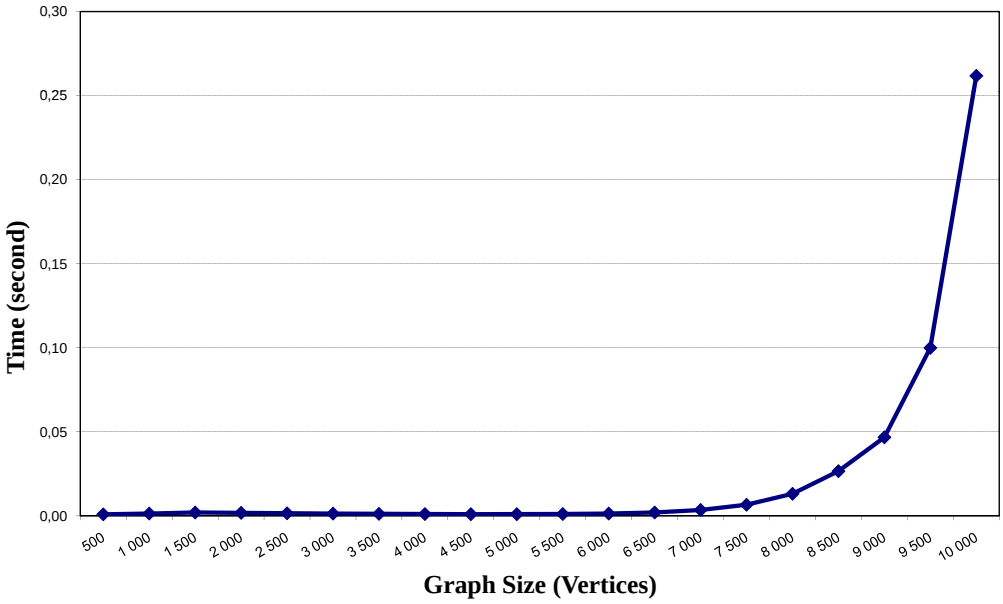


Figure 4.17: Execution Time of Applying SWRL (Table 4.1) wrt Architecture Graph Size

4.4.1 Evaluation of Adapting Architecture to Organization Changes

The results, presented here, were obtained for the reconfiguration rules executed and triggered by the SWRL rules (see Table 4.1 and Table 4.2) that correspond to the flow establishment explained in section 4.2.

4.4.1.1 Flow Establishment SWRL Rule Evaluation

We consider the case of the SWRL rule that handles the flow establishment between two ERCMS participants. We are studying scalability and limits of our model-driven approach implementation. We study the influence over the execution time (see Figure 4.17) of the architecture description graph size. This corresponds to the application of the SWRL rule shown in Table 4.1.The implementation performs well and execution time is under 0.3 second for a 10 000 vertex graph (i.e. a system composed of 10 000 participants).

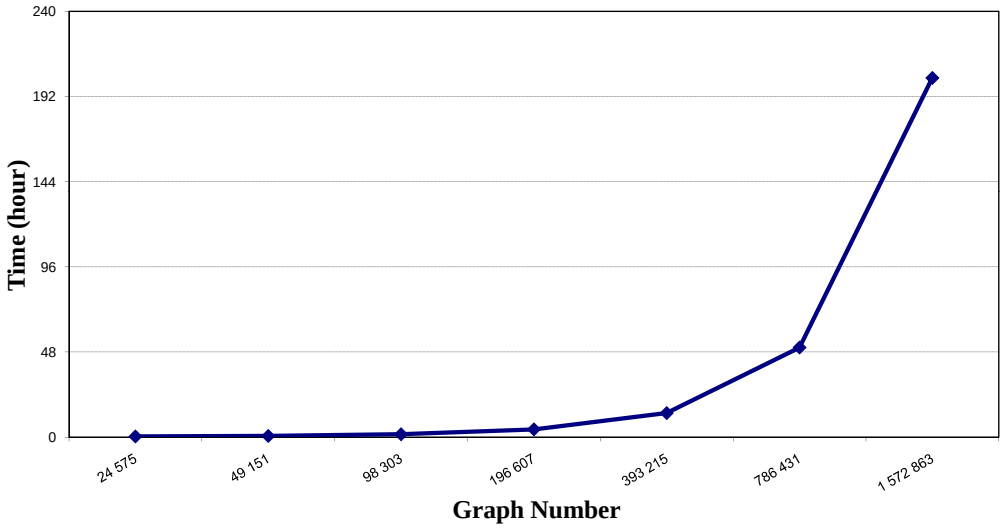


Figure 4.18: Execution Time of Applying $p1$, $p2$, $p3$ and $p4$ of $GG_{COLLAB \rightarrow EBC}$ wrt Number of Generated Graphs

4.4.1.2 Experimentation of Adapting Architecture to Group Membership Changes: Adding a New Investigators

The results, presented here, were obtained for the reconfiguration rules executed and triggered by the SWRL of Table 4.3. This modifies the mission ontology which triggers a refinement and the generation of the collaboration graph and its refinement into a set of EBC Descriptors (middleware graphs). We generate the first N members of this graph grammar corresponding to the N first consistent instances of the architecture. This corresponds to applying productions $p1$, $p2$ and $p3$ of $GG_{COLLAB \rightarrow EBC}$ (see Table 3.5) whenever it is possible to obtain all consistent architecture graphs. The application of production $p4$ ensures the elimination of unnecessary nodes in the graphs. The production applications include finding all graph matchings and transforming the architecture graph by adding nodes and edges. The productions $p2$ and $p3$ add one node and two edges. The production $p1$ adds two nodes and two edges. The system takes about nine hours to calculate the 400 000 first consistent instances and more than 1 500 000 consistent instances are generated in about 194 hours. These values remain acceptable for design-time purposes. Using a unique grid node, the system generates almost the 400 000 first consistent instances in about 140 hours.

4.4.2 Evaluation of Adapting Architecture to Resources Changes

In this subsection, we show simulation results of executing adaptation to resources context changes that are due to evolution on the mission or due to the constraints of the environment (e.g fire) or the nodes (e.g not enough energy). We simulate the case of communication degradation due to mission needs.

4.4.2.1 Connectivity Adaptation and QoS Preservation

We consider here the scenario explained in Figure 4.9 related to group3 (see Figure 4.4). This group is composed of three robots (Robot4, Robot6 and Robot3) with their robot coordinator (Coordinator2).

4.4.2.1.1 Simulation parameters For the simulation, we consider that Robot4 sends to Robot6 a video flow continuously using MEGP2 protocol. We consider X, as a random variable that represents a Gaussian law with parameters m and σ which characterizes the noise variation on the links. The variation is related to the distance between nodes. Figure 4.19 gives first results on the observation of one flow (between Robot4 and Robot6). The metric observed to evaluate the connectivity and the quality of the existing flow is the number of lost packets (packet of data that never reaches its destination).

4.4.2.1.2 Results and interpretations In Figure 4.19, the number of lost packets increases slowly in Step 0. One minute after the beginning of the simulation, Step 1 begins and Robot6 moves to find an injured person in an area near its position. The number of lost packets increases rapidly and reaches the threshold. The process of adaptation is triggered here (see the highest point of the curve in Figure 4.19) and Coordinator2 moves to avoid loosing the connectivity. Step 2 starts and the number of lost packets decreases at the beginning. However, the number of lost packets still beyond the comfort threshold for given time (one minute for our simulation) and an additional adaptation action has to be executed. Robot3 has to moves to serve as a communication relay. It comes near to Coordinator2 and Robot6. This action is required to enhance the quality of communication. Step 3 starts and the number of lost packets decreases. Robot6 arrives at the required position while Coordinator2 and

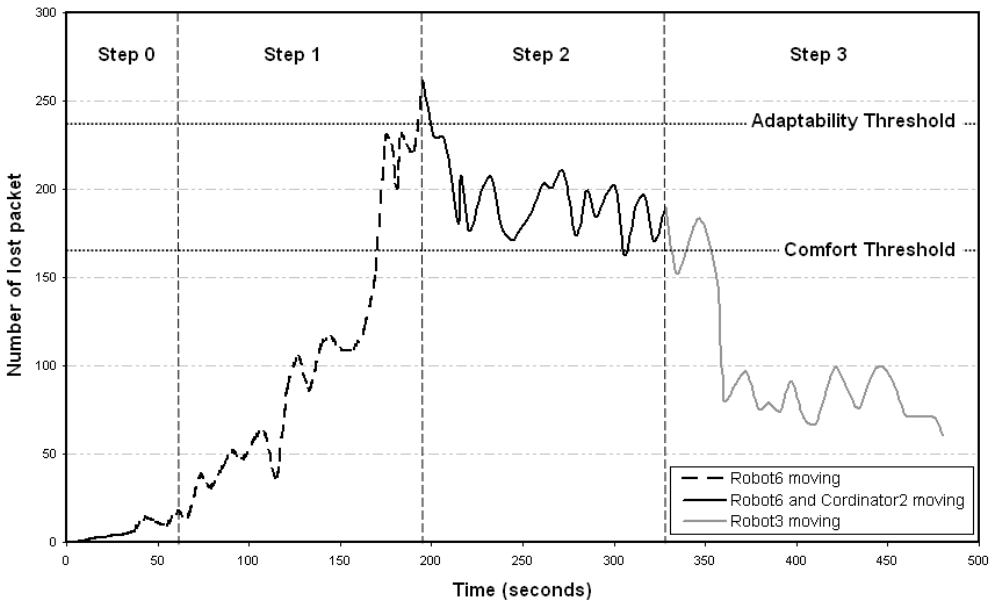


Figure 4.19: Simulation Results of the Adaptation for Connectivity and QoS Preservation

Robot3 maintain the communication between Robot4 and Robot6. There is still packet loses, but in an acceptable level.

4.4.2.2 Energy Evolution

We simulate here the behavior of an investigator that represents the ERCMS participants. We study the impact of the communication buffer storing the outcoming/incoming for this kind of node.

4.4.2.2.1 Simulation parameters We focus here, on the energy evolution for each node. We consider two random variables: X that represents a Poisson law of parameter λ, which characterizes the node's message production and, Y that represents a Gaussian law parameters m and σ, which characterizes the bandwidth variation on the links. For the memory state, we use a trace of a mobile node previously logged while taking into account the variable X. We calculate the node's energy according to the node state (consuming or producing, active, and idle state).

