Reflective Middleware for Automatic Management of service-oriented applications using the theory of Signatures of Failure

Juan Vizcarrondo
Centro Nacional de Desarrollo e Investigación en Tecnologías Libres (CENDITEL)
Mérida, Venezuela
jvizcarrondo@cenditel.gob.ve

José Aguilar
Cemisid; La hechicera, Núcleo Pedro Rincon Gutierrez
Universidad de los Andes
Mérida, Venezuela
aguilar@ula.ve

Ernesto Exposito
CNRS ; LAAS
Université de Toulouse; UPS, INSA, INP, ISAE; LAAS; F-31077
Toulouse, France
ernesto.exposito@laas.fr

Abstract—The composition of web services enables the cooperation going through organizational boundaries, allowing new and complex scenarios for collaboration. Middleware, and particularly reflective middleware, have been used as a powerful tool to cope with inherent heterogeneous nature of distributed systems and to give them greater adaptability capacities to face their environment. In this paper we propose a distributed reflective middleware for service oriented applications aimed at proposing solutions to cope with fault tolerance problems in the context of services composition and choreography based on the Autonomic Computing Architecture.

Keywords: Distributed reflective middleware, Web service composition, Web service fault tolerance, Autonomic Computing Architecture.

1. INTRODUCTION

The development of SOA applications (Service Oriented Architecture) is a software development model in which an application is broken down into small units, logical or functional, called services. SOA allows the deployment of distributed applications very flexible, with loose coupling between software components, which operate in heterogeneous distributed environments. An example of software components to be integrated are the web services. In general, web services are computational entities which are autonomous and platform-independent, that can be composed with others in order to offer composite services.

The Services are inherently dynamic and cannot be assumed to be always stable [4], because during the natural evolution of a service (changes to their interfaces, its internal calculation, misbehavior during its operation, among others) can alter the resulting service. Thus, in the case of service composition the failure of a single service leads to error propagation in the others services involved, and therefore the failure of the system. Such failures often cannot be detected and corrected locally (single service), so it is necessary to develop architectures to enable diagnosis and correction of faults, both at individual (service) and global (composition levels).

Previous works have addressed the problem of propagation of faults in the composition of services, implemented semi-centralized architectures composed of local diagnosticians distributed within each service composition, which are coordinated by a central diagnoser get an overview of the problem and thus implement a repair strategy is finally implemented this diagnostic center. In this paper we propose an architecture for the diagnosis of fully distributed service compositions, which is not coordinated by any central body, in which the diagnosis of faults is performed through the interaction of the diagnosticians present in each service composition and repair strategies are developed through consensus of each repairer distributed equally in the composition. Additionally, our architecture is based on Autonomic Computing to facilitate construction, understanding and dissemination to a robust architecture with Provides methods, algorithms, and tools for self-healing systems.

2. RELATED WORKS

Web services are prone to failure, which can be classified at the level of the service itself and/or the sequence of calls in a composition of these. Thus, in [5] proposes a taxonomy for the analysis of possible failures and perceived effects at both local (service) and composition levels, representing an excellent starting point for this work. In addition, a first attempt is made to correlate the failures and possible mechanisms that have been implemented to solve them.

On the other hand, several architectures have been proposed for fault management and recovery in the web service composition. In [6] defines a reflective middleware called SOAR for fault management in service composition, which is conceived as a global structure (centralized) to monitor and adapt the complete system. The middleware is composed of two levels: The first (base level) is responsible for describing the basic characteristics of an SOA system, and the second level (meta level) is responsible for monitoring and adapting the SOA system. The reflection of the middleware is performed by making use of dynamic binding of web services composition, when connecting or disconnecting the services forming part of the joint task.

A second proposal is a centralized architecture for web services reparation [7], called self-healing architecture. It carries out the SOA system reparation using quality measures from Web Services. The architecture consists of three modules: the Monitoring and Measurement module (it is responsible for observing and keeping records of QoS parameters that are relevant), the diagnosis and decision strategies engine (it detects degradation of the system and identifies reparation plans), and finally, the Reconfiguration module (it implements the reparation plan). Also in [14] proposed a centralized architecture like average based QoS monitoring

