



A Model Driven Approach for Automated Design of Context-Aware Autonomic Architectures

Emna Mezghani, Riadh Ben Halima, Ismael Bouassida Rodriguez, Khalil Drira

► To cite this version:

Emna Mezghani, Riadh Ben Halima, Ismael Bouassida Rodriguez, Khalil Drira. A Model Driven Approach for Automated Design of Context-Aware Autonomic Architectures. 2012. hal-00675352

HAL Id: hal-00675352

<https://hal.science/hal-00675352>

Submitted on 29 Feb 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Model Driven Approach for Automated Design of Context-Aware Autonomic Architectures

Emna Mezghani^{1,2,3}, Riadh Ben Halima^{1,2,3}, Ismael Bouassida Rodriguez^{1,2,3} and Khalil Drira^{1,2}

¹ CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France

² Univ de Toulouse, LAAS, F-31400 Toulouse, France

³ University of Sfax, ReDCAD, B.P.W, 3038 Sfax, Tunisia

{emna.mezghani,khalil.drira}@laas.fr, riadh.benhalima@enis.rnu.tn, bouassida@redcad.org

Abstract

In this paper, we propose a model driven approach automating the design of autonomic systems. We handle non-functional properties with a focus on managing QoS degradation for cooperative M2M applications such as health care. Nowadays, service delivery becomes close to end-users such as M2M applications, which are being incorporated into the existing infrastructure. The Remote Health Care System and specialized sensors for in-home patient monitoring are at the current forefront of new technologies. While there are benefits from technologies such as reducing costs and medical errors, associated architecture and communication infrastructure have to ensure care continuity and quality of service (QoS). In this paper, we propose a model driven approach which enables the generation of autonomic architecture from high level functional and non-functional requirements. Our work instantiates the Model Driven Architecture (MDA) approach. We elaborate formal rules using graph grammars to transform a high level functional requirements to an autonomic architecture implemented under GMTE, a Graph Matching tool¹. We generate a Service Component Architecture (SCA) at the MDA low level (PSM) to implement the architecture in different technologies such as EJB, JMS, SOAP, etc. The Remote Health Care System shows the feasibility and the efficiency of our approach.

1 Introduction

The complexity of M2M applications, the dynamism of the environments in which they are operated, and the need for adaptation to changes are growing rapidly. For that reason, designing autonomic architectures that adapt their services to dynamically context changes is crucial. In general, such architectures have to take into account the management of software both at design and at runtime. These software are deployed in M2M environments including mobile devices, desktop computers, sensors etc.

To design such architectures, designers have to specify functional and non-functional requirements, and should consider the execution context as environment constraints at runtime. Indeed, if a system is able to correctly monitor context, detects changes and reacts to them in a smart way, it will be able to automatically generate a new architecture that meets requirements.

Several research activities handle the dynamic reconfiguration and show different ways to adapt system to the context changes and user requirements. The variety of the solutions in the literature underlines the need to provide tools, models and techniques to build adaptation for cooperative M2M applications. Design approaches have evolved to take into consideration important context changes that increase the design complexity.

In order to tackle the complexity of designing autonomic architectures and the related adaptation issues, we propose a model driven approach that minimizes designer's tasks by auto generating autonomic architecture starting from an initial model describing functional requirements. Our model driven approach automates the generation of

¹The Graph Matching and Transformation Engine (GMTE), available at <http://homepages.laas.fr/khalil/GMTE>

autonomic architectures through several transformation and refinement rules between different MDA levels. These rules extend the initial given architecture by the autonomic capabilities such as: monitoring, analysis, planning and execution [4]. This leads to obtain significant gains on software scheduling and cost, especially for complex systems. To illustrate the proposed models and their transformations, we consider the Remote Health Care System for in-home patient monitoring as a case study. In one hand, it allows to reducing costs and medical errors. In the other hand, it has to be monitored and managed at runtime in order to ensure care continuity with better QoS. For that reason, we propose to apply our approach to automatically generate an autonomic architecture for this system.

This paper is organized as follows. Section 2 gives a survey on the related work and presents issues and challenges related to autonomic computing management. Section 3, we present the Remote Health Care System use case. Section 4 details our model driven approach for autonomic computing which is based mainly on IBM MAPE-K loop [17], MDA approach and Graph grammar. This approach is approved with the Remote Health Care System by detailing the different auto generated models. Finally, conclusions to this work are presented and research perspectives are identified.

2 Related work

To design autonomic system, IBM has proposed the autonomic computing paradigm. Several research activities have adapted this approach to build their autonomic systems in different domain. In this section, we detail different approaches implementing this paradigm, then we present challenges confronted when managing the autonomic computing.

2.1 Autonomic Computing

An autonomic computing paradigm has a mechanism that can trigger changes in the computing system behaviors and structures in order to satisfy user requirements. Furthermore, it is a collection of autonomic mechanisms that manage software systems with minimum human intervention and given high-level objectives fixed by the administrator [17].