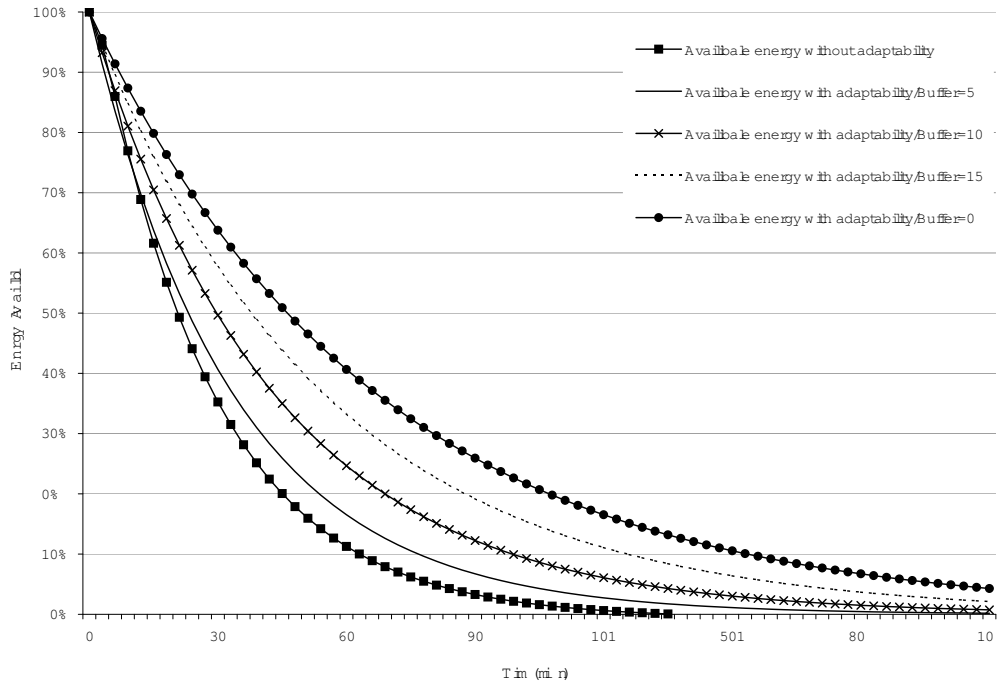


Figure 4.20: Energy Evolution on Nodes

4.4.2.2.2 Results and interpretations We show in Figure 4.20 the curves of node energy evolution during time with and without adaptability. We plot the curves that show the energy evolution on the investigator with and without adaptability that represent the general evolution of the energy on each node. We estimate how much time the node stays "alive" or has enough energy to work properly. We show here the impact of the communication buffer of the communication entities on the nodes. We notice that the use of adaptability maintains the node running more time with any buffer size (buffer size (Mb) = 0, 5, 10, 15, 20). The higher the buffer size is the longer the node stays running. If we focus on the 10% threshold the energy available, we notice that without adaptability the node reaches this threshold after 63 minutes. Using adaptability the node reaches this threshold (10% energy available) later than that. In Figure 4.20, the 10% threshold of the energy left is reached after 80 minutes (resp. 97, 130, 150) using a buffer size equal to 0 (resp. 10, 15, 20).

For this investigator, we notice that adaptability provides more energy due to our refinement and selection policies that discharge participants when their resources are decreasing.

4.5 Conclusion

In this chapter, We have presented present the graph matching and transformation engine that we developed to implement the graph grammars approach.

We have presented also two application scenarios. In the ERCMS scenario, mobile actors collaborate to manage emergency situations such as those occurring during disasters of forest fires. We propose three ontologies to model different aspects related to communication as the activity, collaboration and context. Adaptation actions are also proposed for adaptation in communicating architecture. These actions deal with different changing context parameters such as resource constraints and architecture evolution requirements. We presented refinement and selection procedures as well as examples of external context and of resources context changes adaptations. These examples show adaptation situations which are due to evolution of the application's environment. In the FoodShop scenario, we studied the dynamic reconfiguration of Service Oriented Architectures for maintaining the Quality of Service in perturbation-prone environments. Our approach uses graph grammar theories to implement rules that characterize the set of the different configurations candidate to solve composite or monolithic reconfiguration. Contrarily to enumerative approaches that define extensively the different equivalent and valid configurations, our approach is more appropriate for handling the scalablility.

To validate our approach, we conducted an experimental evaluation using our graph matching and transformation engine, GMTE, and the SWRL rule execution engine, Jess. Different situations show the effectiveness and the tractability of our approach. Our results show that ERCMS participant's resources remain in service as required for mission achievement thanks to our adaptation approach. Situations of disconnection from the mission groups are highly minimized.

Chapter 5

Conclusion

The adaptation of software architectures to support collaborative communication within a dynamically changing context has been addressed in this work. the study includes the automated deployment and management of service-oriented component-based architectures; including reconfiguration description and handling at different levels of interaction. We elaborated an adaptive multi-level modelling approach for reconfigurable software architectures in order to address this problem. Our approach consistently handles inter-level adaptation actions by considering a lower level architecture as a refinement of an upper level architecture. Firstly, we provided a generic level model based on ontologies as a description formalism, and SWRL for refinement rules description both for the application and collaboration layers. Secondly, we developed a graph-based description, and a rule-oriented technique based on graph grammars, for characterization, refinement and selection solution necessary for reconfigurable software architectures.

Adapting the architecture to constraint changes at the lower layers by switching among these multiple architectural configurations keeps the upper layer architectural configuration unchanged.

This adaptation requires two different actions: *refinement* for deriving low level candidate configurations and *selection* for identifying the optimal architectural configuration among all possible architectural configurations at a given layer.

Mastering the complexity of the design of such systems requires an appropriate description of its structural properties that will drive the adaptive deployment process at the different levels of interaction and within the different deployment nodes.

The adaptation process handles the automatic change of the system's architecture, proactively or reactively, with respect to the observed events during run-time. For instance, architectural components may be deployed, connected, and undeployed or disconnected. The events that lead to applying the adaptation actions are triggered when changes in the application context occur. We divided the context into external context, e.g. user preferences, user presence and position, priority of communications, etc.; and execution resource context, e.g. battery level, CPU load, available memory of end user devices, available bandwidth etc.

Knowledge representation by ontologies, is used to detect potential collaboration situations and to decide when group-wide sessions have to be created, adapted, etc. Thus, we show that the use of semantic models in order to support adaptation constitutes an interesting research direction. Adaptation policies are defined by means of SWRL rules. Such rules allow associating the suitable adaptation transformations to the external context changes. Jess engine is used to process the defined SWRL rules.

We implemented a graph rewriting system with expressive graphs and rewriting rules. Compared to the literature, our notations are more expressive combining the edNCE and DPO approaches, and allowing the consideration of both positive and negative matching conditions. Experimental studies are conducted to evaluate the performance of the related implemented tool (GMTE) and show their efficiency.

We elaborated rules that handle architectural mappings between collaboration layer and middleware layer. Using such a rule-based approach allows correct architectural reconfigurations to handle the architectural adaptation. Our approach has been successfully illustrated for collaborative group communication and applied for Emergency Response and Crisis Management Systems. On base of a graph transformation engine, we have simulated our rules with successful scalability tests. The scalability study provided has been conducted to assess the tractability of our approach when dealing with large scale group applications.

According to this work, many issues still open and need further investigation. Inclusion of additional adaptation levels in the proposed framewok is needed. The defined architectural and behavioral adaptation models are used to determine the best deployment configuration for a given activity related goal. These models consider adaptation at the application, collaboration, and middleware levels. Moreover, the transport level adaptation can by considered by dynamic micro-protocol assembly. To fill the gap between the middleware level and the network level, refining Event Based Communi-

cations entities into transport level entities is needed. Extending refinement procedure and graph grammars should be a solution. Graph grammar extensions can be done also to handle middleware layer abstractions instead of EBC. Context awareness is an important aspect of adaptive communicating systems. In case of environmental change (evolution in the application or resources change), "adaptation triggers" play an essential role in terms of notifying the decision maker about initiating adaptation actions. The issues at hand here are how and when triggers need to be activated. Moreover, any dramatic change in the situation may not be known in advance. The challenges here are: How to take into account past trends when predicting future decisions, and how to handle the external forces causing additional loads over the network and the deployment nodes.

Adaptive systems implementing decision distribution are more likely to ensure robustness and scalability. Distributing intelligence is not easy, especially in the case of adaptation, which needs special attention. A distributed decision should feature high availability requirements and always be available to application entities. Future work will include the definition of richer selection policies (considering further architectural characteristics). In addition, assigning priorities to flows as well as monitoring resource deficiencies present interesting challenges. Non-Functional properties, such as resilience and on-demand decision to handle priority requests have to be considered. Non-cooperative situations like QoS and performance degradation and connectivity loss are topics of research worth considering.

We are interested in supporting collaborative activities requiring dynamic implicit group-wide collaborative sessions. Such sessions are initiated spontaneously without user intervention and use potential collaboration situations. Therefore, spontaneous sessions have to be detected, established, and then adapted to context changes. The design and implementation of mechanisms enabling the spontaneous setup of implicit sessions need to be considered.

Chapter 6

Extended French Abstract/Résumé

6.1 Introduction

Les nouvaux environnements communicants combinent plusieurs caractéristiques complexes telles que la mobilité, les interactions ad-hoc, l'hétérogénéité des dispositifs, les communications sans fil, etc. qui posent de nouveaux défis aux systèmes de communication chargés d'assurer le transport des informations échangées.

Dans ce contexte, nous nous intéressons plus précisément aux activités menées par des utilisateurs organisés en groupes. Ces activités, appelées aussi activités collaboratives, peuvent bénéficier des opportunités offertes par les environnements communicants en les exploitant intelligemment. Par exemple, dans les systèmes collaboratifs classiques, ce sont les utilisateurs qui doivent indiquer explicitement le fait qu'ils veulent initier une session, inviter des participants, etc. Dans un environnement communicant, le système peut reconnaître les situations de collaboration potentielles (implicites) et mettre en œuvre des sessions spontanées pour soutenir la collaboration entre les participants.

Le concept de *contexte* est très important pour les systèmes communicants collaboratifs (SCC). Le contexte peut être divisé en deux parties relative d'une part au *contexte externe* (par exemple la présence et la position des utilisateurs, leurs préférences, l'heure de la journée, etc.) et d'autre part au *contexte des ressources* (le niveau de la batterie des dispositifs, la quantité de mémoire libre, le débit des communications, etc.).

Une fois que le contexte est capturé, le système doit répondre aux changements détectés. Ceci traduit une *adaptation au contexte*. Par exemple, si un nouvel utilisateur

arrive dans un groupe, l'architecture du système logiciel qui soutient la collaboration doit changer ; par exemple des composants devront être automatiquement déployés sur la machine de cet utilisateur afin qu'il puisse envoyer et recevoir des données.

Dans ce contexte, notre travail vise à gérer dynamiquement l'adaptation des architectures des applications logicielles distribuées des environnements communicants collaboratifs. Pour ce type d'applications, il est nécessaire de pouvoir modifier dynamiquement leur architecture en créant des composants, en les interconnectant, ou en les supprimant, et ce pendant l'exécution des applications. Cette dynamicité répond à des contraintes liées à des variations de capacités de communication, de calcul et d'énergie, ainsi que des évolutions dans la nature des activités qu'elle soutiennent.

Dans ce mémoire nous proposons une approche orientée-modèle pour la reconfiguration dynamique des applications logicielles. Cette approche est basée sur des techniques orientées-règles. Les règles permettent des actions de transformation élémentaires relatives, par exemple, aux composants logiciels qu'il faut introduire, supprimer, activer ou désactiver, ou aux liens d'interdépendances (entre composants) à introduire ou à supprimer. L'approche de modélisation que nous définissons dans ce manuscrit, couvre plusieurs maillons de la chaîne de description architecturale. Les instances des architectures sont décrites par des graphes étendus où les composants sont représentés par des nœuds et les interdépendances (par exemple les connexions entre les nœuds) sont décrites par des arcs. Les styles architecturaux, permettant de décrire toutes les instances consistantes d'une application à architecture dynamique, sont spécifiés par des grammaires de graphes étendues. Les extensions concernent, par exemple, l'introduction de variables typées dans les productions de grammaires et la notion de restrictions à l'applicabilité de ces productions. Cette approche de modélisation traite aussi la description des contraintes et des propriétés architecturales d'une application. Ce type de contraintes est spécifié par l'absence ou la présence de motifs dans les graphes décrivant l'architecture. Leur vérification automatique est réalisée à l'aide d'algorithmes de recherche d'homomorphismes. Un autre intérêt de l'approche de description concerne la possibilité d'extraire des paramètres et des propriétés architecturales (par exemple le nombre maximal de services élémentaires composant un service composite, une structure de l'orchestration ou des interdépendances entre composants). Dans ce manuscrit, nous traitons de manière détaillée un cas d'étude relatifs aux opérations d'intervention d'urgence. Dans le cadre de ce cas d'étude, nous considérons trois niveaux d'adaptation architectural relatifs aux niveaux application, collaboration et middleware, que nous

caractérisons ci-après. Des raffinements sont définis pour permettre de transformer des descriptions d'architectures considérées à un niveau donné vers les descriptions correspondantes dans un autre niveau. Nous présentons également une implantation qui suit cette approche et qui peut être utilisée par les concepteurs de SCC comme base pour l'exécution des applications.

