Implementing a rule-driven approach for architectural self configuration in collaborative activities using a graph rewriting formalism

Ismael Bouassida Rodriguez, Mohammed Karim Guennoun, Khalil Drira, Christophe Chassot, Mohamed Jmaiel

Implementing a rule-driven approach for architectural self configuration in collaborative activities using a graph rewriting formalism

Ismael Bouassida Rodriguez
LAAS-CNRS ; Université de Toulouse ;
7, avenue du Colonel Roche,
F-31077 Toulouse, France
ibouassi@laas.fr

Karim Guennoun
Ecole Hassania des Travaux Publics
KM 7, Route D'EL JADIDA,
Box 8108,
Oasis, Casablanca, Morroc
guennoun@ehtp-ac.ma

Khalil Drira
LAAS-CNRS ; Université de Toulouse ;
7, avenue du Colonel Roche,
F-31077 Toulouse, France
khalil@laas.fr

Christophe Chassot
LAAS-CNRS ; Université de Toulouse ; INSA
7, avenue du Colonel Roche,
F-31077 Toulouse, France
chassot@laas.fr

Mohamed Jmaiel
National Engineering School of Sfax, Redcad
Route de la Soukra Km 3.5
Box W 3038 Sfax, Tunisia
mohamed.jmaiel@enis.rnu.tn

ABSTRACT

Architectural adaptation is important for handling self- configuring properties of autonomic distributed systems. It can be achieved by model-based management of dynamic architectures. Describing dynamic architectures includes defining rules for reconfiguration management. Within this research context, several works have been conducted using formal specification to handle this complexity. Graph and graph rewriting-based approaches showed, through many studies, their appropriateness to tackle architectural adaptation problems. However, scalability of such approaches remains an open issue and has been rarely explored. In this paper, we investigate this issue. We introduce a graph-based general approach for handling of dynamic architectures, and we illustrate it within a scenario of collaboration support in Crisis Management Systems. We elaborate the formal models for dynamic architecture management. Using the French Grid GRID5000, we conducted an experimental study to assess the scalability of the elaborated models.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Languages*; D.2.9 [**Software Engineering**]: Management—*Software configuration management*; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

CSTST 2008 October 27-31, 2008, Cergy-Pontoise, France

General Terms

Design, Management

Keywords

Autonomic computing, Rule-driven, Graph rewriting

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 [18]. Providing such solutions for distributed software systems supporting group communication requires dynamically managing evolving group membership and dynamically connecting deployment nodes. It also requires dynamically distributing software services and components on such remotely interconnected deployment nodes.

For a number of group communication-based applications, to anticipate reconfiguration is important. In order to be applicable in different situations, one has also to ensure tractability of the elaborated solutions when changing from some to thousands users, nodes, components and services.

Providing generic and scalable solutions for automated self-reconfiguration in group collaboration support systems can be driven by rule-based reconfiguration policies. We elaborate a graph-based modelling approach and structural models that may 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 grammar rewriting. Architectural reconfiguration is handled by defining architectures as graphs where vertices correspond to deployment nodes, software services and their internal components. It provides 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. 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.

To illustrate the proposed models and their transformations, we consider a case study of crisis management systems (CMS) involving several cooperating participants having different roles and functions. Related graph-based models are provided in section 4. Section 5 gives experimental results and analysis comments for scalability and performance of implemented models. 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.

2.1 Adaptability and adaptation

Adaptation is the operation to make 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 ubiquitous computing and context-awareness, adaptation is extended by the concept of adaptability that characterizes a system capability to change its behaviour to improve its performance or to continue its role in different environments. On the usefulness of adaptation, M. Satyanarayanan said in [36] "adaptability is necessary when there is a significant disparity between supply and demand of a resource."

2.2 Goals

After developing an information system, many reasons can lead to adapt it. The reasons for this adaptation can be for corrective, evolutional or perfective purposes [22]. 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. For the 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 accomplished by adding one or more modules to provide new features or modifying existing modules to enrich their functionality while keeping the same application architecture. The objective of the perfective 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.

2.3 Classification

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

2.3.1 Design vs run-Time

Two different adaptability views may be distinguished: the design time adaptability [11, 16, 14] and the run time adaptability [7, 3].