Autonomic system is extended by the concept of adaptability that characterizes system's capability to change itself in order to improve its performance and to continue its role in different environments. This is based on context-awareness features that refer to the ability of the system to know its execution environment through monitoring and be able to react to its changes through reconfiguration actions [22]. According to Dey and Abowd [1], a system is sensitive to the context if it uses it to provide relevant information and services for the user, where the relevance depends on the task required by the user.

Several approaches have been developed to tackle the adaptation in different levels and domains to maintain systems functionalities. In [25], the authors present autonomic mechanisms for QoS-aware service composition and adaptation of end-to-end network service. They introduce a service provisioning framework based on the autonomic principles, covering a number of essential functions, like domain discovery, composition, domain-wide monitoring, adaptation etc. Through domain graph abstraction, they reduce the domain composition and adaptation problem to the classic multi-constrained optimal path problem. As the network condition changes over time, they develop a set of adaptation path algorithms which ensures QoS requirements while minimizing the overall cost. Authors of [23] propose CME, a middleware architecture for service adaptation based on network awareness. CME is structured in the form of a software platform which provides network awareness to applications, and manages network resources in an adaptive fashion. It uses a policy mechanism to facilitate the network management. Policies represent adaptation requirements to ensure at runtime. CME enables monitoring of security privileges by recording which stations communicate with each other.

Other frameworks are proposed to provide adaptability at the middleware level. In [20], an adaptive framework for supporting multiple classes of multimedia services with different QoS requirements in wireless cellular networks is proposed. In [18], 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 and extends them to enable their management at runtime

while using high level rules. The operation of the proposed framework is illustrated using a forest fire management application.

Becker and Giese [3] present an approach based on graph transformation techniques coupled with UML stereotypes in order to model autonomic systems. Reconfiguration, which is performed at runtime, is decomposed into three levels: goal management, change management and component control.

In [14], the authors present the SAFDIS framework. This framework enables the dynamic evolution of service-based architectures by providing all the functionalities of the MAPE model. They focused mainly on the analysis and the planning phases while taking into account different constraints. In the execution phase, SAFDIS considers the migration of services as reconfiguration action. The implementation is built for the OSGi platform, using iPOJO to manage the life-cycle.

In [19], the authors detailed their CEYLON framework that focuses on building autonomic managers based on service-oriented components. Ceylon architecture is based on two-layers: the resources management layer that contains a set of management components managed by the Task Manager; and the manager adaptation layer that contains different manager Strategies and implements the MAPE by supervising the Task Manager and develops a runtime model for adaptation.

2.2 Current issues and challenges

In this section, we highlight challenges in the autonomic computing management for communicating systems.

2.2.1 Design vs. Runtime

Two different views to enforce autonomic computing features may be distinguished: the design time [8, 11, 10] and the runtime managements [7, 2]. For the first view, we can find design support tools which handle the application development cycle and optimize the resources for example. On the other hand, the autonomic management at runtime may relay on several adaptation techniques which use proxy services, change model of interaction and reorganize application structure [12].

2.2.2 Local vs. Distributed

Autonomic computing may have a local or a distributed scope. Following a local approach, the components can be deployed on a single machine where changes are performed locally. Following a distributed approach, the components can be deployed on several distributed machines and synchronization problems between peer autonomic entities have to be managed [5].

2.2.3 Behavioral vs. Structural

The autonomic solutions suggested in the literature distinguish between behavioral and structural reconfigurations. The reconfiguration is behavioral (or algorithmic) when the behavior of the autonomic service can be modified, without modifying its structure. That's way, this reconfiguration is considered as a service state adjustment. It modifies the internal composition of system components in order to correct it. Standard protocols like TCP and specific protocols such as those presented in [24, 21] provide behavior-based autonomic mechanisms. Other approaches, like JavaPod [6] manages the adaptation by using object compositions. It performs the reconfiguration action as dynamic extension of methods with new implementations.

The reconfiguration is structural when the service composition can be modified [16, 13, 9] dynamically. The reconfiguration is done by creating and connecting, removing and disconnecting components during execution to meet the execution context involving, for example, variations of communication networks and processing resources or requirements related to mobility of users and cooperation structure modifications.

3 The Remote Health Care System Case Study

In this section, we motivate the need of autonomic architecture through a case study. First, we present the Remote Health Care System (RHCS) case study. Then, we bring out issues related to the non-use of autonomic features. After that, we propose and formalize a novel model driven approach which adapts systems at runtime.

RHCS represents a use case of an M2M application that supports the remote management of patient illnesses. It enables a smart management platform providing comprehensive tools for physicians and nurses to improve the quality of life of patients. This kind of applications is very crucial in the medical domain. RHCS is able to monitor remotely the patient health, control the medical treatments and trigger alarms when critical conditions are detected. The patient may be either at the clinic in case of critical diseases, or at home to reduce the traveling hassles with elderly and overweight patients.

