Towards a Generic Reconfigurable Framework for Self-adaptation of Distributed component-based Application

Ouanes AISSAOUI[1], Fadila ATIL[1], Abdelkrim AMIRAT[2]

[1] Department of Computer Science, Badji Mokhtar University, Annaba, Algeria
aissaoui.ouenes@gmail.com, atil_fadila@yahoo.fr

[2] Department of Computer Science, Mohamed Cherif Messaadia University
Souk-Ahras, Algeria
abdelkrim.amirat@yahoo.com

Abstract. Software is moving towards evolutionary architectures that are able to easily accommodate changes and integrate new functionality. This is important in a wide range of applications, from plugin-based end user applications to critical applications with high availability requirements. This work presents a component based framework that allows introducing adaptability to the distributed component-based applications. The framework itself is reconfigurable and it based on the classical autonomic control loop Mape-k (Monitoring, Analysis, Planning, and Execution). The paper introduces a prototype framework implementation and its empirical evaluation that shows encouraging results.

***Keywords*:** Dynamic adaptation, Distributed systems, component-based application, Autonomic computing

1 Introduction

With the development of the networks and the increasingly significant delocalization of the enterprises, the implementation of distributed applications was essentially being a solution allowing the cooperation of the various constituent actors of the enterprise. For the purpose of reducing the construction costs of distributed applications increasingly sophisticated development tools were designed.

Nowadays, more and more of distributed applications run round the clock, seven days out of seven. The shutdown of this type of application to improve it, or quite simply to perform updating operations costs a significant sums of money. A solution to this problem is to provide mechanisms allowing the evolution or the modification of an application during its running without stopping it [1]. So, we speak about the dynamic reconfiguration of distributed applications which can be defined as the whole of the changes brought to a distributed application at runtime.

In the critical systems the adaptation must take place at runtime and the application should not be entirely stopped. Unfortunately, such adaptation is not trivial; there are

several conditions and constraints to be satisfied, and this leads to many problems to overcome.

The problems treated in this paper accost the domain of research around the dynamic adaptation of the computing systems and in particular, the distributed component-based systems. Generally, the existing approaches provide solutions for (1) reconfiguration in non-distributed systems [5] or (2) reconfiguration in distributed systems but not distributed reconfiguration [10] [11] which is composed of multiple distributed processes.

Our objective is to facilitate the addition of the dynamic adaptation capabilities to existing component-based applications by providing a solution of management of the distributed and coordinated dynamic adaptation. For that, we propose a component-based framework to add flexible monitoring and adaptation management concerns to a running component-based application. In the proposed framework, we separate the concerns involved in the classical autonomic control loop MAPE (Monitoring, Analysis, Planning, and Execution) [3] and implement those concerns as separate components. As we treat in our context the distributed applications, we integrated in our framework a mechanism to manage the distributed coordinated adaptations. These components are attached to each managed sub-system.

The remainder of the paper is organized as follows. Section 2 presents an overview of our solution for the distributed and dynamic reconfiguration. Section 3 details step by step the design of our framework following the autonomic computing MAPE-K phases. In Section 4, we give implementation details for a prototype of our framework. Finally, Section 5 concludes the paper.

2 Overview of our solution for the distributed and dynamic reconfiguration

We consider an application as a self-adaptable if it's composed of an adaptation system on the one hand and of a set of functional components on the other hand. The adaptation system is responsible for the management of the application context (collection of data, analyzes…) and of its adaptation, whereas the components represent logic trade of the application (functional code). Such separation is also suggested in many works such as [10], [13], [15]. In our context the adaptation system is represented by the framework which we present in this paper. Figure 1 shows an overview of our solution. For reasons of clearness, only two sites are represented.

As we treat the distributed applications, we find at each site a sub-system (a set of application components) plus one instance of our framework which manages the sub-system. The negotiation of adaptation strategy and the execution coordination of an adaptation operation are done via special components integrated in the framework. This organization makes the architecture of our solution decentralized what avoids the problems of the centralized approaches [14].