In [8] proposes a decentralized architecture composed by 2 levels. The first level uses a local diagnoser for each service that is part of the composition, which communicates with a global diagnoser (central) for the diagnosis of the entire composition. The global diagnoser is responsible for the coordination of the local diagnosers making use of the exchange of messages to find the service and the activity responsible for the failure; and implements mechanisms for the composition recovery. Each local diagnoser instances chronicles which describe failure patterns defined previously (off-line), which is propagated to the global diagnoser. It makes a calculation about the sequence of events in the services to find the occurrence of an error according to the chronicles instanced by the local diagnosers. Furthermore, in [15] proposed a structure composed of local diagnosticians are coordinated by a global diagnoser implements work repación

3. Middleware Architecture Proposal

Reflection is the ability of a program to monitor and change its own behavior, as well as aspects of its implementation (syntax, semantic, etc.), allowing the ability to be sensitive to its environment. In this way, we can define programs with a dynamic behavior, with an adaptive architecture; that is, we can design programs that are able to dynamically change or evolve.

An interesting concept to introduce is the meta-programming, the ability of a program to read, transform and write other programs. Thus, the reflection can be seen as a meta-programming which is not performed by an external program but the program itself. In general, reflective computing has two processes [13]:

- **Introspection:** The ability of a component to observe and reason about its own execution state.
- **Intersection:** The ability of a component to modify its own execution state, or alter its own interpretation or meaning.

A reflective system is composed of 2 levels: base level that represents the operation of the system and the meta level that performs the reflection on the system by constructing a representation of the base level (introspection) and modifying the base-level entities (intersection) to modify the system behavior.

In this proposal, the reflective middleware will be fully distributed through all services of the system, in order to have a closer view of the occurrence of events that happen in the application. Our middleware is divided in the classical two levels: the base and the Meta level (see Figure 2), which are described below:

Base Level: A service composition can be seen as a set of calculations and iterations of the services that compose an SOA application and the set of rules and definitions that govern those iterations (SOA System). The base level of the middleware needs to know: the iterations that occur in the choreography and the definitions and rules that govern these iterations. Additionally, to achieve an adequate level of introspection the base level must observe both the SOA system and the SOA application. To achieve this, it uses the next elements: WSDL, UDDI, OWL-S and SCA). In addition, it uses FraSCAti platform for the intersection process of the service choreography.

Meta Level: This is the part of the middleware that provides the capacity for reflection. The base-level introspection is done by analyzing the message exchange between the services that are part of the composition and the components of the SOA system. The meta level is instantiated for each service of the choreography. The meta level is decomposed into 4 components:

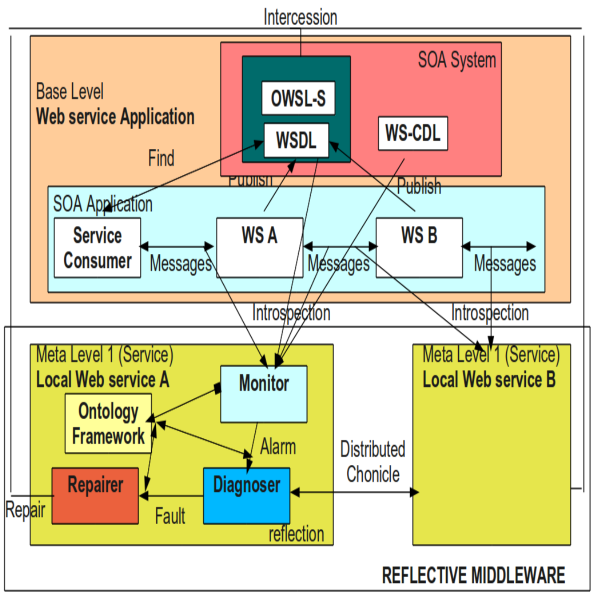

Fig. 1 Our Reflective middleware architecture.

- **Monitor (components Monitoring):** it inspects the communication services, and therefore has the ability to issue alerts to the diagnoser to begin the analysis of possible failures. Monitoring is conducted on the QoS parameters of services in the composition response time, throughput, availability and consistency of data exchanged.
- **Diagnoser (components behavior Analysis):** Performs system diagnostic. Its action is invoked by the Monitor module (an alarm) or another diagnoser module distributed in other service, to achieve the identification of a distributed chronicle of choreography (fault pattern).
- **Repairer (Reparation Plan and Execution):** it has mechanisms for the resolution of faults present in the composition of services.
- **Ontological Framework:** It manages all the knowledge used by the middleware to perform its tasks. Its structure is composed by: Distributed Chronicles of choreography, QoS and service fault recoveries taxonomies, rules, etc, All these functions are achieved using a common knowledge base.