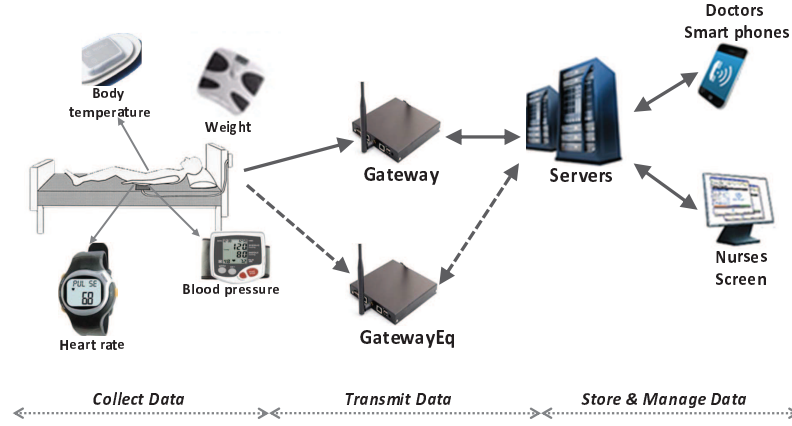


Figure 1. Remote Health Care System case study

Therefore, to get information about the patient's health, appropriate sensors have to be deployed. Generally, the patient wears one or more sensor devices that record health indicators such as blood pressure, body temperature, heart rate and weight as shown in Figure 1. It is also possible to use sensors located in the environment of the patient. This allows the monitoring of parameters related to the health condition of the patient. The health data are collected by these sensors and reported to servers periodically via M2M gateways. Wearable sensors often use wireless links to a gateway device that has WAN connectivity. A sensor device could communicate with gateways through different interfaces (Bluetooth, WiFi, 3G, etc.). For example, the blood pressure sends its collected data either through WiFi or Bluetooth interface. The device kind and the period of getting health information depend on the treated diseases. RHCS offers the continuous collection of patient data and the forwarding to servers that provide doctors or nurses instant access to these data through mobile phone or the Web.

One of the critical issues is to assure the timely and robust delivery of the life-critical health data. Especially in our case, in the resource-constrained wireless sensor networking environment. The RHCS must ensure care continuity to avoid vital signs information loss whatever the traffics even during patient mobility. Such system usually involves the usage of a variety of sensors that requires managing the QoS to maintain the availability to health data instantly. Different scenarios can occur and lead to degrade the performance of RHCS:

- **Scenario1:** The communication between servers and the gateway can take too long. This degradation is expressed by an increase in the response time of the gateway. When a large number of requests from physicians and nurses are sent via servers to get patient health data, the gateway may be overloaded and needs substitution by another equivalent (connected to the same devices "GatewayEq" as shown in Figure 1)
- **Scenario2:** At the level of the gateway, the interface managing the WiFi can be overloaded. This fact provokes an additional delay to the data delivery time. Balancing the traffic using the other available interfaces (for example interface managing Bluetooth) is a solution to avoid additional delay.

For each scenario, the RHCS has to be able to manage available resources to overcome degradations. We need specific components which monitor provided QoS and manage it at runtime through reconfiguration actions: substitution and load balancing.

4 A model Driven Approach for Autonomic Architectures

In this section, we describe our model driven approach to design autonomic architecture. It starts from a high level requirements and ends to an SCA architecture that manage context changes.

4.1 Overview of the proposed model

We present a generic approach which allows an automated generation of autonomic architecture starting from an abstract specification of the functional requirements and ending to an SCA architecture while following MDA. This approach is shown in Figure 2.

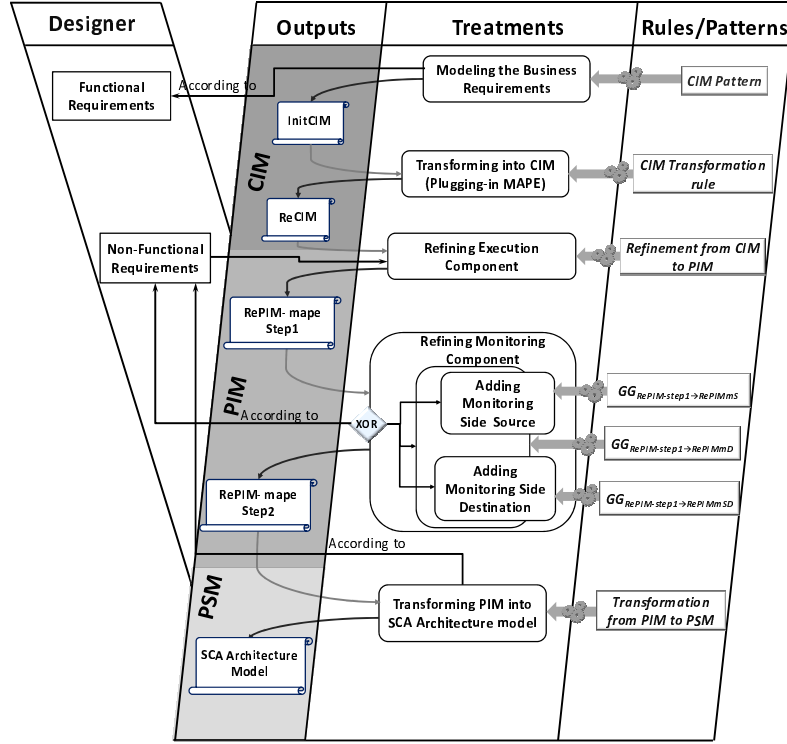


Figure 2. Model Driven Approach for Context-Aware Autonomic Architectures Design

In the left branch of the model (Design branch), the designer focuses on specifying the functional requirements. The designer creates an instance of the system architecture according to the style. In addition, the designer specifies the non-functional requirements. While the right branch, as shown in Figure 2, is composed of three levels as described by MDA: CIM (Computation Independent Model), PIM (Platform Independent Model) and PSM (Platform Specific Model). In each level, we have elaborated a set of automated treatments using patterns and rules able to refine and transform -intermediate- input models. In the bottom level, we provide an SCA based architecture, as final output (see Figure 7).