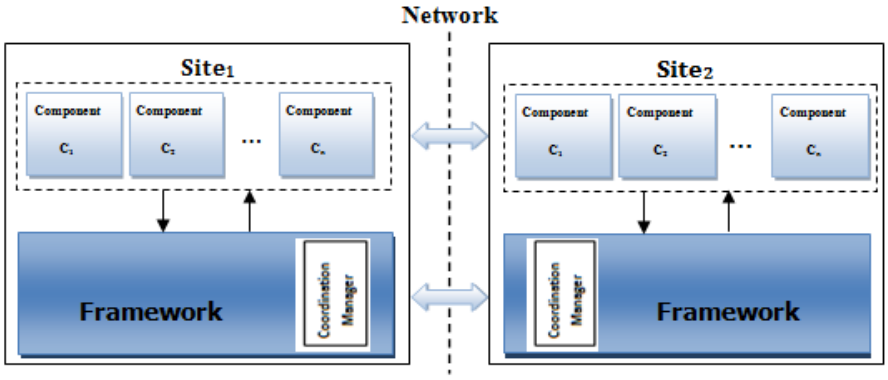

Fig. 1. Overview of our solution for the dynamic and distributed adaptation

3 Design of the proposed Framework

For the definition of our framework, we consider a set of constraints which are: (i) independence of the existing component models, (ii) flexibility and extensibility of the framework, (iii) adaptability of the framework, and (iv) taking into account of the distributed nature of the software to make it adaptable.

Our framework is based on the classical autonomic control loop Mape-k (Monitoring, Analysis, Planning, and Execution) [3]. This loop is used in many works treating the dynamic adaptation [8] [11], [12]. The difference between these current research activities is in the implementation way of the Mape-k loop.

So, in our framework we separate the concerns involved in a classical autonomic control loop and implement those concerns as separate components. The monitoring, analysis and adaptations are carried out by this control loop. We have merged the two phases of analysis and planning and we have integrated them in the same component. A significant part of the coordination, negotiation and the checking (checking of the application structure and the behavior of its components) were externalized of the control loop. The Figure 2 shows an overview of our framework.

The coordinator coordinates the execution of the adaptation operations; the negotiator negotiates an adaptation strategy with its similar at the other sites. The checking component carries out the checking of the application structure as well as the checking of the behavior of its components following the running of an adaptation operation. This checking operation is carried out on the architecture description of the application. For that, the component <<translator>> (presented hereafter) forwards the changes carried out on the application to its architecture description each time that an adaptation operation is done. This is for assuring a causal connection between the architecture description of the application and the system in running.

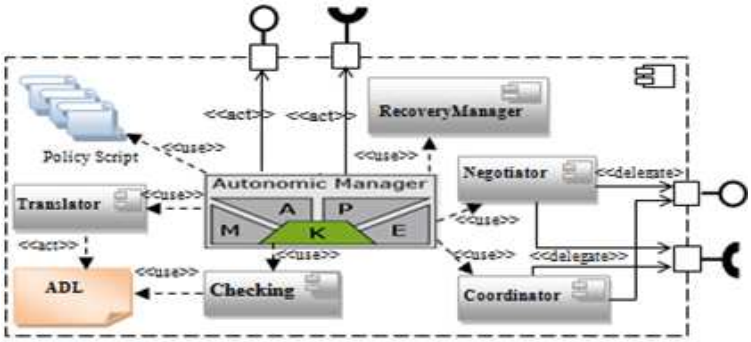

Fig. 2. Overview of the proposed framework

Knowledge Manager. It's a component used by the various components of the framework. It allows (1) to safeguard information on the knowledge base and (2) to provide information contained in the knowledge base to the other components according to their need.

Monitor. It's the first component in the chain which constitutes the control loop. It's composed of sub-component of the type <<CaptureContext>> and a set of sub-components of the type <<Sensor>>.The first supervises the application environment (e.g. memory, CPU, bandwidth…) whereas, the second supervises the various functional components of application (i.e., sub-system of the application managed by the framework) for that, it has a cardinality 1-*. These components (<<CaptureContext>> and <<Sensor>>) are responsible to (1) gather periodically the information of the controlled elements and (2) to pass them to the next object of the chain (the context manager).