For the first view, we can find design support tools handles the application development cycle and optimizes the resources for example. For the run time adaptability [17] presents several adaptation techniques which use proxy services, change model of interaction and reorganize application structure.

2.3.2 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 [4].

2.3.3 Behavioural vs architectural

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 [39, 31] 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 [1, 13] 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.

2.4 Objects of adaptation

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

2.4.1 User interface adaptation

User interface adaptation consist in producing Human-Computer means that can be deployed and used on different types of terminals and which meet the user's preferences. Major existing work in user interface adaptation is based on models that describe the different aspects of the interaction between humans and machines. These models are implemented in different XML or UML languages like UMLi [33] and XIML [34]. 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 was adopted by [29] to produce adapted Web pages to different terminals starting from an XML description. In this kind of adaptation, style sheets are used to specify replacement rules of XML tags by

scripts that can be directly used by the target device. The second approach consist in generating user interfaces code starting from a high level description which is completely independent from the target programming language of the user interface.

2.4.2 Content adaptation

Since the appearance of pervasive systems, 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 [27], image transcoding [38], 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 [28]. However, the usual complexity of adaptation processing hampers the wide adoption of this approach [32]. Indeed, the client's terminal has generally very limited computing capacity, power and storage.

For 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 the data.

2.4.3 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 Web Services or COM/ DCOM (Component Object Model) [19] of Microsoft, the EJB (Enterprise Java Beans) [24] of Sun Microsystems or CCM (CORBA Component Model) [30] of OMG (Object Management Group).

2.4.4 Middleware adaptation

Other frameworks are proposed to provide adaptability for the middleware level. In [29], an adaptive framework for supporting multiple classes of multimedia services with different QoS requirements in wireless cellular networks is proposed. [37] proposes CME, a middleware architecture for service adaptation based on network awareness. CME is structured as the software platform both to provide network awareness to applications and to manage network resources in an adaptive fashion.

2.4.5 Transport adaptation

At the transport level, [15] provide 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.

2.5 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 there general ADL types: formal ADLs like graph grammars [20] and Petri nets [26], semantic ADLs using ontologies [40] and technical ADLs using XML deployment languages in general [10]. The technical ADLs can be proprietary or implementing the formal and the semantic architecture description models. These ADLs are used to guarantee the architecture evolving and correctness during the different predictable and unpredictable changes in the systems environment. The necessary actions to achieve such adaptations are specified in rules according to the application runtime context. [6] is an example of these approaches defining a complete model based architecture adaptation at the Service, content and user interface levels. [8] presents another model based method using graph grammars to adapt cooperative information systems to situation changes in the communication level.

Mobile networked systems for the support of group-wide activities have to provide dynamic adaptability for run-time and design-time changes. Such changes are induced by evolving requirements of the supported activities and by evolving resource constraints.

Designing and implementing self adaptive communicating systems to support such emerging activities is complex. Mastering this complexity may be achieved by adopting model-based design approaches associated with automated management techniques for dynamic architectural and behavioral adaptability management.

The works exposed allow clarifying the different problems related to self-adaptability. The proposed solutions have different features related to the targeted goals (e.g. QoS, security...), the considered layers (e.g. Application layer, Transport layer...), the adaptation actions (e.g. rate control, congestion control...), the adaptation properties (e.g. architectural, behavioural...), and the way to manage the adaptation and its autonomy.

To face the different kinds of requirements and constraints evolving, behavioral and architectural adaptability is required at several levels simultaneously. This typical cross layering problematic requires managing coordination without which it can lead to performances way below the targeted ones.

For group communication-based cooperative activities, ab-

straction levels have to be identified. Adaptation at the highest levels should be guided by the evolving of the activity requirements. Adaptation at the lowest levels should be driven by the changes due to device/network constraints.

Designing and implementing self-adaptive communicating systems is a complex task. To handle this complexity, several studies 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 [23, 2]. 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 architecture aspects [25, 21]. 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.

3. CASE STUDY

To expose the targeted problems and concepts, we consider the example of crisis management systems (CMS). In the sequel, we introduce this example and give two different execution steps and some related scenarios.