In our approach, the final architecture is obtained by applying automatically successive Graph grammar productions the initial CIM. We use the graph grammar also to model the CIM and PIM levels as graph model while we use the SCA to model the PSM level.

4.2 CIM level Description

It is the most abstract model in MDA. It represents the system structure and hides the complexity. In our approach, we distinguish two sub-levels: the InitCIM (Initial CIM) and ReCIM (Reconfiguration enabled CIM) as

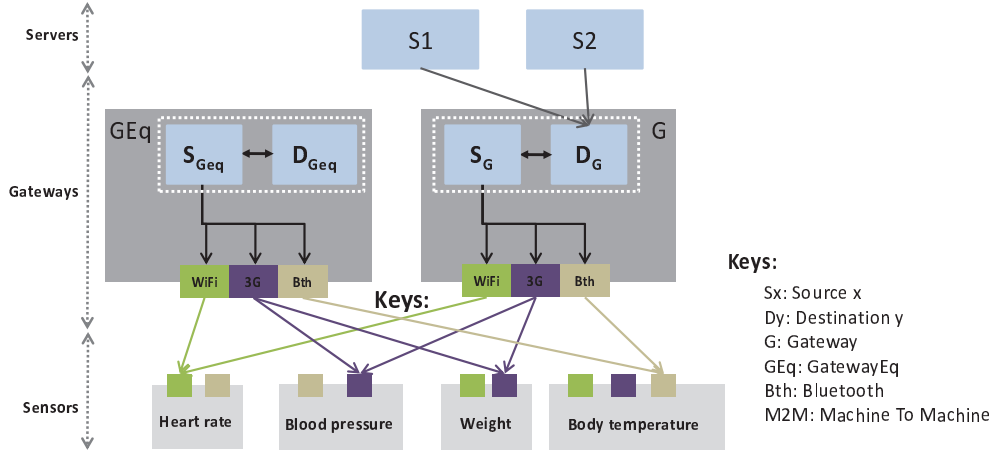


Figure 3. Remote Health Care System software components

represented in Figure 2.

4.2.1 InitCIM Description

The InitCIM models the functional requirements which are described as communications between system's entities (sources and destinations as described in Figure 3), using the graph model. In the case of the Remote Health Care System, communications between entities are as following:

- M2MServers communicate with gateways. Therefore in the InitCIM, M2MServers are represented as "Source" and gateways are represented as "Destination".
- Gateways communicate with devices. These communications are based on different wireless technologies (Bluetooth, Wifi, 3G) that connect gateways to sensors, as shown in Figure 3. In the InitCIM, we consider the gateway communication interfaces as destinations. The equivalent destination is provided by the designer.

Initially, the M2MServers -acting as sources- are connected to the gateway "G". An equivalent gateway (GEq in Figure 3) is available as an equivalent destination since it is connected to the same devices and offers the same service as the already connected one, in our case the gateway "G", ("connect_to_device").

As output of this step, we obtained the graph describing the InitCIM as shown in Figure 4. We distinguish two types of nodes: one has an *id* and a *type* corresponding to "Source" or "ConcreteDestination", the other type has an *id*, a *type* equals to "EqDestination", and a *reference* to the equivalent destination. The source and destination nodes are linked via the "communicate" edge that models the functional communication.

4.2.2 Transformation InitCIM into ReCIM

The first model auto-generated by our approach is the ReCIM. It is obtained by applying the graph grammar $GG_{InitCIM} \rightarrow ReCIM$ on the InitCIM. This step focuses on plug in the MAPE-K loop [15] -as a black box- on the InitCIM model to enable autonomic features on the designed system. The MAPE-K loop is plugged between each destination and its sources. In our case study, the MAPE-K loop is plugged between M2MServers and Gateways and also at the communicating interfaces in the gateways as shown in Figure 5. Despite the transformation made to the InitCIM, the ReCIM always keeps an abstract view. It will be detailed using next rules and patterns.