ContextManager. It's the second object in the execution chain. It's a composite component in charge of the management of the running context. For that, it's composed of two components <<AcquisitionManager>> and <<Interpreter>> with a cardinality 1-1 for both. The first (1) gathers the information collected by the monitor and saves it in the knowledge base via the component <<KnowledgeManager>>, and (2) delegates the execution to the component <<Interpreter>>. This last, interprets data provided by the acquisition-manager. The received data are separately interpreted for each type of measurement in order to provide a significant contextual data. For example, a decreasing bandwidth event can be alone non representative. On the other hand, if it's repeated in time, it can indicate that the user moves away from an access point and thus being significant. So, the interpreter stores the values measured by event type. As the decision maker <<DecisionMaker>> is registered with events near the context manager, the detection of a suitable context change triggers the notification of the decision maker. In this case, the interpreter delegates the execution to the next component of the chain (DecisionMaker).

DecisionMaker. It's the third object in the execution chain of the control loop. It's responsible of making an adaptation decision and provides in exit the adaptation strat-

egy to be applied. For that, it subscribes with events near one or more context managers. The decision maker (1) starts the interpretation of the adaptation policy (script) which is of type ECA "Event, Condition, Action". It is possible that several rules (i.e. several adaptation operations) so, several strategies are applicable at the same time. In this case, the decision maker must order these strategies according to their priorities, then (2) it initiates a negotiation operation of this strategy via the component <<Negotiator>> according to the given sequence. This negotiation is necessary since we speak here about the distributed adaptations. At the end of the negotiation, the decision of the negotiator is the notification of the participants of the negotiation failure or of its success with the strategy selected. Thereafter, the <<DecisionMaker>> (3) delegates the execution to the next object (Executor) in the chain.

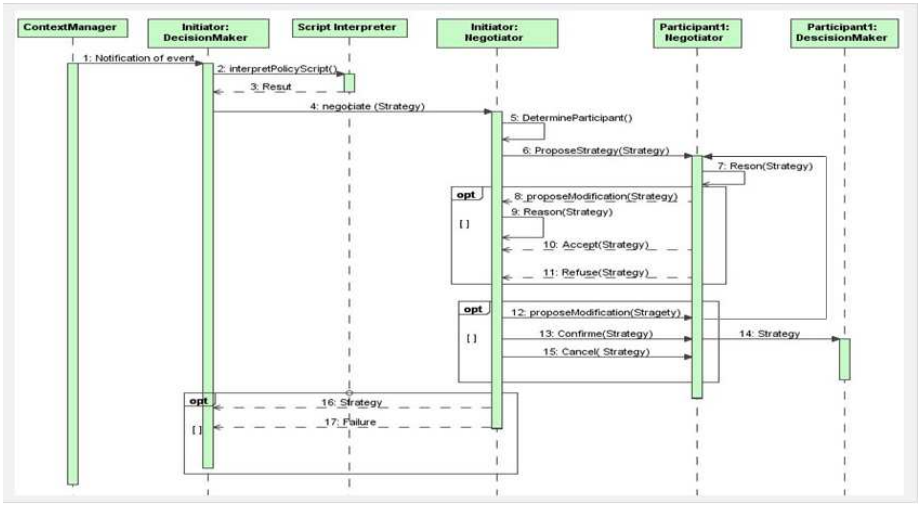

Fig. 3. Sequence diagram of negotiation between two adaptation managers

The diagram showed in figure 3 describes the sequence of messages for the negotiation of a strategy between an initiator and participants. For reasons of clearness, only one participant is represented.