For CMS-like activities, collaboration is based on information exchange between mobile participants collaborating to achieve a common mission. A CMS team is composed of a number of participants having different roles: The controller of the mission, coordinators, and field investigators. Each group of investigators is supervised by a coordinator (Figure 1). Each participant is associated with an identifier, a role

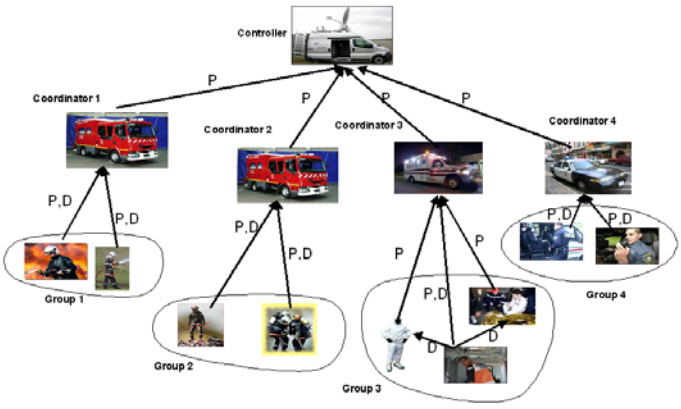

Figure 1: CMS architecture

and the devices he/she uses. Each participant performs different functions. The controller's functions include monitoring and authorizing/managing actions to be done by coordinators and investigators. The controller is the entity which supervises the whole mission. The controller waits for data from his coordinators who synthesize the current situation of the mission. The controller has permanent energy resources and high communication and CPU capabilities. Coordinators that are attached to the controller, have to manage an evolving group of investigators during the mission and to assign tasks to each one 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. The investigator's functions include exploring the operational field, observing, analyzing, and reporting about the situation. Investigators also act for helping, rescuing and repairing.

Functions performed by investigators include generating Descriptive data (D) of the exploration field and Produced data (P) feedbacks to the controller. Feedbacks D are Descriptive data; they are transmitted by means of audio/video and real time text messaging. Feedbacks P are Produced data; they express the analysis of the situation by an investigator. They are transmitted by means of audio and real time text messaging.

The controller's function consists in supervising the whole mission, i.e. deciding actions to be performed from the analysis of the observation feedbacks D transmitted by the coordinators. The scenario is divided into two steps: Investigation step and Action step. Initially, all investigator groups are in the "exploration step" where Investigators provide continuous feedbacks D to the coordinator; they also provide periodical feedbacks P and the coordinator sends continuous feedbacks P to the controller.

When an investigator meets a critical situation, the architecture of its related group has to be reconfigured to face this new situation. The group has to move to another execution step called an "action step" where The investigator that discovers the critical situation keeps sending both feedbacks D and P to the coordinator. It also provides feedbacks P to the other investigators of its group. Other investigators report feedbacks P to coordinator on the basis of feedbacks D transmitted by the critical investigator and the coordinator continue sending feedbacks P to the controller.

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

4. GRAPH-BASED MODELS FOR ARCHITECTURAL ADAPTATION

In this section, we introduce our general framework of models. We use models based on graph grammar and graph rewriting to handle the reconfiguration management.

Graph grammars constitute a powerful and very expressive formalism for style description. Moreover, theoretical work on this field provides formal means to specify and check constraints on these architectures [35].Inspired from Chomsky's generative grammars [9], 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 the set of terminal vertices, and P 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 graph grammar production structure and the mechanisms used for the specification of their execution. 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

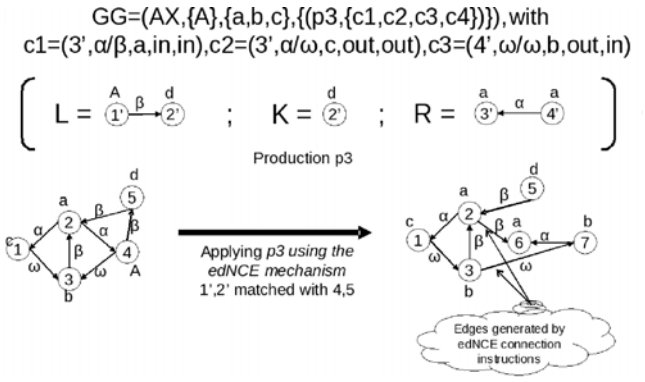

Figure 2: Combining DPO and edNCE

a Double PushOut (DPO) production [12] and where C is a set of connection instructions. The instructions belonging to C are of the edNCE type [35]. 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 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[1] of and d-neighbours[2]. This edge is introduced following the direction indicated by d' and labelled by q.