In our architecture, both the Monitor and Diagnoser are responsible for performing introspection and repairer performs the intersection of the service composition.

4. Reflective Middleware Architecture Based on Autonomic Computing

Autonomic Computing [9] is a self-managing computing model inspired by the autonomic nervous system of the human body. It creates systems that are able to be self-managed and are able of high-level functioning while keeping the system's complexity invisible to the user. It incorporates sensors and effectors to the systems to allow collecting details about the system behavior and to act accordingly. Autonomic Computing defines an architecture composed of 6 levels (see Figure 2)[9]:

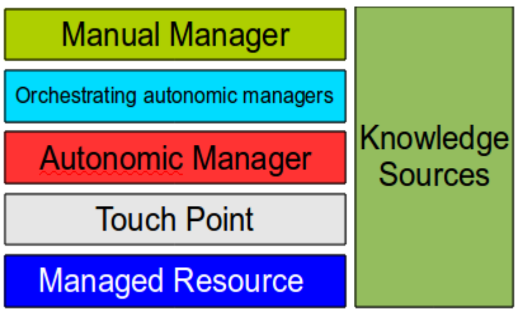

Fig. 2 Autonomic Computing Architecture.

- **Managed Resources:** can be any type of resource (hardware or software) and may have embedded self-managing attributes.
- **Touchpoints:** implements the sensor and/or effector mechanisms for the managed resources.
- **Autonomic Manager:** implements the intelligent control loops that automate the tasks of auto-regulation of the applications. Autonomic manages is composed by four parts called MAPE: Monitoring, Analysis (diagnoser), Plan (to decide how to repair) and Execution (to send the orders to the components).
- **Orchestrating autonomic managers:** Provide coordination between Autonomic Managers.
- **Manual Manager:** creates a consistent human-computer interface for the autonomic managers.
- **Knowledge Sources:** Provides access to the knowledge according to the interfaces prescribed by the architecture.

Additionally, FraSCAti[1] [10] is a platform for the implementation of SCA[2] and fractal components [11], flexible and extensible, that allows [12]: run-time adaptation, property management and reflective capabilities. With Frascati we can implement a component as a component fractal with a set of controllers called membrane (these controllers allow introspection, configuration and reconfiguration).

Our middleware is composed of a set of distributed resources that work together to make a global goal, which can be seen as a Autonomic computing system. For this, we extrapolate the two levels of our middleware (meta and base) into the 5 levels of an Autonomic Computing Architecture (see figure 3).

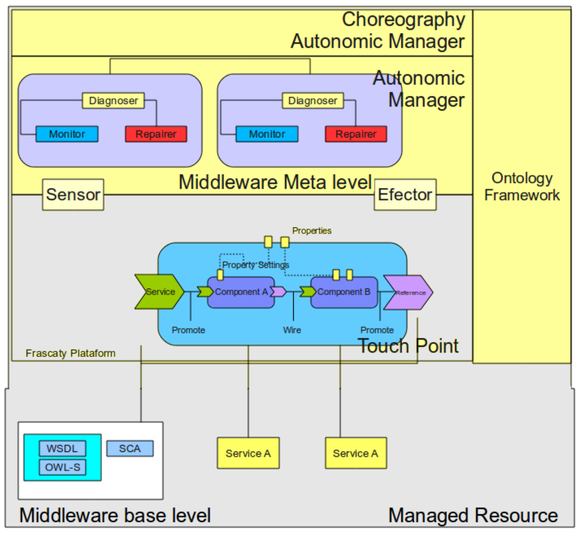

Fig. 3 Middleware Architecture based on Autonomic Computing.

1 FraSCAti project, http://frascati.ow2.org/
2 SCA provides a model for composing applications that follow Service-Oriented Architecture principles

In our case, the Managed resources and the Touch Point correspond to the base level; and the autonomic Manager, the Ontology Framework and the Choreography Autonomic Manager define the meta level. Additionally, the Autonomic Manager is composed by the three modules of our meta level: Monitor, Diagnoser and Repairer, which correspond to the MAPE structure of the Autonomic Computing Architecture. Each autonomic manager can work locally (level of each service - Internal failures of the service) and/or globally (with the interaction between the Autonomic managers is possible to diagnose failures in the services composition), so this component must communicate with the rest of the Repairers of the other services of the composition. Finally, the ontology Framework (it manages all the knowledge used by the middleware to perform its tasks), in our middleware is composed of chronicles and specific web services ontologies.