In the ReCIM graph model, there is only one type of node which has two attributes, an *id* and a *type* that can take as value "Source", "ConcreteDestination", "Monitor", "Analysis", "Plan" and "Execution". And all communications are considered as "relatedTo". *A* is "relatedTo" *B*, while *A* and *B* are two nodes means that there is a communication between *A* and *B*, which could be explained as data flows or invocations. The relation "relatedTo" keeps the abstract view of the CIM level. For example, in Figure 5, we have plugged the MAPE

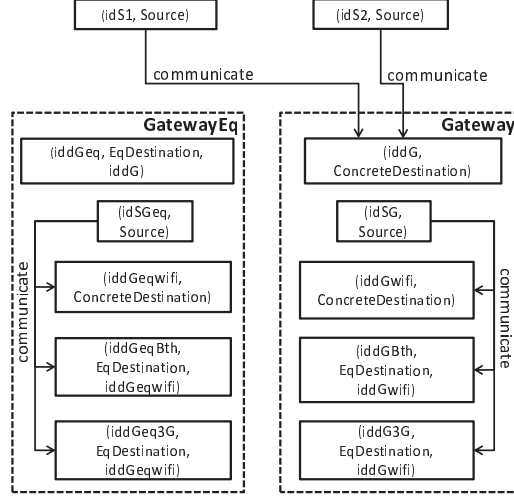


Figure 4. The InitCIM graph of RHCS

loop (idM1, idA1, IP1, idE1) between S1, S2 and the two gateways. Then, the link between the loop components is “relatedTo”. That means that the monitor node having the identifier idM1, can communicate only with the analysis and execution nodes having the identifiers idA1 and idE1. According to our case study (Figure 1) only (iddG, ConcreteDestination) is connected to (idS1, Source) and (idS2, Source) because initially only the “Gateway” is used in RHCS.

At this step, we note that the type “EqDestination” has been replaced by “ConcreteDestination” to be used by the (idE1, Execution) for the reconfiguration. This is due to a set of graph grammar rules which allowed us to consolidate all equivalent destinations and attached them to the “Execution” node to ensure later the dynamic reconfiguration.

4.3 From the CIM to PIM

The PIM includes more details than the CIM. Our starting point in this step is the ReCIM. In our approach, we distinguish two levels in the PIM. First, the RePIM-mape-step1 (Reconfiguration enabled PIM-mape-step1) and second the RePIM-mape-step2 (Reconfiguration enabled PIM-mape-step2). This step takes the ReCIM graph model as input and provides the RePIM-mape-step2 as a final model which describes the PIM level. This is performed through a set of refinements rules. The first refinement, which uses the graph grammar $GG_{ReCIM} \rightarrow RePIM\text{-mape-step1}$, details the Execution component (Figure 6) by dividing it into sub-components. In our reconfigurable architecture, the Execution component contains three components: the “VirtualDestination” which offers the same services as the “ConcreteDestination” with an empty body (a dummy component), the “DestinationManagement” which is responsible of performing the planned reconfiguration and the “DynamicConnector”, which is the main component in the Execution level. It binds “Source” requests to the “ConcreteDestination” according to the planned reconfiguration actions (substitution, load balancing...). This refinement leads to get the RePIM-mape-step1. The second refinement leads to obtain the RePIM-mape-step2, which details the Monitor component (Figure 6). However, this refinement takes input from non-functional requirements. In our reconfigurable architecture, there is three ways to detail the monitoring model:

- Source side: For each source, we associate a monitoring component “SSM”.
- Destination side: For all providers, we associate only one monitoring component “DSM”.
- Both Source and Destination sides: it represents a combination of the first and second cases.

In the RHCS, we propose to integrate autonomic architecture to manage the non-functional properties. Different combinations are possible: we propose to deal with the response time as a QoS parameter between the M2MServers