An example is given in Figure 2. 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, we consider that vertices represent communicating entities (e.g. services, components) and edges correspond to their related interdependencies (e.g. communication links, composition dependencies).

An example of an architecture instance is given in figure 3. This instance includes, in addition to the controller (Cont), two coordinators (coord) that manage respectively three and two investigators (Inv). The graph edges are labelled by the exchanged data types (D/P). Each participant have two attributes: the identifier and the deployment host. For example an investigator node is represented by a label like that $inv(identifier, deploymenthost)$. Architectural reconfiguration management consisting in transforming the system architecture evolving situation is formally ruled using graph rewriting techniques.

[1]p-neighbours of a vertex n are all vertices n' such that there exists an edge labelled by p which connects n and n'.
[2]In-neighbours if $d = in$ and out-neighbours otherwise.

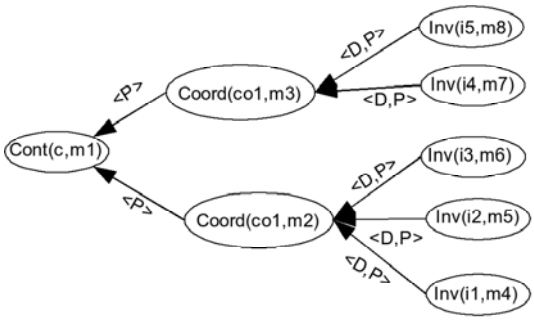

Figure 3: Architecture instance (Exploration step)

5. THE GRAPH GRAMMAR FOR ARCHITECTURE RECONFIGURATION

In the following, we provide an example of graph grammar for our case study to implement architecture reconfiguration.

5.1 Managing changes in activity organisation

The presented rewriting rule allows transforming the architecture from for moving the exploration step to the action step. The graph grammar is reduced to a single production grammar $P_{exp \to act}$ (table1) which is parameterized by the investigator identifier (here, noted A) who 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 A into another communication channel of type D. Figure 4 gives an example of the application of this rule to move the architecture from the exploration step to the action step. Investigator i2 plays the role of the critical investigator A. $P_{exp \to act}$ allows the correct generation of the communication channels. The connection instruction $ic1$ allows to change the labels of the coordinator to the investigators from $< D, P >$ to $< P >$. The connection instruction $ic2$ allows to connect the investigator A to the others investigator by an edge labelled $(< P >)$.

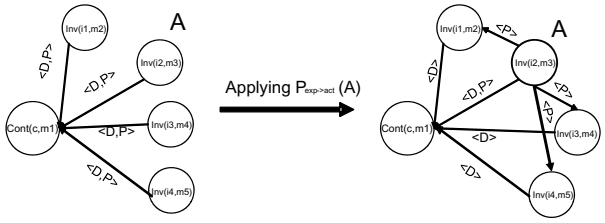

Figure 4: Reconfiguration from the exploration step to the action step

5.2 Managing changes in group membership

We elaborate a graph grammar for our case study to manage the group extension. In the exploration step, the investigators produce D-data continuously and P-data periodically. The coordinators send P-data periodically to the controller. In the graph grammar GG (table2) we have four productions.

$P_{exp \to act}(A) = (p1, C = ic1, ic2)$ with:
$p1 = (L = \{Coord(c, m1), Inv(A, m2), Inv(A, m2) \xrightarrow{<D,P>} Cont(c, m1)\}$,
$K = \{\ \}$,
$R = \{Coord(c, m1), Inv(A, m2), Inv(A, m2) \xrightarrow{<D,P>} Cont(c, m1)\})$
with: $ic1 = (Coord(c, m1), < D, P > / < P >, Inv, in/in)$
$ic2 = (Inv(A, m2), < D, P > / < D >, Inv, in/out)$

Table 1: Production grammar $P_{exp \to act}$

$GG = (AX, NT, T, P)$ with: $T = \{Inv(i, m3), Coord(co, m), Cont(c, m)\}$, and
$NT = \{Temp\}$, and $P = \{p_1, \ldots, p_4\}$