Outre l'état de l'art, le mémoire est structuré en trois parties principales que nous résumons ci-après. La première partie (résumée en section 7.2) décrit l'approche proposée pour la modélisation des architectures logicielles. La deuxième partie (résumée en sections 7.3 et 7.4) décrit l'application de la méthodologie proposée au contexte des systèmes communicants, avec pour cas d'étude les opérations d'interventions d'urgence. Enfin, les résultats expérimentaux de la dernière partie de la thèse sont résumés en section 7.5. Les conclusions et perspectives de nos travaux sont résumés en section 7.6.

6.2 Approche générique de modélisation

Dans cette section, nous présentons une approche générique pour la modélisation des architectures logicielles. Cette approche introduit plusieurs niveaux d'abstraction dans le but de séparer les différentes problématiques abordées. La complexité est ainsi réduite, ce qui permet une conception du système plus claire et plus cohérente. Nous appliquerons par la suite cette approche au cas des systèmes communicants collaboratifs.

6.2.1 Modélisation Architecturale Multi-niveau

Dans notre approche, les modèles manipulés représentent des configurations architecturales, c'est-à-dire, des ensembles d'entités logicielles distribuées sur un ou plusieurs nœuds d'exécution et reliées entre elles par des liens de communication. Une configuration architecturale (ou plus simplement *architecture*) est dénotée $A_{n,i}$, où n est le niveau d'abstraction considéré ($n \in \mathbb{N}$) et i est un index permettant de distinguer les configurations ($i \in \mathbb{N}$). Pour chaque configuration architecturale $A_{n,i}$ de niveau n, il existe plusieurs configurations architecturales $(A_{n-1,1}, \ldots, A_{n-1,p})$ qui l'*implantent* au niveau $n-1$, c'est-à-qui représentent un raffinement ou une vue détaillée de $A_{n,i}$ (voir Figure 6.1).

En général, les architectures des niveaux supérieurs sont déterminées par les exigences de haut niveau de l'application considérée, tandis que les implantations possi-

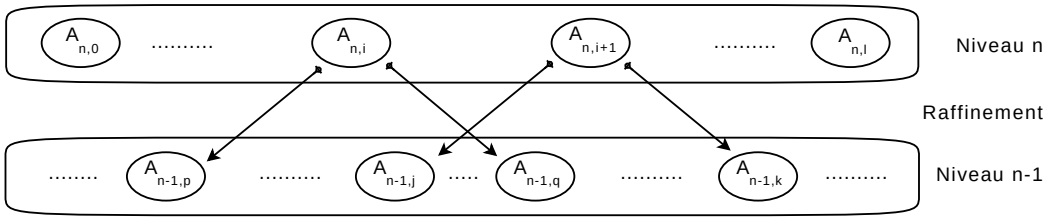


Figure 6.1: Approche de modélisation multi-niveau.

bles aux niveaux bas sont contraintes par les ressources disponibles. L'adaptation du système à ces exigences et à ces contraintes se fait donc par le biais des configurations architecturales retenues à chaque niveau. Quand l'adaptation impose un changement au niveau n, un modèle est choisi parmi les modèles du niveau $n-1$ qui implantent le nouveau modèle de niveau n. Si l'ancienne et la nouvelle configurations au niveau n ont des implantations communes au niveau $n-1$, cela veut dire que la configuration de niveau $n-1$ actuelle implante aussi la nouvelle configuration de niveau n, et en conséquence le processus d'adaptation s'arrête là. Sinon, il faut continuer à choisir des implantations dans les niveaux $n-2$, $n-3$, ..., 0. La procédure générique de raffinement permet ce raffinement.

```
1 Refine()
2 {
3 Soit 𝔸_n l'ensemble de configurations architecturales au niveau n
4 Soit 𝔸_{n-1} l'ensemble de configurations architecturales au niveau n − 1
6 Soit A_{n,i} ∈ 𝔸_n, i ∈ ℕ une configuration architecturale donnée
7 Calculer 𝔸^i_{n-1} = {A_{n-1,j} ∈ 𝔸_{n-1} tel que A_{n-1,j} implante A_{n,i}, j ∈ ℕ}
8 }
```

Table 6.1: Procédure générique de raffinement

L'architecture du niveau le plus bas représente la distribution physique des composants sur les machines peut donc être utilisée pour effectuer un déploiement automatique de ces composants. Ce déploiement automatique est réalisé par un *service de déploiement* qui prend en entrée l'architecture de niveau 0 et qui effectue les déploiements nécessaires pour mettre en œuvre la configuration y décrite. Les actions de déploiement de composants sont basées sur la technologie OSGi [OSG07].

La technique d'adaptation que nous avons décrite a besoin de deux actions différentes : le *raffinement* et la *sélection*. Ces deux actions sont expliquées dans la section suivante.

```
1 Select(Policy)
2 {
3 Soit A_{n,p} ∈ 𝔸_n , p ∈ ℕ
4 Soit C représentant les attributs contextuels
                (i.e.  énergie et mémoire disponible)
5 Sélectionner S_1 = {A_{n-1,k} ∈ 𝔸^p_{n-1}, k ∈ ℕ tel que
     Context_Adaptation(A_{n-1,k}, C) ≥ Context_Adaptation(X, C), ∀X ∈ 𝔸^p_{n-1}}
6 if card(S_1) ≠ 1
7   si Policy = Dispersion
8      Sélectionner S_2 = {A_{n-1,k} ∈ S_1 , k ∈ ℕ tel que
       Dispersion(A_{n-1,k}) ≥ Dispersion(X), ∀X ∈ S_1}
9   si Policy = Distance
10     Soit A_{n,p} et A_{n,q} ∈ 𝔸_n , p, q ∈ ℕ
11     Soit A_{n-1,p} l'implantation courante au niveau n − 1 de A_{n,p}
12     Sélectionner S_2 = {A_{n-1,k} ∈ 𝔸^q_{n-1}, k ∈ ℕ tel que
13     Relative_Cost(A_{n-1,p}, A_{n-1,k}) ≤ Relative_Cost(A_{n-1,p}, X), ∀X ∈ S_1}
14  si card(S_2) ≠ 1
15     Sélectionner une configuration architecturale parmi S_2
16 }
```

Table 6.2: Procédure générique de sélection.

6.2.2 Procédures génériques de Raffinement et de Sélection

Le *raffinement* permet de calculer, à partir d'un modèle de niveau n, l'ensemble de modèles de niveau $n-1$ qui implantent le modèle donné. Nous considérons la procédure générique de raffinement `Refine()`, présentée dans la Table 6.1. Cette procédure est générique du fait que le mot *implante*, dans la ligne 7, aura une signification différente pour chaque niveau d'architecture. Dans un niveau, une fois que tous les modèles possibles pour ce niveau ont été obtenus, il est nécessaire de sélectionner le modèle retenu. Le modèle choisi représente l'architecture qui sera effectivement déployée. Nous appelons ceci le processus de *sélection*. Nous considérons la procédure générique de sélection, `Select()`, présentée dans la Table 6.2.

Cette procédure utilise, en premier lieu, les données disponibles sur le contexte d'exécution des machines (par exemple la quantité de mémoire disponible ou le niveau de batterie des dispositifs) pour éliminer les configurations qui sont incompatibles avec ces contraintes. Pour cela, elle utilise la procédure `Context_adaptation()`, détaillée dans Table 6.3, qui affecte une valeur à chaque architecture. Plus une architecture est adaptée au contexte actuel, plus sa valeur sera grande. Les architectures incompatibles avec le contexte actuel recevront une valeur de -1, ce qui veut dire qu'elles ne pourront pas être retenues pour le déploiement.

Si plusieurs architectures ont la même valeur maximale, alors la procédure `Select()` utilise une politique plus fine pour déterminer la meilleure architecture. Cette politique peut s'appuyer, par exemple, sur l'étendue de la répartition des composants dans l'architecture (fonction `Dispersion()`) ou sur le nombre de redéploiements à faire si l'on part de la configuration actuelle pour arriver à la configuration considérée (fonction `Relative_Cost()`). Ces fonctions sont détaillées dans les Table 6.4 et Table 6.5 dans le cadre de l'application de l'approche au cas des systèmes communicants collaboratifs. Notons que la procédure de sélection peut être facilement modifiée pour considérer d'autres politiques.

6.3 Application au cas des Systèmes Communicants Collaboratifs

Dans cette section, l'approche générale de modélisation introduite dans la section précédente est appliquée à la modélisation des systèmes collaboratifs dans des environnements communicants (SCC). Le but est de spécifier les problèmes inhérents à ce type de systèmes et de proposer un cadre conceptuel pour leur conception. Pour cela, nous identifions un ensemble de niveaux pertinents pour lesquels nous proposons des modèles et des transformations.

La partie gauche de la Figure 6.2 illustre les niveaux retenus. Ces niveaux sont génériques et peuvent ainsi être utilisés (et implantés) de façon différente par des concepteurs de systèmes différents. Cependant, nous proposons une implantation de référence (actuellement en cours de développement) qui peut être utilisée comme cadre de conception et d'exécution pour des SCC. Nos choix pour les formalismes et les technologies utilisés dans les modèles et les transformations dans cette implantation sont illustrés

```
1  Context_Adaptation()
2  {
3  Soit A_{1,q} une configuration architecturale au niveau 1
4  Soit N = card(A_{1,q})
5  Soit Resource_1 ... Resource_R l'ensemble des ressources considérées
6  Soit R = card({Resource_1 ... Resource_R})
7  Soit node_i^q un nœud de déploiement i de A_{1,q}
8  Soit L_r^i le niveau de la ressource r pour node_i^q
9  Soit T_r le seuil associé à la ressource r
10 α_r^i  r ∈ [1..R] les poids associés à chaque ressource r pour node_i^q
11 Soit β_r  r ∈ [1..R] les poids associés à chaque ressource r pour A_{1,q}
12 Soit cadapt=0
13 Pour chaque i ∈ 1..N
14     pour chaque r ∈ 1..R
15          P_r^i = α_r^i L_r^i − T_r
16          si P_r ≤ 0 alors retourner -1
17     fin pour
18 fin pour
19 Pour chaque r ∈ 1..R
20     cadapt=cadapt+β_r min_i(P_r^i)
21 fin pour
22 retourner cadapt
23 }
```

Table 6.3: Procédure d'adaptation au contexte

dans la partie droite de la Figure 6.2. Les paragraphes suivants décrivent brièvement les niveaux retenus.

6.3.1 Niveau application

Le niveau supérieur est le niveau *application*. Ce niveau contient les éléments de haut niveau qui définissent le métier de l'application. Parmi ces éléments, on trouve les interfaces utilisateur, les modules de sécurité, les modèles de données métier, etc. Tous les éléments concernant la collaboration sont capturés dans le modèle architectural de ce niveau, $A_{3,i}$[1]. En conséquence, ce modèle est une vue métier des entités qui collaborent et des relations existantes entre elles.