The touchpoint interface is implemented/required by the services in order to retrieve monitoring data via the sensors interface and to enforce the decisions for repairing via the effectors interface.

5. CASE STUDY

In order to test our proposal, we are going to use a very common example of e-commerce, where there are three business processes involved in the choreography (see figure 4):

- **Shop:** is the store where users buy products.
- **Supplier:** Provides products to the shop; needs to check the warehouse before issuing a response to the store.
- **Warehouse:** is where products are stored. This process has a service agreement (SLA) which consists in that at least one product from the list must be returned. It be able to carry out searches on external sites to buy products.

Now, we describe a classical behavior of this application:

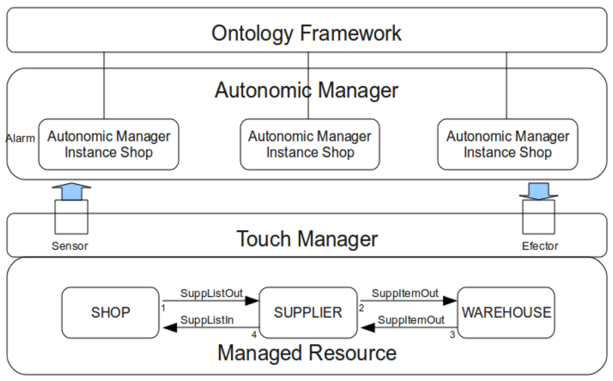

Fig. 6 Choreography example.

(1) SuppListOut: Shop provides the list of products required to the supplier.

(2) SuppItemOut: Supplier checks its deposit invoking the Warehouse process.

(3) SuppItemOut: Warehouse provides the answer about the list of products in the deposit to the Supplier, which must contain at least one product.

(4) SuppListIn: The Supplier notifies the Shop which products can provide.

Remember that the middleware contains many instances of the autonomic manager as services are in the composition, the Ontology Framework is consulted by the Autonomic Manager to perform its functions, and the touch points are sensors and efectors provided service-composition interface. To show the operation of our autonomic manager we are going to consider the following situations:

Warehouse SLA violation (Web service fault): The iterations (1) and (2) are normally given, but (3) provides an empty list of products to the Supplier. This is a violation of the SLA for the Warehouse service. The solution is to adjust the Warehouse service configuration to perform a external search of products in order to provide at least one product. The middleware must perform the reparation (warehouse settings) to ensure the proper functioning of the choreography (see figure 5):

i. Supplier's monitor emits an alarm to the Supplier's diagnoser to begin the process of diagnosis.

ii. Supplier's diagnoser makes an inference in the knowledge base and find that the problem is an external error due to an incorrect result. It propagates the diagnostic to the Warehouse's diagnose.

iii. Warehouse diagnoser finds that the error is internal (Parameter incompatibility), therefore, the diagnoser calls its repairer (Warehouse's Repairer)

iv. Warehouse's Repairer performs the reparation using the knowledge base. It determines that the solution is to change the properties of the service so that it can search products in external storages.

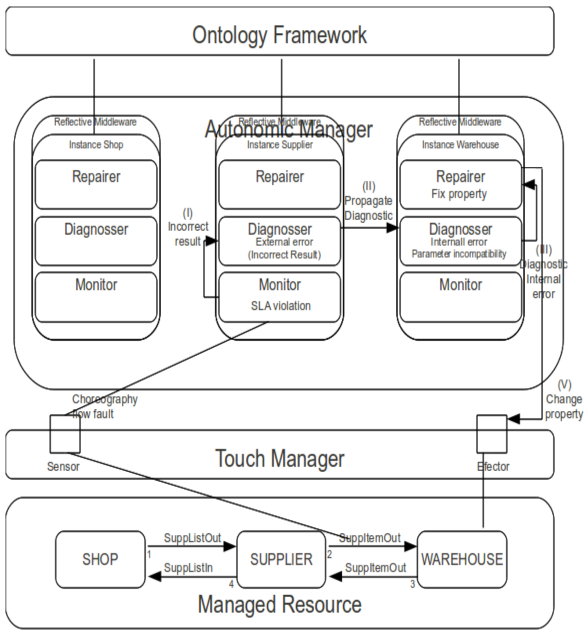

Fig. 7. Middleware self-healing due to faults in SLA violation on the service

Warehouse Service Delay (Choreography flow fault): Again, the iterations (1) and (2) are normally given, but (3) has a delay which generates an error in the composition because the supplier cannot produce results for the shops (Supplier is unable to give its response in time). A possible solution would be warehouse service reallocation (see figure 6):

i. Shop's monitor sees a violation in the QoS turning on an alarm to the Shop's diagnoser.

