Graph grammar-based transformation for context-aware architectures supporting group communication

Ismael Bouassida Rodriguez[*,**], Christophe Chassot[*,**], Mohamed Jmaiel[***]

[*]CNRS; LAAS; 7 avenue du colonel Roche, F-31077 Toulouse, France
[**]Université de Toulouse; UPS, INSA, INP, ISAE; LAAS; F-31077 Toulouse, France
bouassida@laas.fr,chassot@laas.fr
[***]Uiversity of Sfax, ReDCAD Research Unit, B.P. 1173, 3038 Sfax, Tunisia
mohamed.jmaiel@rnu.enis.tn

Abstract. Handling context-aware dynamically adaptable architectures contributes to the design of self-configuring software systems. This kind of problem for communicating systems is even more challenging since adaptation should address simultaneously the different levels. This is necessary for handling both changes in the low level constraints and evolutions in the high level requirements. In this paper, we address this problem by providing a model-based, rule-oriented approach that supports the adaptation process based on a run-time transformation of the system architecture. Such architecture may represent the different possible service compositions and the associated architectural configurations. We consider the multi-level models of the communicating system architecture and the intra-level architecture transformations as the elementary adaptation actions. We handle consistently the related inter-level adaptation actions by considering additional architectural relationships viewing the lower level architecture as a refinement of the upper level.We provide the algorithms characterizing the multi-level architecture-based adaptation process. We then develop a rule-oriented implementation using graph grammar and handling architectural transformations as graph transformation rules. We consider Emergency Response and Crisis Management Systems (ERCMS) as a case study from the more general group communication systems to which our results apply.

1 Introduction

Designing and implementing self-adaptive communicating systems is a complex task, which may be addressed via model-based design approaches associated with automated management techniques for dynamic architectural adaptability. In self-adaptable applications, components are created and connected, or removed and disconnected during the execution. The architectural changes respond to constraints of the communication and resources execution capacities variations. Providing solutions for distributed software systems supporting group communication requires managing dynamically evolving group membership and dynamically connecting deployment nodes.

For a number of group communication-based applications, deciding such a reconfiguration for one or more levels of interaction and distribution will depend on the situation of a run-time changing context.

Providing generic solutions for automated self-reconfiguration in group communication support systems can be guided by rule-based reconfiguration policies. This is the approach we adopt in this paper. In order to guarantee the architecture reconfiguration correctness we use formal techniques.

In particular, graphs represent an appropriate mean to specify respectively static and dynamic architectures aspects. The solution we present in this paper, is based on graph grammar theories. It handles architectural refinement as graphs where vertices are assimilated to software services and components. It provides the graph transformation rules allowing to handle the deployment architecture changes at run-time.

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 having different roles and functions. A model-based approach for adaptability management presented in section 3. The ERCMS case study is presented in section 4. The Graph-Grammar based rule-oriented implementation is provided in section 5. Section 6 gives The graph grammar for architecture reconfiguration. Conclusion is provided in the last section.

2 Related work

Adaptation objectives, actions and properties are among the main facets of adaptability. They are studied and classified in this section.

Two different adaptability views may be distinguished: the design time adaptability Fahmy and Holt (2000); Ermel et al. (2001) and the run-time adaptability Chang and Karamcheti (2000). For the first view, we can find design support that handles the application development cycle and optimizes the resource value by insuring that the infrastructure answering clearly and in a measurable way to activity requirements. For the run-time adaptability Friday et al. (2000) presents several adaptation techniques among which use proxy services, change model of interaction and reorganize application structure.

Adaptation approaches are also targeting different architectural levels including Service, Middleware and Network levels. At the first level, the Service-Oriented Architecture (SOA) paradigm is based on dynamic services publishing and discovering . This kind of architectures provides the possibilities to dynamically compose services for adapting applications to context. Service descriptions are published, via the registry, by service providers and dynamically discovered by service requesters.