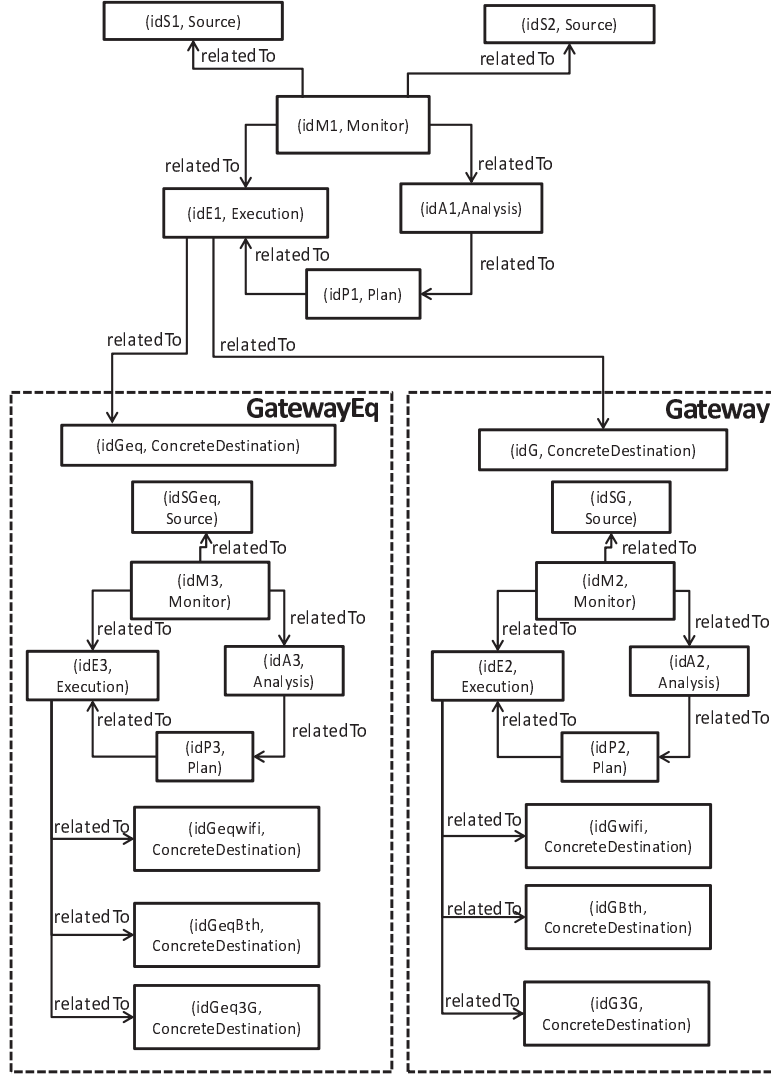


Figure 5. The ReCIM graph of RHCS

and the gateways, while the capacity, which is defined as the number of requests that could be treated without degradation during a period of time, as QoS parameter at the gateway level. So, the monitoring component used between the M2MServers and gateways must monitor both source and destination sides which explains its composition into sub-components “SSM”, for each source, and “DSM” for all “ConcreteDestination”. While, monitoring the overload of the software components requires only monitoring the destinations side in order to measure the capacity of each interface so that we integrate in the gateway only the “DSM” component. As result of the PIM level, we obtain the RePIM-mape-step2 of the RHCS as shown in Figure 6.

Compared to the ReCIM, the RePIM-mape-step2 is more detailed. First, it added other nodes already described and it distinguished between nodes interactions. The “relatedTo” edge is transformed into three types of edges as presented in Figure 6 :

- has-a: A has-a B , that means the node A is a composite of nodes and B represents its child.
- call: A call B , that means the node A calls or invokes the node B .
- monitors: A monitors B that means, the node A observes node B flows.

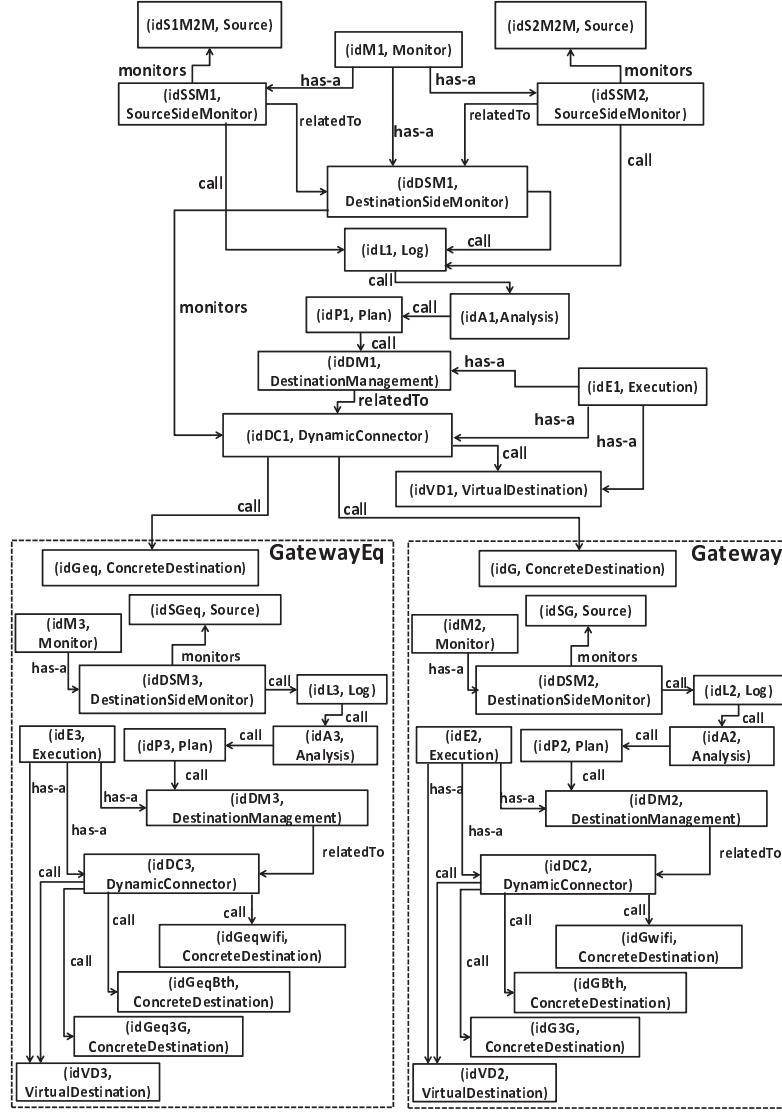


Figure 6. The RePIM graph of RHCS

In Figure 7, we show that we monitor both source and destination sides between servers and gateways. Therefore, the monitoring node (idM1,Monitor) has two “SSM”, (idSSM1,SourceSideMonitor) and (idSSM2,SourceSideMonitor), and a “DSM” (idDSM1,DestinationSideMonitor). In the gateways, we present that we monitor only provider side, in this case, the monitoring nodes (idM2,Monitor) and (idM3,Monitor) have -each one- only a “DSM” ((idDSM2,DestinationSideMonitor) and (idDSM3,DestinationSideMonitor)).