The initiating decision maker chooses an adaptation strategy. Then, it asks its negotiator to negotiate the strategy which it chose. This negotiator proposes simultaneously to each participant the strategy that the decision maker chose. The negotiator of each participant receives the strategy and interprets its policy to reason on its applicability. It can then accepts, refuse or propose a modification of the strategy. Then, it answers the initiating negotiator. When this last receives all the answers, it thinks on the acceptances and/or the applicability of the modifications asked. When all the participants accept the strategy, the negotiation succeeds. Otherwise, it detects and solves the conflicts and it can then in its turn propose a modification of the strategy. The negotiation process is stopped if one negotiator refuses a strategy or if a stop condition is checked. This condition is in connection to the authorized maximum time of negotiation or with the maximum number of negotiations cycles. If the negotiation

succeeds, the initiating negotiator returns to the initiating decision maker the strategy resulting from the negotiation and sends to the negotiator of each participant the final strategy. At the reception of this strategy, the negotiator of the participant asks to this last (3) to adopt the strategy resulting from the negotiation and delegates the execution to the next object in the control loop <<Executor>>.

Executor. We adopted the transaction-based system technique [16] to make our adaptation operations transactional i.e. having the properties ACID (Atomicity, Consistency, Isolation, Durability) of transactions. So, we consider an adaptation operation as a set of primitive operations of adaptation.

The purpose of this decomposition is to facilitate the detection of errors during the running of these operations and much more their recovery what allows to preserve the consistency of the application to be adapted. An adaptation operation is validated (commit) only if all its primitive operations are carried out without faults. If an error is detected before finishing the execution of the adaptation operation, the effect of all primitive operations is cancelled for preserving the application consistency. Figure 4 shows our abandon model of an adaptation operation.

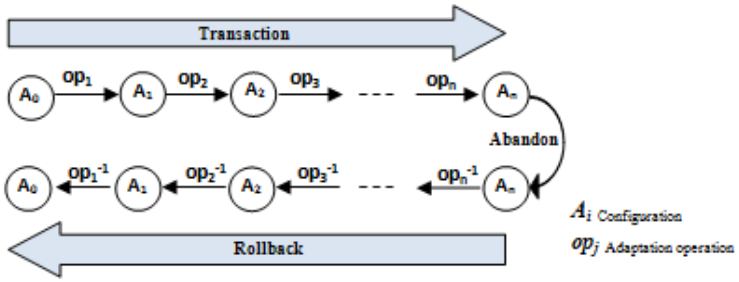

Fig. 4. Abandon model of an adaptation operation

According to our model the effect of the reconfiguration operation is cancelled by the running of the reverse action of each primitive operation done. We define the concept of opposite operation which is used to undo the effects of a reconfiguration operation and which is useful to ensure its atomicity. All the reconfiguration operations are not necessarily invertible. The operations of modification of the component properties are generally their own reverse. In addition, the opposite (or reciprocal) operation of a primitive operation is not necessarily a primitive operation but can be a composite operation.

Given a configuration *A*, by application of the composite operation $op^{-1} \circ op$ on *A*, we obtain normally $op^{-1} \circ op\ (A) = A$ according to the following diagram where $\overset{op}{\Rightarrow}$ indicates the reconfiguration by the operation *op*:

$$A \overset{op}{\Rightarrow} \acute{A} \overset{op^{-1}}{\Longrightarrow} A$$

For example, the reverse of the operation *removeComp* which allows removing a component is the operation *addComp* which allows adding a component.

In certain component model, certain operations are not invertible, like in the Fractal model where the operation *new* for which the opposite operation would correspond to an operation of destruction of component does not exist. For the particular case of the non reversible operations, the cancellation of reconfiguration requires a specific treatment in the event of abandonment of the transaction. Compensation operations can then be associated to these operations; moreover no guarantee on atomicity can then be given because the state of the system resulting from the abandonment cannot be completely identical to the state before the running of the reconfiguration.

Algorithm 1

```
Begin
For all op_j ∈ startegy do
        RunOp(op_j) ;
        TranslateChanges(); // translate changes to the architectural description
        if not IsConsistentApplication () then
                SendMessageToCoordinator (''Adaptation failure'');
                For all executed primitive operation op_j do   RecoveryManager.undo(op_j) ;
                end_for
        end_if
        else
            SendMessageToCoordinator (''ApplyNextOperation'');
            response←coordinator.decisionCoordinator() ;
            if response != '' ApplyNextAction '' then
                SendMessageToCoordinator(''Adaptation failure'') ;
                For all executed primitive operation op_j do RecoveryManager.undo(op_j);
                end for
              BREAK; // to exit the more external loop (for)
            end_if
         end_else
end_for
if all operations in strategy are executed // if the adaptation is succeeds
          LogExecutedOps();
end_if
End.
```

The <<Executor>> is the component responsible for the execution of the adaptation strategy suggested by the component <<DecisionMaker>> and of its control. For that, it (1) triggers the execution of each reconfiguration action in the strategy according to the order of their appearances.