$p_1 = (L = \{AX\}$;
$K = \{\ \}$;
$R = \{Cont(c, m1), Coord(co, m2), Inv(A, m3), Temp)$,
$Coord(co, m2) \xrightarrow{<P>} Cont(c, m1)$, $Inv(i, m3) \xrightarrow{<D,P>} Coord(co, m2)\})$
$C = \{\ \})$

$p_2 = (L = \{Cont(c, m1), Temp\}$;
$K = \{Cont(c, m1), Temp\}$;
$R = \{Coord(co, m2), Inv(i, m3), Coord(co, m2) \xrightarrow{<P>} Cont(c, m1)$,
$Inv(i, m3) \xrightarrow{<D,P>} Coord(co, m2)\})$
$C = \{\ \})$

$p_3 = (L = \{Coord(co, m1), Temp)\}$;
$K = \{Coord(co, m1), Temp)\}$;
$R = \{Inv(i, m2), Inv(i, m2) \xrightarrow{<D,P>} Coord(co, m2)$
$C = \{\ \})$

$p_4 = (L = \{Temp\}$;
$K = \{\ \}$;
$R = \{\ \})$
$C = \{\ \})$

Table 2: Group membership change grammar GG

The productions p_1, p_2 and p_3 allow us to add new participants to the mission. The production p_1 allows to initiate the group by connection an investigator, a coordinator and a controller. The production p_2 allows to connect a coordinator and the edges necessary to that. According to the description of the case study, in the group, it is not possible to have a coordinator without an investigator. So the production p_2 adds an investigator to the added coordinator. The production p_3 allows to add an investigator to an existent coordinator. Such a propriety constitute the style of the architecture. The production p_4 allows to finish the generation process and eliminate the non-terminal $Temp$. The example shown in figure 3, can be generated[3] by this sequence: p_1;p_2;p_3;p_3;p_3;p_4. The existence of this generation path prove the consistency of this architecture instance.

6. EXPERIMENTAL STUDY

We conducted an experimental study of our graph rewriting system using the rules of the architecture adaptation models presented above.

These experiments have been achieved under grid GRID5000 [5], an experimental grid platform that interconnects several 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. We are studying scalability and limits of this model to investigate the tractability of an anticipation approach for adaptability.

The results, presented here, were obtained for graph grammar GG of section 4 and the reconfiguration rules executed to handle critical situation discovery (grammar production $P_{exp \to act}$ in table 1 of section 4). The experimental performance evaluation focuses execution time.

6.1 Experimentations for adapting architecture to organization changes

In this section we present, the results related to the execution of the $P_{exp \to act}$ rewriting rule. We are studying scalability and limits of our model-driven approach implementation.

We study the evolution of the execution time (figure 5), when the architecture description graph size growth. This corresponds to the application of production $p1$ of $P_{exp \to act}$ (Table 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).

6.2 Experimentation adapting architecture to group membership changes

We generate the first N members of this graph grammar corresponding to the N first consistent instances of the ar-

[3]This is not the unique generation path

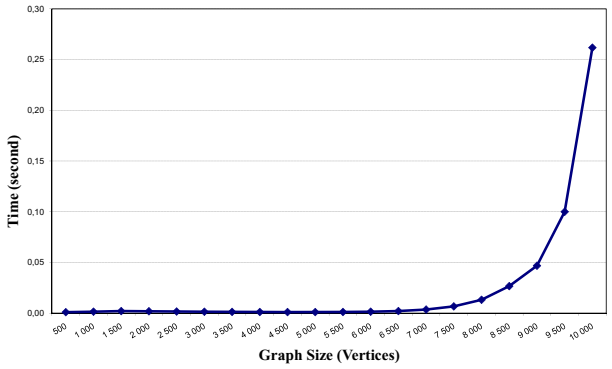

Figure 5: Execution time of applying $p1$ of $P_{exp \rightarrow act}$ wrt architecture graph size

chitecture. This corresponds to applying productions $p1$, $p2$ and $p3$ of GG (Table 2) whenever it is possible to obtain all consistent architecture graphs. The application of production $p4$ ensures the elimination of the non-terminal nodes in the graphs. The production applications include finding all graph matchings between the rules graph and transforming the architecture graph by adding nodes and edges. The production $p2$ adds two nodes and two edges. The production $p1$ adds one node and one edge.