[1]Bien sûr, la représentation d'autres éléments ne concernant pas la collaboration peut être envisagée.

```
1 Dispersion()
2 {
3 Soit A_{1,q} une configuration architecturale au niveau 1
4 Soit node_i^q un nœud de déploiement i de A_{1,q}
5 weight=0
6 Pour chaque node_i^q
7  si ∃CM déployé sur node_i^q alors weight=weight+1
8 retourner weight
9 }
```

Table 6.4: Procédure de calcul de la dispersion

```
1 Relative_Cost()
2 {
3 Let A_{1,q} et A_{1,k} deux configurations architecturales au niveau 1
4 Soit node_i^q(component_j) un nœud de déploiement de component_j ∈ A_{1,q}
5 rcost=0
6 Pour chaque component_j ∈ A_{1,q} ∪ A_{1,k}
7  si node_i^q(component_j) ≠ node_i^k(component_j) alors rcost=rcost+1
8 }
```

Table 6.5: Procédure de calcul du coût relatif

Dans notre implantation, le modèle de niveau application est une ontologie, décrite au moyen du langage OWL (le standard du web sémantique pour la description d'ontologies) [SWM04]. Nous avons fait ce choix car les ontologies sont un système de représentation des connaissances bien adapté à l'expression de concepts de haut niveau, proches des activités humaines, ce qui est le cas dans ce niveau. De plus, elles permettent de réaliser des inférences et des raisonnements, ce que nous exploitons pour faire le raffinement du niveau application vers le niveau suivant.

Nous considérons, comme communément fait dans les systèmes ontologiques, deux niveaux d'ontologies : une ontologie générique de la collaboration (qui décrit des sessions, des utilisateurs, des rôles, des flux de données, etc ; voir [STV08]) et des ontologies spécifiques à chaque application. Les ontologies spécifiques sont définies par les concepteurs des applications, tandis que l'ontologie générique est fournie dans notre implantation[2].

[2]Cette ontologie est disponible sur http://homepages.laas.fr/gsancho/ontologies/sessions.owl.

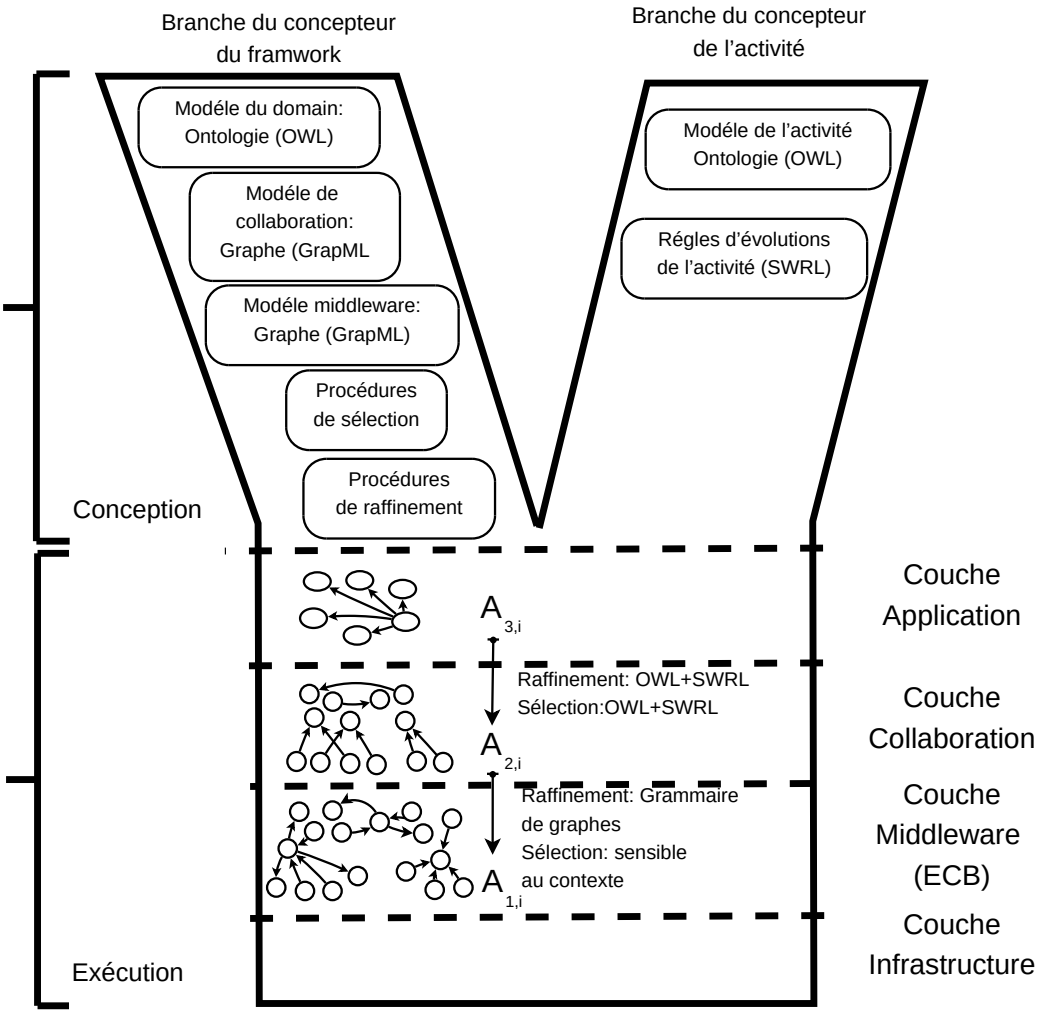


Figure 6.2: Cadre de modélisation pour les SCC et technologies d'implantation.

6.3.2 Niveau collaboration

Le niveau suivant est le niveau *collaboration*. Ce niveau représente des groupes d'utilisateurs organisés en sessions où ils communiquent avec des flux de données. Le modèle du niveau collaboration, $A_{2,i}$, est un graphe (un diagramme de collaboration dynamique ; voir [BDVT04]) qui contient la structure d'une ou de plusieurs sessions. Une *session* est un ensemble de *flux de données* (texte, video or audio). Chaque flux a comme origine et comme destination des *composants* logiciels différents. Les composants font partie d'un *outil*. Les composants sont déployés dans des *nœuds*, qui représentent les dispositifs des utilisateurs. Le graphe contenant ces éléments est exprimé avec le

langage GraphML, un dialecte de XML pour la représentation de graphes.

6.3.3 Niveau middleware

Le niveau suivant est le niveau *middleware*. Ce niveau supporte un modèle de communication qui fait abstraction des détails de bas niveau (tel que les sockets TCP, les adresses IP, la diffusion, etc). Ce niveau peut être basé, par exemple, sur des paradigmes connus tels que le pair-à-pair, les appels procéduraux à distance, CORBA, etc. Dans notre cas, le paradigme retenu est la communication orientée événement (*Event-Based Communications* ou EBC, [MC02]) ; les éléments représentés dans ce modèle sont ainsi des producteurs d'événements (EP), des consommateurs d'événements (EC) et des *channel managers* (CM), ainsi que les liens entre eux. Le modèle correspondant à ce niveau est un graphe représentant ces différents éléments et il est également exprimé avec le langage GraphML.

6.3.4 Niveau infrastructure

Le dernier niveau est le niveau *infrastructure*. Ce niveau représente tous les éléments logiciels et matériels qui sont nécessaires pour l'exécution d'applications dans les dispositifs. Par exemple, on y trouve le système d'exploitation, les pilotes, la pile TCP/IP, les capteurs, etc. Nous faisons l'hypothèse que ces éléments sont disponibles et qu'ils ont été correctement configurés ; il n'y a donc pas de modèle associé à ce niveau.

6.3.5 Implantation du raffinement et de la sélection

Les trois premiers niveaux de modélisation des SCC induisent deux opérations de raffinement et de sélection pour réaliser les transformations d'un niveau vers le niveau inférieur.

Puisque les modèles de niveau application sont exprimés avec des ontologies, nous avons retenu l'exécution de règles SWRL [HPSB+04b] pour le raffinement d'un modèle de niveau application vers un modèle de niveau collaboration. En effet, ces règles s'appliquent sur les éléments d'une ontologie et peuvent les modifier, en créer d'autres, etc. De cette façon, en fonction des instances des concepts de niveau métier exprimés dans l'ontologie, les règles extrairont le schéma de collaboration implicitement contenu dans cette ontologie. Ce schéma résultant sera exprimé selon les concepts de l'ontologie

générique de la collaboration. Puisque l'ontologie générique de la collaboration et les modèles de niveau collaboration contiennent des éléments équivalents, la traduction de l'ontologie en un graphe GraphML est une tâche relativement simple et automatique car les deux langages sont des dialectes de XML. L'application d'un ensemble de règles SWRL sur une ontologie de niveau application produit un seul modèle de niveau collaboration. Le processus de sélection n'est donc pas nécessaire pour cette première transformation.

Puisque les modèles de niveau collaboration et de niveau middleware sont des graphes, la technique de *transformation de graphes* est adaptée pour réaliser le raffinement entre ces deux niveaux. Plus concrètement, nous utilisons des grammaires de graphes génératives [Cho56].

Nous nous appuyions sur les graphes abstraits de composants ou ACG en anglais. L'approche ACG est une structure marquée et générique que nous employons pour définir les graphes d'architecture (ACG totalement instanciés), et les graphes de règles (ACG partiellement instanciés). Les graphes décrivent une architecture comme un ensemble de composants associés aux nœuds du graphe et un ensemble d'arcs dénotant les relations d'interdépendance entre ces composants.

Dans une structure ACG, les nœuds décrivent des composants logiciels et sont marqués par les champs suivants : l'identifiant de composant, le type de composant, la session, le type des données traitées et la localisation du composant (par exemple, la machine sur laquelle il est exécuté).

Nous distinguons deux catégories de nœuds : les nœuds de règles et les nœuds de graphes. La différence entre ces deux types de nœuds est que le premier est une abstraction d'un type de composants et peut avoir des champs variables, alors que le second correspond à un composant instancié de l'architecture et ne peut donc avoir que des champs totalement instanciés.

Nous avons défini une grammaire de graphes dont les nœuds non-terminaux sont les éléments du graphe de collaboration et les nœuds terminaux sont des entités EBC. Ainsi, un graphe de niveau collaboration produit un ensemble de graphes de niveau middleware qui l'implantent. Ceci est réalisé à travers la procédure de raffinement basée sur la grammaire de graphes présentée dans la Table 6.6 utilisant les production de la grammaire de la Table 3.5.

Une architecture est décrite par un graphe enrichi permettant d'intégrer les différents paramètres de ses composants et leurs interdépendances. Nous utilisons le concept de

Soit $\mathbb{A}^i_{n-1}$ l'ensemble des configurations architecturales du niveau n qui raffinent $A_{n,i}$ $\forall A_{n-1,j} \in \mathbb{A}^i_{n-1}$, $\exists p_1, \ldots p_m \in GG$ tel que $A_{n,i} \xrightarrow{p_1,\ldots,p_m} A_{n-1,j}$

Table 6.6: Procédure de raffinement basée grammaire de graphes

style architectural pour décrire les architectures dynamiques et la spécification de leurs instances consistantes. Un style architectural est caractérisé par un modèle basé sur des grammaires de graphes étendues. Nous utilisons une approche formelle permettant de décrire les systèmes de transformation pour traduire une description définie à un niveau d'abstraction donné vers un autre niveau d'abstraction. Cette approche se base aussi sur les grammaires de graphes. Nous intégrons dans ce formalisme des mécanismes de type NCE. Enfin, nous présentons une approche de modélisation permettant de caractériser les événements et les règles ainsi que les protocoles de reconfiguration des architectures. Nous utiliserons pour la caractérisation des règles de reconfiguration des règles de transformation de graphes considérant des conditions d'application négatives et des instructions de connexion de type NCE.