We consider the adaptation of distributed application as a global adaptation process composed of distributed local adaptation processes. For that, a coordination component of the execution of an adaptation is necessary.

Following the running of each primitive adaptation operation, the <<Executor>> (2) calls the translation function of the component <<Translator>> for transferring the changes performed in the application in running to its architectural representation. After, it (3) carries out the checking of the consistency of the application structure and the checking of the validity of the behavior of its components via the component <<Checking>>. If a constraint is violated, the <<Executor>> asks to the recovery component <<RecoveryManager>> to carry out the rollback for preserving the con-

sistency of the application. In this case there, the component <<RecoveryManager>> undoes the effect of all the primitive adaptation operations which are already executed through the execution of their reverse operations as explained in the previous section. Moreover, this initiating executor notifies the coordinator of the failure of execution of the primitive adaptation operation in question. This last, deals with the notification of the other participants of this failure (the participants are the coordinators of the adaptation execution which are deployed at the other sites) so that they can undo the effect of the primitive operations already carried out at their level in order to preserve the global consistency of the application.

In the opposite case, i.e. if the <<Checking>> does not detect any error following the running of a primitive operation of adaptation, the <<Executor>> sends a message *"ApplyNextAction"* to the coordinator. This last awaits the reception of all the participants' messages. If one of them replies negatively (i.e. adaptation failure), the coordinator announces the failure of the execution of the adaptation operation. Otherwise, it indicates to the participants to carry out the next primitive adaptation operation and the process is still repeated. After the running and the validation of all the primitive operations of all adaptation operations in the strategy, the <<Executor>> (4) logs these executed operations in the journal of the application for a future use. The end of the execution of this operation determines the end of the control loop cycle. The running of the <<Executor>> is summarized by algorithm 1.

4 Implementation and validation

In this section, we give details and technical choices made to implement an instance of our framework. We present also the result of the evaluation of this framework.

4.1 Background

For the implementation of the elements of our framework which we have presented in the section 3, we have used the component model ScriptCOM [9] which is an adaptable model extension of the model COM (Component Object Model) [2]. It's a component model which we have proposed in an earlier work. We have used this model because it allows the development of a scripting component as it's based on the use of the scripting languages. These languages allow the incremental programming, i.e. the possibility of running and developing simultaneously the scripts which represents in this context the components implementation. This adaptation is possible via a set of three controllers which are: the Interface controller, script controller and property controller. Moreover, this model benefited from contributions and advantages of the COM model since it's an extension of the latter. We have chosen this component model in order to make our framework itself adaptable.

4.2 Framework implementation

The framework is implemented via the component model ScriptCOM as a set of non functional components that can be added, removed or modified at runtime. We have designed a set of predefined components that implement each one of the elements

which we have described in Section 3. This is just one of possible implementations and particularly, this has been designed to provide self-adaptable capabilities to the framework.

4.3 Validation plan

In order to validate our proposal, we have used the industrial model EJB [4] for the development of an application example which is an http server. We have chosen this model to prove that our framework is generic because it's implemented via the component model ScriptCOM and the adapted application is developed via another component model (EJB). We have chosen this application (HTTP Server) since it's used in the evaluation of many works [5-7]. Therefore, it represents a reference for us. This application of the type server does not interact directly with a user. However, the need for performances implies that it must be able to adapt to the characteristics of its host and the type of load which it undergoes. In this example, the significant context for the adaptation will be thus that of the material and software resources rather than the characteristics of the end-user. In order to improve the performances, we have integrated a mechanism to put in cache the content of files which it reads.