4.4 From the PIM to PSM

This phase is considered as the most important and complex part of our approach. The PIM, modeled as a graph model, is transformed to the PSM. We choose to benefit from the advantages of the SCA for modeling the PSM.

Using the SCA didn’t limit the developer to implement with one platform, since SCA exposes its functionality as services independently from the implementation. The mapping from the PIM to the PSM, which is made by applying a “Transformation Algorithm”, is partial. In this phase, a set of the component (such as “Log”,

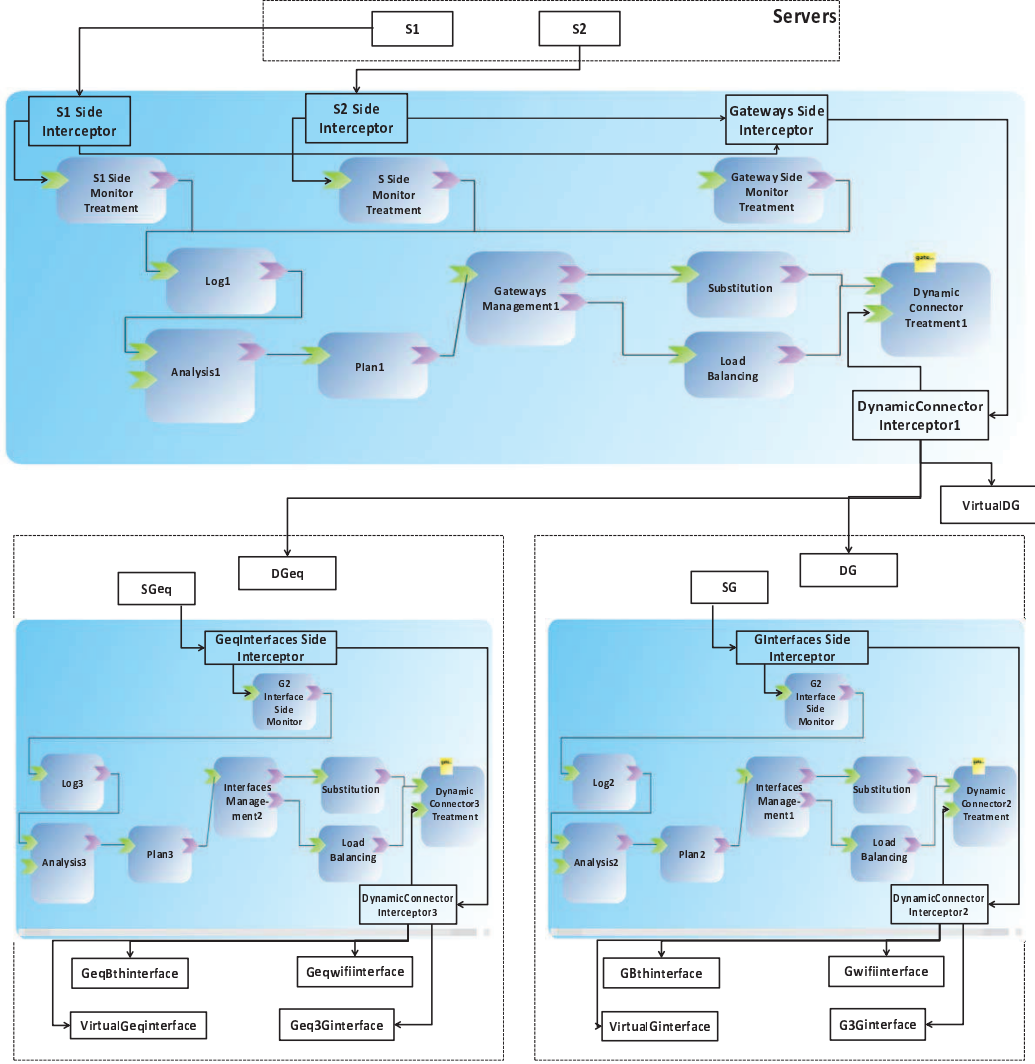


Figure 7. SCA architecture model

“Analysis” and “Plan”) are presented in SCA component while the “Monitoring” and the “Execution” are hybrids a combination of SCA and non-SCA components. For example, in the monitoring module, the “SSM” (respectively “DSM”) is composed of an SCA component implementing the monitoring treatment and a non SCA component “SourceSideInterceptor” (respectively the “DestinationSideInterceptor”) which intercepts the incoming inflow/outflow to the “Source” (respectively the “Destination”) and invokes the monitoring service to extract QoS parameter from the message. This composition is also applied in the “DynamicConnector” which intercepts messages destined to the “VirtualDestination”, which is modeled as a non-SCA component, using the non-SCA component namely “DynamicConnectorInterceptor” that redirects them to the “ConcreteDestination” through “DynamicConnectorTreatment” implemented in an SCA Component. The reconfiguration actions are modeled as SCA component which facilitates the plug-in of new actions. The technique of interception is fixed by the developer. The Figure 7 depicted the final model obtained by our approach in the case of the Remote Health Care System.