Notre implantation s'appuie pour cela sur un moteur de transformation de graphes[3]. La procédure générique de sélection est utilisée ici pour choisir l'architecture optimale à retenir pour déploiement. Il suffit de faire des choix pour les fonctions `Context_Adaptation()`, `Dispersion()` et `Relative_Cost()`.

6.4 Cas d'étude

Afin de démontrer la faisabilité de l'approche, nous avons pris le choix de présenter un traitement détaillé du cas d'étude des opérations d'intervention d'urgence (OUI). Au travers de ce cas d'étude, nous avons considéré des exigences d'adaptabilité dues aux changements du contexte d'exécution, à la panne des composants et à des contraintes liées au provisionnement de la QdS (par exemple le taux de perte de messages).

Les OIU impliquent des groupes structurés de robots ou de personnels militaires coopèrant pour la réalisation d'une mission commune. Les éléments de l'architecture possèdent différents rôles et disposent de ressources inégales en capacités de commu-

[3]Disponible sur http://homepages.laas.fr/khalil/GMTE.

nication, en CPU et en énergie. Ils sont déployés sur des machines fixes et mobiles et communiquent via des réseaux filaires et sans fil. L'activité comporte deux phases d'exécution correspondant à une phase d'exploration du champ d'investigation et à une phase d'action faisant suite à la découverte d'une situation critique. Les rôles dans l'application et la structure des interactions entre participants évoluent d'une étape à l'autre pour s'adapter à l'évolution des objectifs applicatifs et du contexte d'exécution. Trois niveaux d'abstraction sont identifiés en qui font référence à trois couches protocolaires : application, collaboration et middleware. Cependant, le principe d'indépendance des couches OSI est remis en cause en autorisant la prise en compte, au niveau considéré, d'informations dont la sémantique est d'un niveau différent; c'est le principe de base du cross layering où une réorganisation affectant un niveau de l'architecture peut se traduire par une réorganisation à un autre niveau architectural. Les composants logiciels, qu'ils relèvent des couches application, collaboration ou middleware, auront à prendre en compte des besoins multiples et évolutifs dans le temps liés à l'activité ciblée, à la mobilité des utilisateurs, aux flux de données échangés (audio, vidéo et texte) et aux contraintes de l'environnement de communication. De plus, l'évolution de la mission induira inévitablement des changements dans la hiérarchie et dans la structure des coopérations entre intervenants, par exemple, suite aux informations acquises par les investigateurs sur le terrain ou suite aux décisions du superviseur et des coordinateurs de la mission. La collaboration est basée sur l'échange de données entre participants, notamment des données d'observation et des données d'analyse, produites périodiquement ou immédiatement après un événement particulier. Une équipe d'intervention d'urgence est ainsi constituée de participants ayant différents rôles : un superviseur de la mission, plusieurs coordinateurs, et plusieurs sections d'investigateurs. Le superviseur de la mission gère l'ensemble des coordinateurs et chaque coordinateur dirige une section d'investigateurs. À chaque rôle correspondent les fonctions suivantes :

- Un superviseur a pour fonction de diriger et d'autoriser les actions qui sont déléguées aux coordinateurs. Le superviseur est l'entité qui supervise toute l'application, il attend des rapports de tous ses coordinateurs qui synthétisent le contexte courant de l'application et l'informent du déroulement de la mission. Le coordinateur est déployé sur une machine fixe, il dispose d'un accès à l'énergie permanent et de capacités de communication et de CPU conséquentes.

- Selon les actions et les objectifs assignés par le superviseur, un coordinateur doit diriger ses investigateurs en leur assignant des tâches à exécuter. Il doit également collecter, interpréter et synthétiser les informations reçues des investigateurs et les diffuser vers le superviseur. Les coordinateurs sont déployés sur des machines mobiles.

- Les investigateurs ont pour fonction d'explorer le champ opérationnel, d'observer, d'analyser et de faire un rapport décrivant la situation aux coordinateurs qui les contrôlent. Ils sont déployés sur des machines mobiles et disposent, donc, de ressources limitées en énergie et en CPU.

Les fonctions assignées aux participants impliquent d'observer (D) le champ d'investigation et de rapporter (P) sur ce qui est observé. Les données de retour D sont des données descriptives tandis que les données de retour P sont des données produites et expriment l'analyse de la situation par un investigateur ou un coordinateur. Le superviseur supervise l'ensemble de la mission, en décidant des actions à exécuter en fonction des objectifs opérationnels et de l'analyse des retours P transmis par les coordinateurs. Il possède sous ses ordres, au moins un coordinateur, et chaque coordinateur possède au moins un investigateur. Un coordinateur est en charge de la partie de la mission qui lui a été assignée par le superviseur. Il décide localement des actions à exécuter en fonction de l'observation et de l'analyse des données D transmises par les investigateurs. Pour prendre cette décision, le coordinateur peut également utiliser les données P transmises par les investigateurs. Les coordinateurs rapportent l'évolution de la sous mission au superviseur en utilisant des données de retour de type P.

Pour la modélisation de l'activité OIU nous avons défini une ontologie. Le concept principal de cette ontologie est le concept `Participant` qui a plusieurs propriétés. Les diffèrent types de participant (`Supervisor`, `Coordinator`, et `Investigator`) sont modélisé en tant que sous-concepts dont chacun possède propriétés additionnelles. Un `Participant` appartient a un `Group` qui est géré par un `Manager`. Un `Manager` peut être un `Supervisor` qui gère un `CoordinatorGroup` ou un `Coordinator` qui gère un `InvestigatorGroup`. L'autre concept important de cette ontologie est le concept `Entity`. En effet, le concept `Participant` hérite du concept `Entity` qui peut être soit `Artificial` soit `Human`. Plusieurs participants humains sont représentés par les concepts : `Fireman`, `Pilot`, ou `Walker`. Plusieurs participants artificiels sont représenté par les concepts : correspondant a des robots ou des véhicules (`AmphibiousRobot`, `Drone`,

et `GroundRobot`).

Cette ontologie est liée à l'ontologie de collaboration à travers les concepts `Flow` et `Entity` qui correspondent aux concepts de collaboration : `CommunicationFlow` et `Node`. Nous avons utilisé l'ontologie pour décrire le contexte qui est liée à ontologie de l'activité OIU à travers le concept `Participant`. Nous présentons un raffinement complet pour la phase initiale de l'OIU. Nous avons expliqué le raffinement et les procédures de sélection ainsi que des exemples présentant des adaptations a l'évolution du contexte externe et des ressources de communication et d'exécution. Ces exemples montrent des situations d'adaptation qui sont dues à l'évolution de l'environnement de l'activité.

6.5 Expérimentations et évaluations

Pour implanter notre approche, nous avons développé un moteur d'appariement de transformation de graphes (GMTE : Graph Matching and Transformation Engine). Le GMTE est un outil efficace que nous avons mis en œuvre en C++. Cet outil est basé sur une extension de l'algorithme de Messmer [Mes95]. Il est capable de rechercher des modèles de graphes petits et moyens dans des graphes de taille importante. Une analyse de la complexité de notre algorithme de calcul a été effectuée qui a montré sa performance. Nous avons également montré que, lorsque que les étiquettes sont considérées comme des constantes, cette complexité est similaire à la complexité de l'algorithme de Ullmann [Ull76]. Nous utilisons deux modèles de graphe : les graphes de règles et les graphes hôte qui sont des graphes marqués. Un graphe de règles peut être totalement ou partiellement instancié. L'unification est effectuée pour les étiquettes non instanciées.

L'outil peut être utilisé d'une manière non-interactive, en tant que une bibliothèque C++ qui fournit une API qui peut être invoqué à partir d'un programme C++ ou d'un programme Java. L'outil peut être utilisé via un exécutable C++ qui lit un graphe de règles et un graphe hôte à partir des fichiers TXT ou XML.

Le standard XML utilisé est GraphML (graphique Markup Language). GraphML est un format de fichier basé sur XML pour les graphes. Il se compose d'un noyau qui permet de décrire les propriétés structurelles d'un graphe et offre aussi un mécanisme d'extension flexible pour ajouter des données spécifiques à l'application. Contrairement à de nombreux autres formats de fichier pour la représentation des graphes, GraphML

n'utilise pas une syntaxe personnalisées. En effet, il est basé sur XML et donc idéal comme dénominateur commun pour tous les types de services de production, d'archivage ou de traitement de graphes.

Notre outil a été associé à une interface utilisateur composé notamment des zones et des éléments suivants:

- Une barre de menu offrant à l'utilisateur de nombreux éléments tels que la création, la suppression et la sauvegarde des projets, des graphes et des règles.
- Une barre d'outils que l'utilisateur peut utiliser pour éditer des graphes et des règles (sauvegarde, annuler, refaire ...).
- Un explorateur de projets donnant à l'utilisateur un arbre qui représente la liste des projets, des graphes et des règles ouverts.
- Un panneau de composants contenant une liste de boutons pour créer des noeuds et des liens.
- Une zone d'édition de graphes qui offre à l'utilisateur la possibilité d'ouvrir, d'afficher et d'éditer des graphes hôtes.
- Une zone d'édition de règles qui offre à l'utilisateur la possibilité d'ouvrir, d'afficher et d'éditer les règles.
- Une zone de visualisation des graphes transformés résultants des applications des règles aux graphes hôtes.

Par ailleurs, nous avons étudié la reconfiguration dynamique des architectures orientées services pour le maintien de la qualité de service dans les environnements communicants sensibles au contexte. Nous avons adopté un modèle d'architecture virtualisée qui peut faire face au problème de l'adaptation de la communication demandeur-fournisseur dans des situations différentes et à différents niveaux d'interaction. Notre approche utilise la théorie des grammaires de graphes pour mettre en œuvre des règles qui caractérisent l'ensemble des configurations candidates adaptées au contexte. Contrairement aux approches énumératives qui définissent l'ensemble des configurations valides, notre approche est plus appropriée pour le passage à l'échelle qui caractérise sans énumérer les configurations valides. Il peut gérer des applications Web dans une

vision du monde ouverte comme dans les activités de collaboration où le nombre de participants peut varier d'une manière incontrôlée et imprévisible.

Pour valider notre approche, nous avons effectué des expériences d'évaluation en utilisant notre moteur GMTE et le moteur de règles Jess. Nous avons également simulé également les attributs architecturaux quantitatifs. Pour ceci nous avons utilisé la grille de calcul *Grid'5000*. Les grilles informatiques sont des plate-formes de calcul à grande échelle, hétérogènes et distribuées. Le concept de grille informatique correspond à la réalisation de vastes réseaux mettant en commun des ressources informatiques géographiquement distantes. Les grilles de calcul permettront d'effectuer des calculs et des traitements de données à une échelle sans précèdent. Le concept de grille peut englober des architectures matérielles et logicielles très différentes, en fonction des objectifs recherchés. La plate-forme *Grid'5000* est une grille matérielle et logicielle, interconnectant à très haut débit une dizaine de clusters de PC de grandes tailles. Pour fixer un ordre de grandeur, chaque cluster peut comprendre 500 unités de calcul, d'où le total de 5000 qui donne le nom de code du projet *Grid'5000* [CCD+05].

Nous avons simulé différentes situations montrant l'efficacité de notre approche. Nos résultats montrent qu'un participant l'OIU a plus de ressources pour poursuivre sa mission grâce notre approche d'adaptation. Les situations de déconnexion de participant de la mission sont fortement réduites.