Other frameworks are proposed to provide adaptability for the Middleware level. In Nasser and Hassanein (2004), an adaptive framework supporting multiple classes of multimedia services with different QoS requirements in wireless cellular networks is proposed. Sun et al. (2003) proposes CME, a Middleware architecture for service adaptation based on network awareness. CME is structured as a software platform both to provide network awareness to applications and to manage network resources in an adaptive fashion. Friday et al. (2000) applies reflection to design of Middleware adaptive platforms.Wu et al. (2001) addresses the need for adaptation in video streaming applications distributed over the Best-Effort Internet. Several techniques have been proposed based on two mechanisms: an applicative

congestion control, which can be implemented in several ways: rate control, rate-adaptive video encoding, rate shaping; and error control integrating concepts such as delay-constrained retransmissions and forward error correction.

At the Network level, Exposito et al. (2003) 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 micro-protocol by another following a plug and play approach.

The adaptation solutions suggested in the literature distinguish behavioural and architectural aspects. The adaptation is behavioural (or algorithmic) when the behaviour of the adaptive service can be modified, without modifying its structure. Standard protocols such as TCP and specific protocols such as Wu et al. (2001) provide behaviour-based adaptation mechanisms. Behavioural adaptation is easy to implement but limits the adaptability properties.

The adaptation is architectural when the service composition can be modified Garlan and Perry (1995); Ellis et al. (1996) dynamically. In self-adaptive applications components are created and connected, or removed and disconnected during the execution. The architectural changes respond to constraints related to the execution context involving, for example, variations of communication networks and processing resources. They may also respond to requirement evolution in the supported activities involving, for example, mobility of users and cooperation structure modification.

Designing and implementing self-adaptive communicating systems is a complex task. To handle this complexity, several studies Ganek and Corbi (2003) showed the need to lay on model-based design approaches associated with automated management techniques.

Static architectures are described by instances of components and interconnection links. The dynamic character of architectures requires additional description rules. Several works have addressed the dynamic architecture description, using different approaches Allen and Garlan (1997). In order to guarantee the architecture evolving, correctness formal techniques are used. In particular, graphs represent a powerful expressive mean to specify respectively static and dynamic architectures aspects LeMetayer (1998); Hirsch et al. (1999). For such approaches, graph vertices represent the software components, and the edges represent the links between these components. Dynamic architectures are described as graph grammars and architecture transformation is specified and ruled using graph rewriting models. This is also our approch here.

3 Model-based Approach for Adaptability Management

Managing self-adaptability in group communication activities requires considering several kinds of evolving requirements and changing constraints, and lead to architectural adaptations at different levels of the communication stack. This raises a coordination problem which, if not properly addressed, may conduct to inefficient or even inconsistent solutions.

Managing architectural adaptations require defining and modelling abstraction levels dedicated to specific parts of the whole adaptation. Distinguishing these different abstraction levels allows designers and developers to respectively master the specification and the implementation of adaptation rules. For a given deployment configuration $A_{n,i}$[1] at level n, a set

[1] In the notation $A_{n,1}$, n denote the level that the configuration A belongs and i the index of this configuration.

TAB. 1 – *Abstract Adaptability Algorithm*

1 Select the initial configuration of level n: $A_{n,0}$
2 Call $G\_Refine()$ to compute the correspondent configurations at level $n-1$
3 Get context values (e.g the available power, the available memory)
4 Call $Weight\_based\_selection()$ to select the suitable configuration at level $n-1$
5 loop
6 {
7 Wait for a reconfiguration event: e
8 Call $G\_Reconfigure()$ to generates a new deployment configuration (level n)
9 Call $G\_Refine()$ to compute the correspondent configurations at level $n-1$
10 Get context values (e.g the available power, the available memory)
11 Call $Distance\_based\_selection()$ to select the suitable configuration at level $n-1$
12 }

of deployment configurations $\mathbb{A}_{n-1,i}$[2]$= (A_{n-1,1}, ..., A_{n-1,p})$ may be implemented at level $n-1$. Adapting the architecture to constraint changes at level $n-1$ by switching among these multiple deployment configurations allows maintaining unchanged the n-level deployment configuration.