As shown in Figure 7, we maintain Substitution and LoadBalancing components between the servers and gateways and even under gateways in order to be used if requirements evolve over time. Different combinations of the reconfiguration actions may be presented to manage the QoS. For example according to the scenario 1 (cf Section

3), the SCA Substitution component is invoked as reconfiguration action between the servers and gateways. Thus, the degraded gateway will be replaced by the equivalent one that offers a better QoS by decreasing response time. In scenario2 (cf Section 3), for each gateway, the SCA LoadBalancing component is invoked in order to balance physicians/ nurses requests over different interfaces. As a consequence, the connected gateway to servers take into consideration the QoS by controlling the interfaces overload.

4.5 Summary

Our approach aims to obtain significant gains on software scheduling and cost, especially for complex systems. The only effort considered from the architect side was to generate the initial CIM, in otherwise, mapping the requirements from a set of specifications to a graph model. In this stage, our approach supports the transition from abstract level to detailed levels and transforms at the same time this application into an autonomic one while managing the non-functional requirements. Therefore, our approach hides application complexity and minimizes both design and time costs.

To illustrate the use of our model driven approach to design context-aware autonomic architectures, we provide the initial CIM of the Remote Health Care System as an input. After processing this input, we obtain a SCA architecture (Figure 7) supporting different technologies and ready to be deployed.

Acknowledgement

This research is supported by the ITEA2's A2NETS (Autonomic Services in M2M Networks) project².

5 Conclusion

In this paper, we proposed a model driven approach to automate the design of autonomic architectures following an MDA-based approach. We generate autonomic architectures from a high level functional requirements implemented by graph grammars rules. Our approach extends an initial architecture with MAPE components through transformation and refinement implemented using GMTE a Graph Matching and Transformation Engine. Then, we map the resulting architecture to different implementation technologies following the SCA standard such as EJB, JMS and SOAP. The Remote Health Care System represents our case study application. The generated architecture manages automatically the QoS through two reconfiguration actions : load balancing and substitution.

Our approach supports the synchronous communication based-applications. We are currently working on the refinement of our approach in order to manage the one way (asynchronous) communication . At the same time, we are implementing a GUI in order to make easy the capture of the high level function requirements.

References

- [1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. 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.
- [2] 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.
- [3] B. Becker and H. 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.
- [4] R. Ben-Halima, M. Jmaiel, and K. Drira. A QoS-oriented reconfigurable middleware for self-healing Web services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'08)*, Beijing, China, September 2008. IEEE Computer Society.
- [5] 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.

²<https://a2nets.erve.vtt.fi/>

- [6] E. Bruneton and M. Riveill. Experiments with javapod, a platform designed for the adaptation of non-functional properties. In *REFLECTION '01: Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, pages 52–72, London, UK, 2001. Springer-Verlag.
- [7] F. Chang and V. Karamcheti. Automatic configuration and run-time adaptation of distributed applications. In *HPDC*, pages 11–20, 2000.
- [8] 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.
- [9] W. Ellis, R. Hilliard, P. Poon, D. Rayford, T. Saunders, B. Sherlund, and R. Wade. Toward a recommended practice for architectural description. In *2nd IEEE International Conference on Engineering of Complex Computer Systems*, pages 21–25, Montreal, Canada, October 1996.
- [10] 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.
- [11] H. Fahmy and R. Holt. Using graph rewriting to specify software architectural transformations. In *15th IEEE international Conference on Automated Software Engineering, ISBN 0-7695-0710-7*, pages 187–196, Grenoble, France, September 2000.
- [12] A. Friday, N. Davies, G. Blair, and K. Cheverst. Developing adaptive applications: The most experience. *Integrated Computer-Aided Engineering*, 6(2):143–157, 2000.
- [13] D. Garlan and D. Perry. Introduction to the special issue on software architecture. *IEEE Transactions On Software Engineering*, 21(4):269–274, April 1995.
- [14] G. Gauvrit, E. Daubert, and F. Andr. Safdis: A framework to bring self-adaptability to service-based distributed applications. In *SEAA'10: Proceedings of the 2010 36th EUROMICRO Conference on, Software Engineering and Advanced Applications*, pages 211 – 218. IEEE Computer Society, 2010.
- [15] IBM. An architectural blueprint for autonomic computing. White paper, IBM Corporation, June 2006.
- [16] IEEE. Ieee std 1471-2000, ieee recommended practice for architectural description of software-intensive systems. In *IEEE*, pages i–23, 2000.
- [17] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, January 2003.
- [18] H. 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.
- [19] Y. Maurel, A. Diaconescu, and P. Lalanda. Ceylon: A service-oriented framework for building autonomic managers. In *Engineering of Autonomic and Autonomous Systems (EASe), 2010 Seventh IEEE International Conference and Workshops on*, pages 3 –11, march 2010.
- [20] 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.
- [21] 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.
- [22] M. Parashar and S. Hariri. Autonomic computing : An overview. pages 247–259, 2005.
- [23] 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.
- [24] 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.
- [25] J. Xiao and R. Boutaba. Qos-aware service composition and adaptation in autonomic communication. *IEEE Journal on Selected Areas in Communications*, 23(12):2344–2360, 2005.