Nous avons évalué les temps d'exécution des règle SWRL notamment la règle de mise en place de flux (c.f. Figure 4.17), nous avons expérimenté le temps de calculs des reconfigurations architecturales notamment celles qui sont due aux modifications, des membres d'un groupe (c.f. Figure 4.18).

Nous avons montré comment l'architecture de OIU s'adapte aux dégradations de la communication. Nous montrons des résultats de simulation concernant l'adaptation aux changements contexte que sont dus à l'évolution de la mission ou en raison des contraintes de l'environnement (incendie par exemple) ou les nœuds (par exemple, pas assez d'énergie).

6.6 Conclusion

La reconfiguration dynamique de l'architecture des applications communicantes collaboratives évoluant dans un environnement où le contexte est variable est un problème de

recherche important que nous avons adessé dans nos travaux. Pour traiter ce besoin, nous proposons une approche de modélisation multi-niveau pour la reconfiguration des architectures. Nous proposons de gérer la reconfiguration en tenant compte des contraintes de niveau inférieur comme des exigences de niveau supérieur. D'abord, nous définissons un modèle générique basé sur les ontologies comme un modèle pour la couche application, et les règles SWRL comme un raffinement de la couche application vers la couche collaboration. Ensuite, nous développons un modèle de graphe pour les couches collaboration et middleware, et des productions de grammaires de graphes, pour la caractérisation, des architectures de la couche middleware qui raffine une architecture de la couche collaboration. Nous proposons une technique d'adaptation aux changements de l'état de ressources en conservant l'architecture de la couche application inchangée. Cette technique d'adaptation nécessite deux actions : le raffinement et la sélection, ce qui permet choisir la configuration optimale (la plus adapteé au contexte) parmi tous les configurations possibles de la couche middleware.

Les événements qui provoquent l'adaptation sont relatifs au contexte de l'environnement de l'application. Nous distinguons deux types de contexte : le contexte externe incluant, par exemple, les préférences de l'utilisateur, la présence et la position de l'utilisateur, la priorité de communications, etc., et le contexte lié aux ressources d'exécution, par exemple, le niveau de batterie, la charge CPU, la mémoire disponible de dispositifs utilisateurs, etc.

Nous avons utilisé les ontologies pour décrire le contexte et pour détecter les situations de collaboration possibles et décider quand les sessions doivent être créées, adaptées, etc. Les politiques d'adaptation sont définies par au moyen de règles SWRL. Ces règles permettent d'associer les transformations d'adaptation à l'évolution du contexte externe. Jess moteur est utilisé pour traiter les règles définies SWRL.

Nous avons développé un moteur d'appariement de transformation de graphes (GMTE : Graph Matching and Transformation Engine). Le GMTE est capable de rechercher de modèles de graphes petits et moyens dans des graphes de taille importante en peu de temps. L'outil peut être utilisé d'une manière non-interactive, en tant que bibliothèque, en fournissant une interface qui peut être invoquée à partir d'un programme externe. L'outil peut être utilisé en ligne de commande en lisant un graphe de règle et un graphe hôte à partir de fichiers TXT ou GraphML.

Pour valider notre approche, nous avons effectué des expérimentations en utilisant notre moteur GMTE et le moteur de règles Jess sur la grille de calcul *Grid'5000*.

Nous avons simulé différentes situations de notre cas d'étude (les OIU) montrant l'efficacité de notre approche. Nos résultats montrent qu'une machine gère mieux ces ressources grâce notre approche d'adaptation. Les situations de déconnexion de participant sont fortement réduites. Nous avons évaluée les temps d'exécution des règles SWRL et les temps de calcul des reconfigurations architecturales dues aux modifications des membres d'un groupe. Nous avons montré comment l'architecture du système de communication supportant une OIU s'adapte et résiste à la dégradation de la communication face à une évolution des besoins liés la mission ou suite aux changements de contexte liés à l'environnement.

De nombreuses questions restent encore ouvertes et nécessitent des investigations complémentaires. Elles sont énoncées ci-après.

Pour combler l'écart entre la couche middleware et la couche infrastructure, il semble opportun d'ajouter la couche transport. Le raffinement des entités des communications en entités au niveau du transport est ainsi nécessaire. Des extensions de la procédure de raffinement et des grammaires graphe devraient être une solution à cette question.

L'utilisation d'autres modèles pour la couche application est à considérer comme l'utilisation de UML à la place des ontologies. Concernant la couche middleware, l'utilisation d'un autre paradigme que les communications orientées évènements comme les architectures pair à pair est aussi à étudier.

La définition des politiques de sélection les plus riches (en tenant compte d'autres caractéristiques architecturales) est une perspective importante. Outre la dispersion et le coût relatif des politiques qui exploitent les structurelles des architectures logicielles serait intéressant.

Enfin, l'adaptation au contexte peut être encore plus étendue en ajoutant des concepts plus pertinent à l'ontologie du contexte relatif a des contraintes transport par exemple.

Résumé des Travaux de thèse

Titre : "Gestion dynamique des architectures pour les systèmes communicants collaboratifs"

Doctorant : Ismael Bouassida Rodriguez

Les travaux de recherche effectués dans le cadre de cette thèse abordent les problématiques d'auto adaptabilité ou d'auto configurabilité des applications distribuées dans des environnements machines et réseaux aux capacités et ressources variables. Pour ce genre d'applications, et pour faire face aux ressources variables de leur environnement d'exécution, il est nécessaire de pouvoir modifier leur architecture interne pendant leur exécution, ce qui caractérise l'aspect dynamique et introspectif de ces applications. De façon plus précise, la notion d'architecture dynamique caractérise des applications dont les composants sont créés, interconnectés et supprimés pendant l'exécution. Afin de garantir la validité des mises à jour de l'architecture, nous avons recours à des techniques formelles. En particulier, les grammaires de graphes représentent un moyen formel avec un pouvoir expressif suffisamment puissant pour spécifier les aspects statiques et dynamiques des architectures. Pour pouvoir décrire une architecture, nous nous basons sur l'approche ACG[4] qui traite l'évolution dynamique des architectures logicielles par la transformation de graphe. Les architectures dynamiques répondent à des contraintes liées à des variations de capacités de communication, de calcul et d'énergie, ainsi que des évolutions dans la nature des activités qu'elles soutiennent telles que les activités de groupe d'utilisateurs mobiles de type opération d'intervention d'urgence. Le caractère dynamique des architectures pose des difficultés supplémentaires pour leur description. Dans ce manuscrit, nous proposons de concevoir et de mettre en œuvre un environnement logiciel pour une "gestion guidée par les modèles" des changements dans les architectures des applications distribuées coopératives.

Les aspects adaptabilité des applications, les aspects transformations de graphe et les aspects particuliers des applications distribuées coopératives sont étudiés. Un moteur de transformation de graphes et un moteur de transformation d'architectures sont étudiés et étendus dans le but de les valider. Une approche d'adaptation s'appuyant sur une

[4]Abstract Component Graph

modélisation par les graphes et un style architectural de type Poducteur/Consommateur est présentée pour des applications communicantes collaboratives sensibles au contexte. Une démarche de raffinement est proposée permettant de garantir un certain degré d'adaptabilité en faisant un compromis entre les différents paramètres du contexte. Ces travaux de recherche ont aussi permis de définir un cadre algorithmique générique de reconfiguration architecturale multi-niveaux pour la sélection des architectures de déploiement les plus adaptées à un contexte et aux situations associées. Ce cadre a été appliquée au cas de la communication et de la coopération de groupe. Elle a aussi permis de modéliser le style architectural Producteur/Consommateur pour une communication orientée évènement. Des règles d'adaptation ont été définies. Elles comportent une partie basée sur SWRL[5] pour la description du contexte et des règles d'adaptation, et une partie basée sur les grammaires de graphes pour la transformation des configurations de déploiement.

Enfin, nous avons apporté les modifications et les extensions nécessaires au moteur de transformation d'architecture afin de l'adapter aux besoins des applications communicantes collaboratives et implanté le cadre algorithmique selon une technique guidée par les modèles pour élaborer des politiques de reconfiguration correctes par construction. Nous avons développé des ontologies et des règles SWRL pour décrire le contexte et le raisonnement lié à l'adaptation des configurations aux différentes situations.

[5]Semantic Web Rule Language

Author's Publications

Publications in International Journals

[1] I. Bouassida, K. Drira, C. Chassot, and M. Jmaiel. Context-aware adaptation for group communication support applications with dynamic architecture. *System and Information Sciences Notes*, 2(1):88–92, 2007.

[2] I. Bouassida, K. Drira, C. Chassot, and M. Jmaiel. An architectural refinement model for group-wide communications with priorities applied to the ROSACE project senario. *International Tarnsactions on Systems Science and Applications*, 6(1), 2010.

[3] I. Bouassida, K. Guennoun, K. Drira, C. Chassot, and M. Jmaiel. A rule-driven approach for architectural self adaptation in collaborative activities using graph grammars. *International Journal of Autonomic Computing*, 1(3):226–245, 2010.

[4] I. Bouassida, N. Van Wambeke, K. Drira, C. Chassot, and M. Jmaiel. Multi-layer coordinated adaptation based on graph refinement for cooperative activities. *Communications of SIWN*, 4(1):163–167, 2008.

Publication in a French Journal

[5] I. Bouassida, C. Chassot, and M. Jmaiel. Graph grammar-based tarnsformation for context-aware architectures supporting group communication. *Revue des Nouvelles Technologies de l'Information*, L(19):29–42, 2010.

Book Chapter

[6] J. Lacouture, I. Bouassida, K. Drira, C. Chassot, M. Sibilla, T. Desprats, C. Tessier, J-P. Arcangeli, V. Noel, and F. Garijo. Mission-aware adaptive communication for collaborative mobile entities. In *Handbook of Research on Mobility and Computing*. IGI Global, 2010.

Publications in International Conferences

[7] I. Bouassida, R. Ben Halima, K. Drira, C. Chassot, and M. Jmaiel. A policy-driven dynamic reconfiguration for virtualized web services-based architectures. In *Business System Management and Engineering workshop (BSME 2010)*, Malaga (Spain), 2010. 15 p.

[8] I. Bouassida, K. Drira, C. Chassot, and M. Jmaiel. Context-aware adaptation for group communication support applications with dynamic architecture. In *3rd International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS'2007)*, pages 88–92, Leipzig (Germany), 2007.

[9] I. Bouassida, K. Drira, C. Chassot, and M. Jmaiel. A model-based multi-level architectural reconfiguration applied to adaptability management in context-aware cooperative communication support systems. In *Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, WICSA/ECSA 2009*, pages 353–356, Cambridge (United Kingdom), 2009.

[10] I. Bouassida, K. Guennoun, K. Drira, C. Chassot, and M. Jmaiel. Implementing a rule-driven approach for architectural self configuration in collaborative activities using a graph rewriting formalism. In *5th International Conference on Soft Computing as Tarnsdisciplinary Science and Technology (CSTST'2008)*, pages 484–491, Cergy Pontoise (France), 2008.

[11] I. Bouassida, J. Lacouture, and K. Drira. Semantic driven self-adaptation of communications applied to ercms. In *The 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010)*, pages 1292–1299, Perth (Australia), 2010.

[12] I. Bouassida, G. Sancho, T. Villemur, S. Tazi, and K. Drira. A model-driven adaptive approach for collaborative ubiquitous systems. In *AUPC 09: Proceedings of the 3rd workshop on Agent-oriented software engineering challenges for ubiquitous and pervasive computing*, pages 15–20, Londres (United Kingdom), 2009. ACM.