Moreover, when adaptation requires changes at level n, this may need no changes at level $n-1$ if initial and new deployment configurations of level n (e.g. changes from $A_{n,1}$ to $A_{n,2}$) share common implementations (e.g. $A_{n-1,p}$) at level $n-1$.

3.1 Generic Graph Adaptability Algorithm

We present an algorithm (table 1) and four procedures (tables 2, 3, 4 and 5) that handle the refinement and the reconfiguration process. We use graph grammar-based implementation of the reconfiguration and the refinement procedures. Generative grammars are described in general as a classical grammar system where there is an axiom, a set of non-terminal nodes, a set of terminal nodes, and a set of transformation rules P, also called grammar productions. An instance belonging to the graph grammar is a graph containing only terminal nodes and is obtained starting from the axiom by applying a sequence of productions from P.

We define also functions that allow selecting the suitable deployment configurations at each step of the reconfiguration process.

3.2 Algorithm Execution

The adaptation algorithm begins by selecting and refining the initial deployment configuration (table 1 line 1 and 2). The procedure $G\_Refine()$ (table 2) handles these two actions.

After the refinement of the initial deployment configuration, the adaptability algorithm selects the optimal deployment configuration at the level $n-1$ (table 1 line 4) that implements

[2]In the notation $\mathbb{A}_{n-1,i}$, $n-1$ denote the level and i the index of the n-level configuration implemented by $\mathbb{A}_{n-1,i}$.

TAB. 2 – *The Graph Refinement Procedure*

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

TAB. 3 – *The Weight Based Selection Algorithm*

1 Weight_based_Selection()
2 {
3 Let C denote the context attributes (e.g the available power, the available memory)
4 Let $A_{n,p} \in \mathbb{A}_n, p \in \mathbb{N}$, be a given deployment configuration
5 Select $A_{n,p} \in \mathbb{A}_n, p \in \mathbb{N}$
6 Select $S_1 = \{A_{n-1,k} \in \mathbb{A}_{n-1,p}, k \in \mathbb{N}$ such that:
$Weight(A_{n-1,k}) \leq Weight(X), \forall X \in \mathbb{A}_{n-1,p}\}$
7 if $card(S_1) > 1$
8 Select $S_2 = \{A_{n-1,k} \in S_1, k \in \mathbb{N}$ such that:
$Contextaware\_Cost(A_{n-1,k}, C) \leq Contextaware\_Cost(X, C), \forall X \in S_1\}$
9 if $card(S_2) > 1$
10 Select any deployment configuration from S_2
11 }

the initial deployment configuration at the level n. The refinement procedure $G\_Refine()$ (table 2) corresponds to the application of a set of grammar productions $p_1 \dots p_k$ that implement the refinement of a deployment configuration from the level n to level $n-1$.

The procedure $Weight\_based\_Selection()$ (table 3) formalizes the selection suitable architectures. The selection is based on minimizing the $Weight()$ defined as a generic function independent of the context (table 3 line 6). This function corresponds to the cost or to the efficiency/performance of an architecture. For example, it can be defined as the number of software components by deployment node or the scope of an architecture.

The $Contextaware\_Cost()$ (table 3 line 8) is a generic function that is aware of dependent of the context (table 3 line 3). This function can be related to the communication and the resources constraints of the architecture. For example it can express the availability level of a given resource (Bandwidth, Memory). After selecting the initial configuration, the adaptability algorithm waits for a reconfiguration event e (table 1 line 8).

When a reconfiguration event occurs, this triggers the reconfiguration procedure $G\_Reconfigure()$ (table 4). This procedure reconfigures generates a new deployment configuration at level n. In this function we use an application m that associates to each couple of a deployment configuration $A_{n,p}$ and a reconfiguration event e a new deployment configuration $A_{n,q}$ (table 4 line 5 and 6). When a reconfiguration event e occurs, this triggers a

reconfiguration procedure $G\_Reconfigure()$ (table 4). This procedure reconfigures the current deployment configuration and generates a new deployment configuration at level n by the application of a sequence of grammar productions $p_l \dots p_k$. In this function we use an application m that associates to each reconfiguration event e a sequence of grammar productions $p_l \dots p_k$. The sequence of grammar productions $p_l \dots p_k$ reconfigures the current deployment configuration $A_{n,i}$ and generates a new deployment configuration $A_{n,j}$.