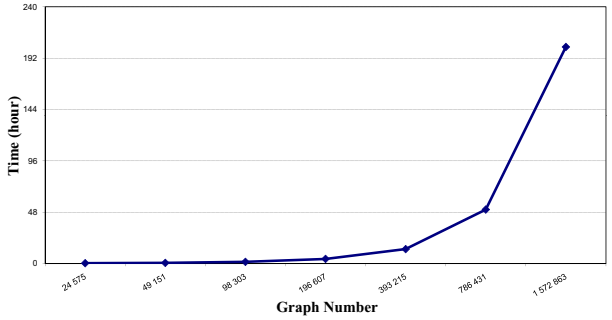

Figure 6: Execution time of applying $p1$, $p2$, $p3$ and $p4$ of GG wrt graph number

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 remains acceptable for design-time purposes. Using a unique grid node, the system generates almost the 400 000 first consistent instances in about 140 hours.

7. CONCLUSION

In this paper, we have studied a graph-based approach for implementing a model-based management of architectural reconfiguration. Our approach is useful when designing self-configuring software for large scale activities support. It supports scalable management of automated reconfiguration while allowing formal proofs to be developed for ensuring the consistency of the generated configurations. The models presented in this paper have been elaborated for the design of self-configuring Crisis Management Systems. The scalability study provided in this paper has been conducted to assess the tractability of our approach when dealing with large scale group applications.

8. REFERENCES

[1] In *IEEE Std 1471-2000, IEEE Recommended practice for architectural description of software-intensive systems*, pages i–23, 2000.

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

[3] K. Bade, E. W. D. Luca, A. Nürnberger, and S. Stober. Carsa - an architecture for the development of context adaptive retrieval systems. In K. van Rijsbergen, A. Nürnberger, J. M. Jose, and M. Detyniecki, editors, *Adaptive Multimedia Retrieval: User, Context, and Feedback*. Springer-Verlag, 2006.

[4] P. G. Bridges, W.-K. Chen, M. A. Hiltunen, and R. D. Schlichting. Supporting coordinated adaption in networked systems. pages 162–162, 2001.

[5] F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, and O. 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.

[6] T. Chaari, F. Laforest, and A. Celentano. Adaptation in Context-Aware Pervasive Information Systems: The SECAS Project. *Int. Journal on Pervasive Computing and Communications(IJPCC)*, 3(4):400–425, Dec. 2007.

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

[8] C. Chassot, K. Guennoun, K. Drira, F. Armando, E. Exposito, and A. Lozes. Towards autonomous management of qos through model-driven adaptability in communication-centric systems. *ITSSA*, 2(3):255–264, 2006.

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

[10] E. M. Dashofy, A. van der Hoek, and R. 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.

[11] A. B. 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.

[12] H. 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.

[13] 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.

[14] 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.

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

[16] 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.

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

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

[19] R. Grimes and D. R. Grimes. *Professional Dcom Programming*. Wrox Press Ltd., Birmingham, UK, 1997.

[20] D. Hirsch, P. Inverardi, and U. 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.

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

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

[23] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In W. Schafer and P. Botella, editors, *Proc. 5th European Software Engineering Conf. (ESEC 95)*, volume 989, pages 137–153, Sitges, Spain, 1995. Springer-Verlag, Berlin.

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

[25] D. L. Metayer. Describing software architecture styles using graph grammars. *IEEE Transactions On Software Engineering*, 24(7):521–533, July 1998.

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

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

[28] 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, Nov. 1999.

[29] N. Nasser and H. 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.

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

[31] Özgür B. Akan and I. 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.

[32] A. Perkis, Y. Abdeljaoued, C. Christopoulos, T. Ebrahimi, and J. F. Chicharo. Universal multimedia access from wired and wireless systems. *Circuits, Systems, and Signal Processing*, 20(3-4):387–402, 2001.

[33] P. Pinheiro da Silva and N. W. Paton. UMLi: The unified modeling language for interactive applications. In A. Evans, S. Kent, and B. 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.

[34] A. Puerta and J. 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.

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

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

[37] J. Z. Sun, J. Tenhunen, and J. 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.

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

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

[40] Y. Zhou, J. Pan, X. Ma, B. Luo, X. Tao, and J. 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.