[13] I. Bouassida, N. Van Wambeke, K. Drira, C. Chassot, and M. Jmaiel. Multi-layer coordinated adaptation based on graph refinement for cooperative activities. In *4th International Conference on Self-organization and Adaptation of Computing and Communications (SACC 2008)*, pages 163–167, Glasgow (United Kingdom), 2008.

[14] K. Drira and I. Bouassida. A demonstration of an efficient tool for graph matching and tarnsformation. In *10ème Conférence Internationale Francophone sur l'Extraction et la Gestion des Connaissances (EGC 2010)*, pages 71–73, Hammamet (Tunisia), 2010.

[15] S. Marzouk, A. Jmal Maalej, I. Bouassida, and M. Jmaiel. Periodic checkpointing for strong mobility of orchestrated web services. In *International Workshop on Self-Healing Web Services (SHWS 2009). 7th IEEE International Conference on Web Services (ICWS 2009)*, pages 203–210, Los Angeles (USA), 2009.

[16] G. Sancho, I. Bouassida, T. Villemur, and S. Tazi. What about collaboration in ubiquitous environments. In *10th annual international conference on New Technologies of Distributed Systems (NOTERE'10)*, pages 143–150, Tozeur (Tunisia), 2010.

[17] G. Sancho, I. Bouassida, T. Villemur, S. Tazi, and K. Drira. A model-driven adaptive framework for collaborative ubiquitous systems. In *9th annual international conference on New Technologies of Distributed Systems (NOTERE'09)*, pages 233–244, Montréal (Canada), 2009.

Publications in French Spoken Conferences

[18] I. Bouassida. Approche d'adaptation au contexte pour les applications de communication de groupe à architectures dynamiques. In *Colloque de l'Ecole Doctorale Informatique et Télécommunications (EDIT'2007)*, pages 111–114, Toulouse (France), 2007.

[19] I. Bouassida. Dynamic architectures management for cooperative applications adaptability. In *Ecole d'été RESCOM 2008*, Saint-Jean-Cap-Ferrat (France), 2008. 2 p.

[20] I. Bouassida. Rule-driven approach for architectural self configuration using graph grammars. In *Ecole d'été RESCOM 2009*, La Palmyre (France), 2009. 2 p.

[21] I. Bouassida, C. Chassot, and M. Jmaiel. Graph grammar-based tarnsformation for context-aware architectures supporting group communication. In *4ème Conférence francophone sur les Architectures Logicielles*, pages 29–42, Pau (France), 2010.

Project Reports

[22] I. Bouassida, K. Drira, J.C. Fabre, J. Lauret, and A. Maalej. Imap (information management for avionics plateforme) wp1.1.4. state of the art. Contract reports, LAAS, 128p., 2010. IMAP, Airbus.

[23] C. Chassot, E. Exposito, N. Van Wambeke, F. ARMANDO, I. Bouassida, K. Drira, C. Brandauer, F. Strohmeier, P. Dorfinger, P.A. Aranda Gutierrez, A. Flizikowski, P.N.M. Sampaio, I. Miloucheva, S. Avallone, L. Vollero, S. Pietro Romano, M. Michalak, M. Roth, and S. Rao. D2.1. qos solution analysis and user/application requirements and business scenarios. Contract reports, LAAS-CNRS, 78p., 2006. Projet IST NETQoS 033516.

[24] K. Drira, T. Villemur, S. Tazi, G. Sancho, and I. Bouassida. Full framework specification of M2M service network. Contract reports, LAAS-CNRS, 126p., 2010. Projet UseNet ITEA2 06004.

[25] J. Latvakoski, J. Kallio, T. Hautakoski, T. Vaisanen, J. Toivonen, T. Parkkila, A. Peterzens, A. Deckers, P. Dobbelaere, P. Dedecker, J. Hoebeke, B. Meekers, F. Martinez, L. Manero, I. Lucas, D. Pirez, A. Zabala, K. Drira, T. Villemur, S. Tazi, V. Baudin, T. Chaari, G. Sancho, I. Bouassida, and P. Rossignol. M2M requirements. Contract reports, LAAS-CNRS, 129p., 2008. Projet UseNet ITEA2 06004.

[26] P. Rossignol, C. Sauvignac, P. Meurice, J.M. Tarn, P. Maseres, B. Tesniere, S. Ramos, V. Baudin, I. Bouassida, T. Chaari, K. Drira, G. Sancho, S. Tazi, T. Villemur, D. Pirez, and Y. Lopez. Uses cases components. Contract reports, LAAS-CNRS, 36p., 2008. Projet UseNet ITEA2 06004.

[27] P. Rossignol, C. Sauvignac, P. Meurice, J.M. Tarn, P. Maseres, B. Tesniere, S. Ramos, V. Baudin, I. Bouassida, K. Drira, G. Sancho, S. Tazi, T. Villemur, D. Pirez, and Y. Lopez. Specification of smart usage demonstrator. Contract reports, LAAS-CNRS, 41p., 2009. Projet UseNet ITEA2 06004.

Bibliography

[ABB+06] N. Agoulmine, Sasitharan Balasubramaniam, Dmitri Botvich, John Strassner, E. Lehtihet, and William Donnelly. Challenges for autonomic network management. In *1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE)*, 2006.

[ADB+99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.

[BBF09] G. Blair, N. Bencomo, and R.B. France. Models@ run.time. *Computer*, 42(10):22 –27, oct. 2009.

[BDVT04] V. Baudin, K. Drira, T. Villemur, and S. Tazi. *E-Education applications: human factors and innovative approaches*, chapter A model-driven approach for synchronous dynamic collaborative e-learning, pages 44–65. Ed. C.Ghaoui, Information Science Publishing, ISBN 1-59140-292-1, 2004.

[BEH+01] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M. Scott Marshall. GraphML Progress Report. In *Graph Drawing*, pages 501–512, 2001.

[BG08] Basil Becker and Holger Giese. Modeling of correct self-adaptive systems: a graph transformation system based approach. In *CSTST '08: Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology*, pages 508–516, New York, NY, USA, 2008. ACM.

[BLNS06] Korinna Bade, Ernesto William De Luca, Andreas Nürnberger, and Sebastian Stober. Carsa - an architecture for the development of context adaptive retrieval systems. In Keith van Rijsbergen, Andreas Nürnberger, Joemon M. Jose, and Marcin Detyniecki, editors, *Adaptive Multimedia Retrieval: User, Context, and Feedback*. Springer-Verlag, 2006.

[Bri01] P. G. Bridges. Supporting coordinated adaption in networked systems. In *HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, page 162, Washington, DC, USA, 2001. IEEE Computer Society.

[CCD+05] Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier Richard. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In *SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing Grid'2005*, pages 99–106, Seattle, USA, November 13-14 2005. IEEE/ACM.

[CGD+06] Christophe Chassot, Karim Guennoun, Khalil Drira, François Armando, Ernesto Exposito, and André Lozes. Towards autonomous management of qos through model-driven adaptability in communication-centric systems. *ITSSA*, 2(3):255–264, 2006.

[Cha07] Tarak Chaari. *Adaptation d'applications pervasives dans des environnements multi-contextes*. PhD thesis, INSA de Lyon, September 2007.

[Cho56] N. Chomsky. Three models for the description of language. *Information Theory, IEEE Transactions on*, 2(3):113–124, 1956.

[CK00] Fangzhe Chang and Vijay Karamcheti. Automatic configuration and run-time adaptation of distributed applications. In *HPDC*, pages 11–20, 2000.

[CL04] Tarak Chaari and Frédérique Laforest. Génération et adaptation automatiques des interfaces utilisateurs pour des environnements multiterminaux projet sefagi (simple environment for adaptable graphical interfaces). *Ingénierie des Systèmes d'Information*, 9(2):11–38, 2004.

[CLC07] Tarak Chaari, Frï¿½dï¿½rique Laforest, and Augusto Celentano. Adaptation in Context-Aware Pervasive Information Systems: The SECAS Project. *Int. Journal on Pervasive Computing and Communications(IJPCC)*, 3(4):400–425, December 2007.

[CS99] Peter H. Carstensen and Kjeld Schmidt. Computer supported cooperative work: New challenges to systems design. In *In K. Itoh (Ed.), Handbook of Human Factors*, pages 619–636, 1999.

[DDF+06] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, 2006.

[DK04] Alejandro Buchmann Dimka Karastoyanova. Extending web service flow models to provide for adaptability. In *OOPSLA '04 Workshop on Best*

Practices and Methodologies in Service-oriented Architectures: Paving the Way to Web-services Success, Vancouver, Canada, 2004.

[DvdHT02] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. An infrastructure for the rapid development of xml-based architecture description languages. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 266–276, New York, NY, USA, 2002. ACM.

[EBP01] C. Ermel, R. Bardhol, and J. Padberg. Visual design of software architecture and evolution based on graph transformation. In *Uniform Approches to graphical process specification Techniques*, Genove, Italy, April 2001.

[Edw05] W. Keith Edwards. Putting computing in context: An infrastructure to support extensible context-enhanced collaborative applications. *ACM Trans. Comput.-Hum. Interact.*, 12(4):446–474, 2005.

[EGR91] Clarence A. Ellis, Simon J. Gibbs, and Gail Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

[EHP+96] W. Ellis, R. Hilliard, P. Poon, D. Rayford, T. Saunders, B. Sherlund, and R. Wade. Toward a recommended practice for architectural description. In 2^{nd} *IEEE International Conference on Engineering of Complex Computer Systems*, pages 21–25, Montreal, Canada, October 1996.

[Ehr87] Hartmut Ehrig. Tutorial introduction to the algebraic approach of graph grammars. In *Proceedings of the 3rd International Workshop on Graph-Grammars and Their Application to Computer Science*, pages 3–14, London, UK, 1987. Springer-Verlag.

[EK91] Hartmut Ehrig and H.-J Kreowski. *Graph Grammars and Their Application to Computer Science: 4th International Workshop, Bremen, Germany, March 5-9, 1990 Proceedings*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1991.

[ESB07] Dejene Ejigu, Marian Scuturici, and Lionel Brunie. Coca: A collaborative context-aware service platform for pervasive computing. In *Information Technology, 2007. ITNG '07. Fourth International Conference on*, pages 297–302, April 2007.

[ESD03] Ernesto Exposito, Patrick Senac, and Michel Diaz. FPTP: the XQoS aware and fully programmable transport protocol. In *Proc. The 11th*

IEEE International Conference on Networks (ICON'2003), Sydney, Australia, 2003.

[FDBC00] A. Friday, N. Davies, G. Blair, and K. Cheverst. Developing adaptive applications: The most experience. *Integrated Computer-Aided Engineering*, 6(2):143– 157, 2000.

[FH00] H. Fahmy and R. Holt. Using graph rewriting to specify software architectural transformations. In 15^{th} *IEEE international Conference on Automated Software Engineering, ISBN 0-7695-0710-7*, pages 187–196, Grenoble, France, September 2000.

[FNBS07] Y. Fawaz, A. Negash, L. Brunie, and V.-M. Scuturici. Conami: Collaboration based content adaptation middleware for pervasive computing environment. In *Pervasive Services, IEEE International Conference on*, pages 189–192, July 2007.

[GDD04] Karim Guennoun, Khalil Drira, and Michel Diaz. A proved component-oriented approach for managins dynamic software architectures. In *Proc. 7th iasted international conference on software engineering and application*, Marina Del Rrey, CA, USA, 2004.

[GG97] Richard Grimes and Dr Richard Grimes. *Professional Dcom Programming*. Wrox Press Ltd., Birmingham, UK, 1997.

[GP95] D. Garlan and D. Perry. Introduction to the special issue on software architecture. *IEEE Transactions On Software Engineering*, 21(4):269–274, April 1995.