TAB. 4 – *The Graph Reconfiguration Algorithm*

```
1 G_Reconfigure()
2 {
3   Let E denote the set of the reconfiguration events
4   Let A_{n,i} ∈ 𝔸_n, i ∈ ℕ be a given deployment configuration
5   Let m : E ⟶ P
        e ⟼ p_l ... p_k
6   ∃ a sequence p_l ... p_k = m(e)
7   Apply m(e) on A_{n,i}
8   Let A_{n,j} the result of the application of m(e)
}
```

After the reconfiguration of the initial deployment configuration, the adaptability algorithm selects the optimal deployment configuration at level $n-1$ (table 1 line 11) that implements the deployment configuration at level n obtained through $G\_Refine()$.

The procedure $Distance\_based\_Selection()$ (table 5) minimizes the distance between two deployment configurations at level $n-1$ both implementing the correspondent deployment configurations at the level n (table 5 line 6) and the $Contextaware\_Cost()$ (table 5 line 8) according to the context values and characteristics (table 5 line 3).

The adaptability algorithm loops (table 1 line 5) and waits for another reconfiguration event that will trigger a new reconfiguration at level n.

4 Case Study

In this section we detail the abstraction levels for adaptability management. To expose the targeted problems and concepts and to show the usefulness of the graph-based models, we consider the example of ERCMS. We introduce this example and give two different execution steps and develop the related scenarios.

4.1 Considered Abstraction Levels

For adaptability management, we consider two main abstraction levels allowing component-to-component and service-to-service architectural properties to be described. From a communication point of view they represent respectively, the Middleware layer and the upper service layers. In the following, we will refer to these two levels as: the Middleware-level (M-level) and the Service-level (S-level).

TAB. 5 – *The Distance-Based Selection Algorithm*

1 Distance_based_Selection()
2 {
3 Let C denote the context attributes (e.g the available power,the available memory)
4 Let $A_{n,q}$ the result of the reffinement of $A_{n,p}$
5 Let $A_{n-1,p}$ the current mapping at level $n-1$ of $A_{n,p}$
6 Select $S_1 = \{A_{n-1,k} \in \mathbb{A}_{n-1,q}, k \in \mathbb{N}$ such that:
$Relative\_Cost(A_{n-1,p}, A_{n-1,k}) \leq Relative\_Cost(A_{n-1,p}, X), \forall X \in \mathbb{A}_{n-1,q}\}$
7 if $card(S_1) > 1$
8 Select $S_2 = \{A_{n-1,k} \in S_1, k \in \mathbb{N}$ such that:
$Contextaware\_Cost(A_{n-1,k}, C) \leq Contextaware\_Cost(X, C), \forall X \in S_1\}$
9 if $card(S_2) > 1$
10 Select any deployment configuration from S_2
11 }

The S-level constitutes the highest communication level. It describes the services and their associated requirements and constraints provided by communicating software entities exchanging high level information. Such constraints may represent the incapacity of a given device to host containers or components. A requirement may represent a communication priority between or from a given group of users. Requirements and constraints may change dynamically depending on the supported cooperative activity and its evolution. S-level entities can be instantiated in different ways. For instance, they can represent the different roles the human participants may have within the considered activity. For group communication activities, depending on its role in the activity, each participant has to perform a set of given functions. These functions are dynamically assigned to the participants according to the evolution of the activity, considering their skills.

The M-level is viewed as a component-to-component communication level aiming at supporting a given S-level architecture, considering resource-related constraints. Three roles are distinguished: "event producers" (EP), "event consumers" (EC) and "channel managers" (CM). Multiple producers and consumers are associated together by the same channel manager.

4.2 ERCMS Example

ERCMS-like activities involve structured groups of participants communicating to achieve a common mission (e.g. save human lives, fight against a huge fire...).