The objective of the validation in this paper is to test the adaptation mechanism influence on the application response time and the adaptation time. We have obtained encouraging results, where the influence on the response time is stable and that overhead time is about 15%. The adaptation time average is approximately 2 seconds. Of course, this figure is large compared to the response time of one request which is approximately 30ms. Notice, that this test is done via machines equipped with Intel(R) Core(TM) 2 Duo CPU T5670 @ 1.80GHz 1.79GHz and 1 GB of RAM.

5 Conclusion

We have presented a generic reconfigurable component-based framework for supporting the dynamic adaptation of distributed component-based applications. Our framework is based on the classical autonomic control loop Mape-k (Monitoring, Analysis, Planning, and Execution). It implements each phase of the autonomic control loop as a separate component, and allows multiple implementations on each phase, giving enough runtime flexibility to support evolving non functional requirements on the application. To the difference of the others frameworks, our framework is conceived to support the distributed adaptations. Moreover, it's independent of the component models and designed to minimize the cost and the time of the addition of capacities of self-adaptation to a large variety of system. A prototype of this framework has been implemented using an adaptable component model *ScriptCOM*. Moreover, an empirical evaluation of this prototype is done and it shows encouraging results.

Our future work focuses on improving the response time by the improvement of the negotiation and coordination algorithms.

References

1. Taylor, R.N., Medvidovic, N., et al. Software Architecture: Foundations, Theory, and Practice. 736 pgs., John Wiley & Sons (2008)
2. Microsoft Corp., Component Object Model, `http://www.microsoft.com/COM`
3. IBM. An architectural blueprint for autonomic computing. Autonomic computing white-paper, 4th edition (2006)
4. Matena, V., Hapner, M.: Enterprise Java Beans Specification v1.1 - Final Release. Sun Microsystems, Mai (1999)
5. David, P.C.: Développement de composants Fractal adaptatifs: Un langage dédié à l'aspect d'adaptation. PhD thesis, université de Nantes, France (2005)
6. Léger, M. : Fiabilité des Reconfigurations Dynamiques dans les Architectures à Composant. PhD thesis, Ecole Nationale Supérieure des Mines de Paris (2009)
7. Dormoy, J., Kouchnarenko, O., Lanoix, A.: Using Temporal Logic for Dynamic Reconfigurations of Components. In: FACS, 7th Int. Ws. on Formal Aspects of Component Software, Portugal (2010)
8. Ruz, C., Baude, F., Sauvan, B.: Flexible adaptation loop for component-based soa applications. In: ICAS 2011, the Seventh International Conference on Autonomic and Autonomous Systems, pp. 29–36. , May (2011)
9. Aissaoui, O., Atil, F.: ScriptCOM an Extension of COM for the Dynamic Adaptation. In: Proc. of 2nd IEEE International Conference on Information Technology and e-Services, pp. 646-651, Tunisia (2012)
10. Garlan, D. Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow : Architecture-based self-adaptation with reusable infrastructure. In : IEEE Computer, 37(10) :46-54, (2004)
11. Maurel, Y., Diaconescu, A., Lalanda, P.: Ceylon: A service-oriented framework for building autonomic managers. In: Engineering of Autonomic and Autonomous Systems (EASe), Seventh IEEE International Conference and Workshops, pp. 3 –11, (2010)
12. Gauvrit, G., Daubert, E., Andr, F.: Safdis: A framework to bring self-adaptability to service-based distributed applications. In: SEAA'10, Proceedings of the 36th EUROMICRO Conference on, Software Engineering and Advanced Applications. IEEE Computer Society, pp. 211–218, (2010)
13. Baresi, L., Guinea, S. : A3: Self-Adaptation Capabilities through Groups and Coordination. In : ISEC '11, Kerala, India, (2011)
14. Tan, C., Mills, K.: Performance characterization of decentralized algorithms for replica selection in distributed object systems. In: WOSP, pages 257-262. ACM, (2005)
15. Zouari, M., Segarra, M.T., André, F.: A Framework for Distributed Management of Dynamic Self-adaptation in Heterogeneous Environments. In: IEEE International Conference on Computer and Information Technology: 265-272, (2010)
16. Gray, J., Reuter, A. : Transaction Processing : Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, (1992)