[GUE06] K. GUENNOUN. *Architectures dynamiques dans le contexte des applications à base de composants et orientées service*. PhD thesis, Université Paul Sabatier, Toulouse, 183p., 2006. Doctorat.

[GvdHT09] J.C. Georgas, A. van der Hoek, and R.N. Taylor. Using architectural models to manage and visualize runtime adaptation. *Computer*, 42(10):52 –60, oct. 2009.

[HIM98] Dan Hirsch, Paolo Inverardi, and Ugo Montanari. Graph grammars and constraint solving for software architecture styles. In *ISAW '98: Proceedings of the third international workshop on Software architecture*, pages 69–72, New York, NY, USA, 1998. ACM.

[HL96] Barron C. Housel and David B. Lindquist. Webexpress: a system for optimizing web browsing in a wireless environment. In *MobiCom '96:*

Proceedings of the 2nd annual international conference on Mobile computing and networking, pages 108–116, New York, NY, USA, 1996. ACM.

[HPSB+04a] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. Swrl: A semantic web rule language combining owl and ruleml. W3c member submission, W3C, May 2004.

[HPSB+04b] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004, 2004.

[IBM06] IBM. An architectural blueprint for autonomic computing. White paper, IBM Corporation, June 2006.

[IEE00] IEEE. Ieee std 1471-2000, ieee recommended practice for architectural description of software-intensive systems. In *IEEE*, pages i–23, 2000.

[JI07] Jeffrey H. Johnson and Pejman Iravani. The multilevel hypernetwork dynamics of complex systems of robot soccer agents. *ACM Trans. Auton. Adapt. Syst.*, 2(2):5, 2007.

[KBC02] A. Ketfi, N. Belkhatir, and P. Y. Cunin. Adaptation dynamique, concepts et experimentations. In *Proceedings of ICSSEA*, 2002. In French.

[KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.

[KK88] Kenneth L. Kraemer and John Leslie King. Computer-based systems for cooperative work and group decision making. *ACM Comput. Surv.*, 20(2):115–146, 1988.

[Lit07] Marin Litoiu. A performance analysis method for autonomic computing systems. *TAAS*, 2(1), 2007.

[LJP+06] M.C. Lee, H.K. Jang, Y.S. Paik, S.E. Jin, and S. Lee. Ubiquitous device collaboration infrastructure: Celadon. In *Software Technologies for Future Embedded and Ubiquitous Systems, 2006 and the 2006 Second International Workshop on Collaborative Computing, Integration, and Assurance. SEUS 2006/WCCIA 2006. The Fourth IEEE Workshop on*, pages 6 pp.–, April 2006.

[LP06] Hua Liu and M. Parashar. Accord: a programming framework for autonomic applications. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 36(3):341–352, May 2006.

[LV07b] Marco P. Locatelli and Giuseppe Vizzari. Awareness in collaborative ubiquitous environments: The multilayered multi-agent situated system approach. *ACM Trans. Auton. Adapt. Syst.*, 2(4):13, 2007.

[Mah04] Mahadev Satyanarayanan. The many faces of adaptation. *IEEE Pervasive Computing*, 3:4–5, 2004.

[Mao09] S. Maoz. Using model-based traces as runtime models. *Computer*, 42(10):28 –36, oct. 2009.

[MBJ+09] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. Models@ run.time to support dynamic adaptation. *Computer*, 42(10):44 –51, oct. 2009.

[MC02] René Meier and Vinny Cahill. Taxonomy of distributed event-based programming systems. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 585–588, Washington, DC, USA, 2002. IEEE Computer Society.

[Mes95] B.T. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs*. PhD thesis, Institut für Informatik und angewandte Mathematik, Universität Bern, Switzerland, 1995.

[MH00] Vlada Matena and Mark Hapner. *Applying Enterprise Javabeans: Component-Based Development for the J2ee Platform*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[MHSC04] Francis J. Mullany, Lester T. W. Ho, Louis G. Samuel, and Holger Claussen. Self-deployment, self-configuration: Critical future paradigms for wireless access networks. In *Proceedings of the 1st IFIP TC6 WG6.6 International Workshop on Autonomic Communication (WAC 2004)*, volume 3457 of *Lecture Notes in Computer Science*, pages 58–68. Springer, 2004.

[Mic01] Microsoft. Microsoft .Net passport technical overview. Technical report, Microsoft, september 2001.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[NH04] Nidal Nasser and Hossam Hassanein. Adaptive bandwidth framework for provisioning connection-level qos for next-generation wireless cellular networks. *Canadian Journal of Electrical and Computer Engineering*, 29(1):101–108, 2004.

[NKSB99] T. Nandagopal, T. Kim, P. Sinha, and V. Bharghavan. Service differentiation through end-to-end rate control in low bandwidth wireless packet networks. In *Proceedings of the 6th IEEE International Workshop on Mobile Multimedia Communications (MOMUC'99)*, San Diego, California, USA, November 1999.

[NSS01] Katashi Nagao, Yoshinari Shirai, and Kevin Squire. Semantic annotation and transcoding: Making web content more accessible. *IEEE MultiMedia*, 08(2):69–81, 2001.

[OBAA04] Özgür B. Akan and Ian F. Akyildiz. Atl: an adaptive transport layer suite for next-generation wireless internet. *IEEE Journal on Selected Areas in Communications*, 22(5):802–817, 2004.

[OMG99] OMG. Corba components: Joint revised submission. Technical Report orbos/99-02-05, Object Management Group, 1999.

[OSG07] OSGi Alliance. *OSGi Service Platform Release 4*, May 2007.

[PAC+01] Andrew Perkis, Yousri Abdeljaoued, Charilaos Christopoulos, Touradj Ebrahimi, and Joe F. Chicharo. Universal multimedia access from wired and wireless systems. *Circuits, Systems, and Signal Processing*, 20(3-4):387–402, 2001.

[PE02] Angel Puerta and Jacob Eisenstein. Ximl: a common representation for interaction data. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces*, pages 214–215, New York, NY, USA, 2002. ACM.

[PJYF05] Filip Perich, Anupam Joshi, Yelena Yesha, and Tim Finin. Collaborative joins in a pervasive computing environment. *The VLDB Journal*, 14(2):182–196, 2005.

[PLZ08] A. Padovitz, S.W. Loke, and A. Zaslavsky. Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 38(4):729–742, July 2008.

[PP00] Paulo Pinheiro da Silva and Norman W. Paton. UMLi: The unified modeling language for interactive applications. In Andy Evans, Stuart Kent, and Bran Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings*, volume 1939, pages 117–132. Springer, 2000.

[Rog97] Dale Rogerson. *Inside COM: Microsoft's Component Object Model*. Microsoft Press, 1997.

[Roz97] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[Sat01] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, pages 10–17, August 2001.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90, Dec. 1994.

[STS03] Jun Zhao Sun, Jari Tenhunen, and Jaakko Sauvola. Cme: a middleware architecture for network-aware adaptive applications. In *Proc. 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 3, pages 839–843, Beijing, China, 2003.

[STV08] G. Sancho, S. Tazi, and T. Villemur. A semantic-driven auto-adaptive architecture for collaborative ubiquitous systems. In *5th International Conference on Soft Computing as Transdisciplinary Science and Technology (CSTST'2008)*, pages 650–655, Cergy Pontoise (France), 2008.

[SWM04] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL Web ontology Langage Guide. W3C Recommendation, February 2004. URL: http://www.w3.org/TR/owl-guide/.

[Tho98] Anne Thomas. Enterprise javabeans technology: Server component model for the java platform. Technical report, patrica Seybold Group, December 1998.

[Ull76] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, January 1976.

[vdMDS+06] Sven van der Meer, Willie Donnelly, John Strassner, Brendan Jennings, and Mícheál Ó Foghlú. Emerging principles of autonomic network management. In *1st IEEE International Workshop on Modelling Autonomic Communications Environments (MACE)*, 2006.

[WA03] Susie J. Wee and John G. Apostolopoulos. Secure scalable streaming and secure transcoding with jpeg-2000. In *ICIP (1)*, pages 205–208, 2003.

[WHZ+01] D. Wu, Y. Hou, W. Zhu, Y. Zhang, and J. Peha. Streaming video over the internet: Approaches and directions. *IEEE Transactions on Circuit and Systems for Video*, 11(3):282–300, 2001.

[XB05] Jin Xiao and Raouf Boutaba. Qos-aware service composition and adaptation in autonomic communication. *IEEE Journal on Selected Areas in Communications*, 23(12):2344–2360, 2005.

[YYF00] Alexander V Konstntinou Yechiam Yemini and Danilo Florissi. Nestor: An architecture for network selfmanagement and organization. *IEEE Journal on selected areas in communications*, 18(5):758 –766, May 2000.

[ZLL07] Xianggang Zhang, Zhiqiang Li, and Jiude Liu. An adaptive infrastructure concept model based on corba in pervasive computing. In *Pervasive Computing and Applications, 2007. ICPCA 2007. 2nd International Conference on*, pages 490–495, July 2007.

[ZPM+07] Yu Zhou, Jian Pan, Xiaoxing Ma, Bin Luo, Xianping Tao, and Jian Lu. Applying ontology in architecture-based self-management applications. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 97–103, New York, NY, USA, 2007. ACM.

Résumé

Dans ce manuscrit, nous proposons de concevoir et de mettre en oeuvre un environnement logiciel pour une ``gestion guidée par les modèles'' des changements dans les architectures des applications distribuées coopératives. Les aspects adaptabilité des applications, les aspects transformations de graphe et les aspects particuliers des applications distribuées coopératives sont étudiés. Une approche d'adaptation s'appuyant sur une modélisation par les graphes et un style architectural de type Poducteur/Consommateur est présentée pour des applications communicantes collaboratives sensibles au contexte. Une démarche de raffinement est proposée permettant de garantir un certain degré d'adaptabilité en faisant un compromis entre les différents paramètres du contexte. Ces travaux de recherche ont aussi permis de définir un cadre algorithmique générique de reconfiguration architecturale multi-niveaux pour la sélection des architectures de déploiement les plus adaptées à un contexte et aux situations associées. Ce cadre a été appliqué au cas de la communication et de la coopération de groupe. Elle a aussi permis de modéliser le style architectural Producteur/Consommateur pour une communication orientée évènement. Des règles d'adaptation ont été définies. Elles comportent une partie basée sur SWRL pour la description du contexte et des règles d'adaptation, et une partie basée sur les grammaires de graphes pour la transformation des configurations de déploiement

Mots clefs
Reconfiguration dynamique, Grammaire de graphes, Collaboration, Système communicants adaptatifs.

Abstarct

In this work, we study dynamic reconfiguration of collaborative communicating applications. Providing generic and scalable solutions for automated self-reconfiguration in group collaboration support systems can be driven by rule-based reconfiguration policies. To achieve this goal, we elaborate a dynamic graph-based modelling approach and we develop structural models that can represent the different interaction dependencies from different configuration-related point of views: communication flows between the distributed machines, the networked deployment nodes, and the service composition. Our solution is based on graph grammars rewriting. We provide graph transformation to specify rules for changing deployment architecture while being in conformance to an architectural style. In order to handle the complex design of communicating collaborative system architectures and the related adaptation issues, we propose a multi-layer modelling approach. This approach assures generic solutions for automatic context aware adaptation. Our approach is based on the observation that semantic data analysis that can be exploited to manage priorities and more generally to manage communications. This allows us to represent, in a richer way, the semantics of the managed systems.

Keywords
Dynamic Reconfiguration, Graphs Grammar, Collaboration, Communicating Systems