The scenario involves different categories of mobile actors that carry different types of communication devices. We distinguish human actors that may be professional actors with a professional and specific communication device or occasional actors that carry a mobile device (e.g PDAs, Phones). We distinguish also, robot actors like planes, helicopters and ground robots. For all the actors the communication system must deal with unexpected or expected evolution of user needs or the changes due to device/network constraints.

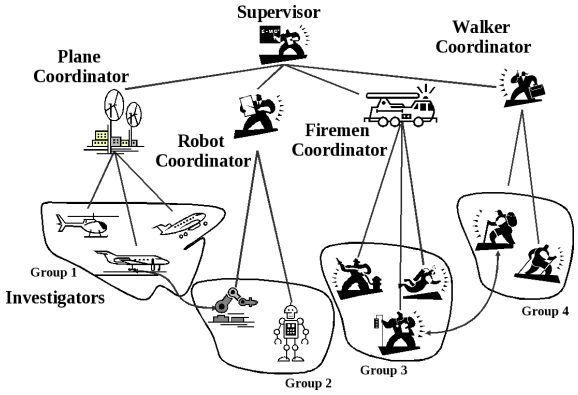

FIG. 1 – *ERCMS actors*

We define three different participant roles : The supervisor of the mission, the coordinators, and the field investigators. Each group of investigators is supervised by a coordinator (Figure 1).

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 achieved by coordinators and investigators. The supervisor is the entity which supervises the whole mission. The supervisor waits for data from coordinators who analyse the current situation of the mission. The supervisor has permanent execution resources and high communication and storage capabilities.

Coordinators that are attached to the supervisor, 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 and hardware capabilities. In Figure 1, we can distinguish the robot coordinator, the plane coordinator, the firemen coordinator that manage professional actors and The walkers coordinator (located in a watch tower) that manage occasional actors.

The investigator's functions include exploring the operational field, observing, analyzing, and reporting about the situation. Investigators also act for helping, rescuing and repairing.

We can distinguish different steps during the mission We give the two most representative execution steps: "Exploration step" (for the localization and the identification of the crisis situation) and "Action step" (after the identification event). Initially, all investigator groups are in the "Exploration step". Investigators provide continuous feedbacks D to the coordinator; they also provide periodical feedbacks P .The coordinator sends continuous feedbacks P to the controller.

When an investigator discovers a critical situation, its group has to be reconfigured to face this new situation. It moves to another execution step called an "Action step". The investigator that discovers the critical situation keeps sending both observations D and analysis P to the coordinator. The investigator also provides analysis P to the other investigators of its group.

Other investigators report analysis P to coordinator on the basis of observations D transmitted by the critical investigator. The coordinator continue sending analysis P to the investigator.

When the critical situation is resolved, the investigation group comes back to the exploration step.

5 The Graph Grammar-Based Rule-Oriented Implementation

To illustrate our approach, we use here the ERCMS case study. We define three graph grammars to implement architecture reconfiguration at the service level (n) and architecture refinement for mapping S-level onto M-level.

In both cases, the proposed grammars generalize the case study by considering a variable number of investigators. The refinement graph grammars $GG_{S\rightarrow M,exp}$ and $GG_{S\rightarrow M,act}$[3], for a given configuration $A_{n,i} \in \mathbb{A}_n, i \in \mathbb{N}$ at the service level, produce the set of configurations $\mathbb{A}_{n-1,i}$ that can implement $A_{n,i}$ at the middleware level $(n-1)$. These graph grammars are an implementation of the refinement algorithm (table 2). We present also a graph grammar $P_{exp\rightarrow act}$ that allows transforming the architecture to handle evolving of communication requirements from the exploration step to the action step. This graph grammar implements the reconfiguration algorithm (table 4). For the ERCMS case study, we define the context C (table 3) as the percentage of the available energy on each node (L_E) and the percentage of the available memory on each node (L_M). To illustrate our proposal, we also define the functions $Weight()$, $Contextaware\_Cost()$ and $Relative\_Cost()$ used in table 3 and table 5.

The function $Weight()$ is defined as the number of transformations that refine a given deployment architecture from level n to level $n-1$.

For the function $Contextaware\_Cost()$, we proceed in two steps. First, for each deployment node $Node_i$, for a given deployment configuration $A_{n-1,p}$, we calculate an evaluation

$$V_i = \frac{\alpha L_E + \beta L_M}{\alpha + \beta}$$

where the values α and β are weights that give the importance degree to be associated with each factor (L_E,L_M). For instance, if we know that for a specific node the memory saturation level is the most important factor, we set β to a value higher than α. Second, we calculate

$$Contextaware\_Cost(A_{n-1,p}, C) =: \frac{1}{N}\sum_{1}^{N} V_i$$

where the value N is the number of nodes in the deployment configuration $A_{n-1,p}$.

For two given deployment configurations $A_{n-1,q}$ and $A_{n-1,k}$, $Relative\_Cost(A_{n-1,q}, A_{n-1,k})$ corresponds to the minimum redeployment actions for switching from $A_{n-1,q}$ to $A_{n-1,k}$.

The algorithm of table 6 implements our abstract algorithm table 1. In the first step we capture the context: the percentage of the available energy (L_E) and the percentage of the

[3] M stand for middleware, exp stand for Exploration step and act stand for Action step

TAB. 6 – *The Adaptability Algorithm Implemented by Graph-Grammars*

Select $A_{n,0}$ the initial deployment configuration
For all hosting nodes $\in A_{n,0}$ capture the percentages of
available energy and of available memory: (L_E) and (L_M)
Apply $GG_{S\rightarrow M,exp}$ to refine $A_{n,0}$ into $\mathbb{A}_{n-1,0}$
Apply $Weight\_Selection()$ to select the optimal deployment configuration $A_{n-1,0}$
OnEvent investigator N discovers a critical situation
(change from the exploration step to the action step)
Apply $P_{exp\rightarrow act}$ to reconfigure $A_{n,current}$ into $A_{n,new}$
Apply $GG_{S\rightarrow M,act}$ to refine $A_{n,new}$ into $\mathbb{A}_{n-1,new}$
For all hosting nodes in $A_{n,new}$ capture (L_E) and (L_M)
Apply $Distance\_based\_Selection()$ to select the optimal configuration $A_{n-1,new}$

available memory (L_M) of each node in the initial deployment configuration $A_{n,0}$. After the refinement and the application of the weight-based selection algorithm, we obtain the optimal deployment configuration $A_{n-1,i}$ that implements $A_{n,0}$ at the middleware level. After a reconfiguration, we obtain the deployment configuration $A_{n,1}$. In the last step, we refine and apply the distance-based selection procedure. We obtain the optimal deployment configuration $A_{n-1,1}$ that implements $A_{n,1}$ at the middleware level and that represents an efficient adaptation for the event that triggers the reconfiguration.

5.1 The Graph Grammar for Architecture Reconfiguration

Following the commonly used conventions, we consider that vertices represent communicating entities (e.g. services, components) and edges correspond to their inter-dependencies (e.g. communication links, composition dependencies). For our study, we consider an architecture instance that includes a coordinator (Coord) managing four investigators (Inv). The graph edges are labelled by the exchanged data types (D/P). Each participant has three attributes: the identifier, the used data type and the deployment node.

In the following, we provide an example of graph grammar for our case study to implement an architecture reconfiguration. The presented rewriting rule allows transforming the architecture to handle evolving of communication requirement from the exploration step to the action step. The graph grammar is reduced to a single production grammar $P_{exp\rightarrow act}$ which is parametrized by the investigator identifier (here, noted N) that has discovered the critical situation. The architecture is transformed by splitting the communication channels between the coordinator and the other investigators into a communication channel of type P between these investigators and the controller and another communication channel of type D between them and Node N.

Figure 2 gives an example of the application of this rule to move the architecture from the exploration step to the action step (from $A_{n,0}$ to $A_{n,1}$). Investigator i2 plays the role of the critical investigator N. $P_{exp\rightarrow act}$ allows the correct generation of the communication channels.

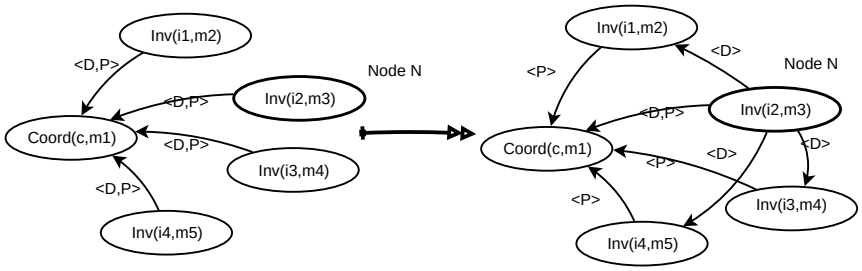

FIG. 2 – *Reconfiguration from the Exploration Step to the Action Step*

5.2 The Graph Grammars for Architecture Refinement

In this section, we give the graph grammar addressing the refinement of a given architecture of the S-level in all possible architectures of the M-level, during the exploration step and the action step. Since this graph grammar refines a given S-level architecture into M-level architectures, its non-terminal nodes are S-level entities while terminals nodes are M-level entities. $GG_{S\rightarrow M,exp}$ allows implementing this refinement in the exploration step. Figure 3 gives

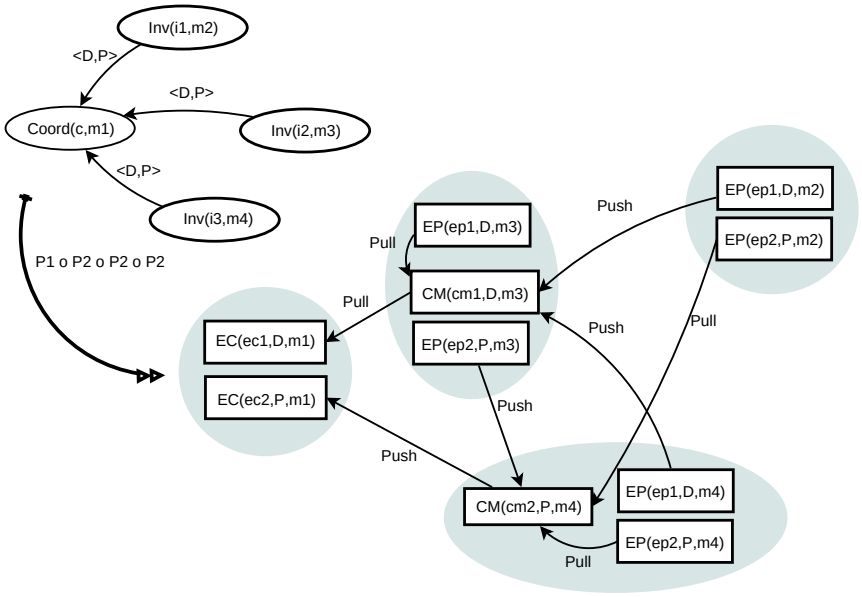

FIG. 3 – *Using $GG_{S\rightarrow M,exp}$ to achieve the refinement from S-level to M-level during exploration step*

the refinement generated by $GG_{S\rightarrow M,exp}$ of $A_{n,0}$. The application of the sequence "$p1$; $p2$; $p2$; $p2$" generates a configuration containing only terminal nodes (i.e. nodes belonging to the

M-level). Production $p1$ allows the refinement of the pattern consisting of the coordinator, and the two investigators whose matching host the channel managers $CM1$ and $CM2$. Production $p2$ allows refining the pattern for the other investigators. The result configuration is considered as a refinement of the initial architecture exploration step.

We also give the graph grammar addressing the refinement of any architecture of the S-level in all possible architectures of the M-level, during the action step. Since this graph grammar transforms S-level architecture into M-level architectures, its non-terminal nodes are S-level entities while terminals nodes are M-level entities.

For the coordinator, we generate an event producer and channel managers to communicate with the investigators. $p1$ deploys channel managers on the coordinator node. Figure 4 gives the refinement generated by $GG_{S\rightarrow M,act}$ of $A_{n,1}$ by the application of $p4$. We can generate an alternative refinement by the application of the sequence $p1$; $p5$. This sequence of applications generates configurations containing only terminal nodes (i.e. nodes belonging to the M-level). This configuration is considered as a refinement of the initial architecture in the action step.

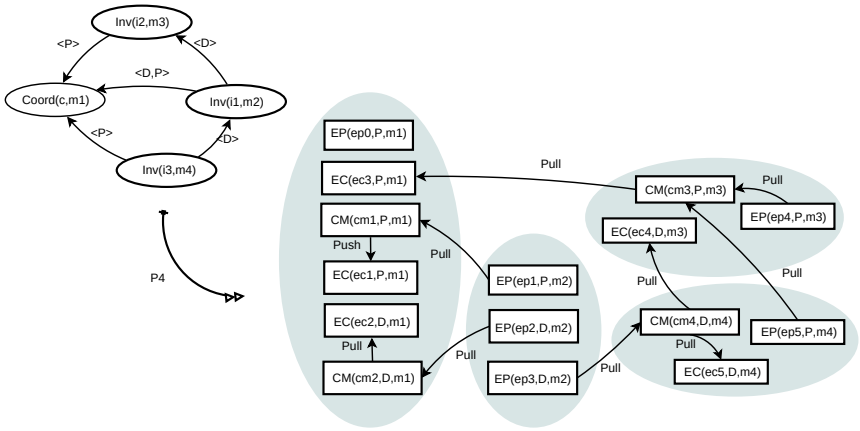

FIG. 4 – *Using $GG_{S\rightarrow M,act}$ to achieve the refinement from S-level to M-level during exploration step*

6 Conclusion

In this paper, we have presented a multi-level architectural reconfiguration approach for implementing context-aware adaptation procedures. We have shown how describing deployment architectures as graphs and how using graph grammars allows a rule-based management model to be elaborated. The rules handle both transforming a given architecture within the same level and architectural mappings between different levels. Using such a rule-based approach allows correct architectural reconfigurations to be characterized and used either offline to help implementing the decision process, or on-line 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 trans-

formation engine, we have simulated and validated our rules with successful scalability tests. Current work addresses real experiments on top of ubiquitous networks started within the context of the European project UseNet. Future research actions include defining the complete adaptation process and refining the elaborated selection functions.

References

Allen, R. and D. Garlan (1997). A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology 6*(3), 213–249.

Chang, F. and V. Karamcheti (2000). Automatic configuration and run-time adaptation of distributed applications. In *HPDC*, pp. 11–20.

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

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

Exposito, E., P. Senac, and M. Diaz (2003). FPTP: the XQoS aware and fully programmable transport protocol. In *Proc. The 11th IEEE International Conference on Networks (ICON'2003)*, Sydney, Australia.

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

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

Ganek, A. and T. Corbi (2003). The dawning of the autonomic computing era. *IBM Systems Journal 42*(1), 5–18.

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

Hirsch, D., P. Inverardi, and U. Montanari (1999). Modeling software architectures and styles with graph grammars and constraint solving. In 1^{st} *Working IFIP Conference on Software Architecture*, San Antonio, TX, USA, pp. 127–142. ISBN 0-7923-8453-9, Kluwer.

LeMetayer, D. (1998). Describing software architecture styles using graph grammars. *IEEE Transactions On Software Engineering 24*(7), 521–533.

Nasser, N. and H. Hassanein (2004). 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.

Sun, J. Z., J. Tenhunen, and J. Sauvola (2003). Cme: a middleware architecture for network-aware adaptive applications. In *Proc. 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Volume 3, Beijing, China, pp. 839–843.

Wu, D., Y. T. Hou, W. Zhu, Y.-Q. Zhang, and J. M. Peha (2001). Streaming video over the internet: approaches and directions. *IEEE Trans. Circuits Syst. Video Techn. 11*(3), 282–300.