ii. Shop's diagnoser finds an external error (Time Out) and propagates the diagnostic to the Supplier's diagnoser.

iii. Supplier's diagnoser infers an external error (Service Delay), and propagates the diagnostic to the Warehouse's diagnoser.

iv. Warehouse's diagnoser finds an internal error (Service Delay) and calls its repairer.

v. Warehouse's repairer executes a self-reparation (reallocation of the warehouse service).

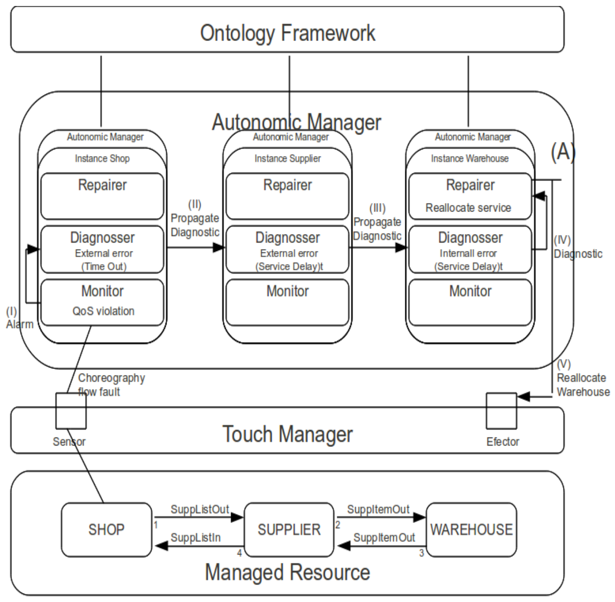

Fig. 8. Middleware self-healing due to faults in service Delay.

6. CONCLUSIONS

In this paper we propose a reflective middleware Architecture for management of service-oriented applications. Our middleware is designed to be fully distributed through all services of the SOA Application, counting for this with the base level that contains both the SOA system and the SOA Application, and the meta level with components to execute the reflection. The architecture uses the model of Autonomic computing which will allow an easy adaptation of our self-healing system. Additionally, our base level will use Frascati to facilitate the implementation of Introspection and Intersection in our middleware.

We have tested the design of our middleware, shown the distributed adaptive capabilities of our architecture for failures covering both services and the composition of services, thus our design is able to fix these failures, which contrasts with other approaches that do not show a fully distributed (See TABLE I)

Finally, the ontology framework represents the knowledge needed to perform the operations of the middleware, which being distributed completely it also requires that this be distributed. So, in future works we need to develop structures to represent this distributed knowledge (chronicles, etc.) and mechanisms to use it. This work is still in progress, but the initial results are interesting and promising.In this paper we propose a reflective middleware Architecture for management of service-oriented applications. Our middleware is designed to be fully distributed through all services of the SOA Application, counting for this with the base level that contains both the SOA system and the SOA Application, and the meta level with components to execute the reflection. The architecture uses the model of Autonomic computing which will allow an easy adaptation of our self-healing system. Additionally, our base level will use Frascati to facilitate the implementation of Introspection and Intersection in our middleware.

TABLE I. COMPARING OUR ARCHITECTURE

Architecture	**Monitoring phase**	**Diagnosis phase**	**Recovery phase**
SOAR [6]	Centralized	Centralized	Centralized
self-healing Architecture [7]	Centralized	Centralized	Centralized
self-healing in Dynamic Web Service Composition [14]	Centralized	Centralized	Centralized
Chronicle Architecture [8]	Distributed	Semi-centralized	Centralized
Web Services with Diagnostic Capabilities [15]	Distributed	Semi-centralized	Centralized
Our Reflective Middleware	Distributed	Distributed	Distributed

REFERENCES

[1] J. Camara , C. Canal and J. Cubo, "Issues in the formalization of Web Service Orchestrations". Second International Workshop on Coordination and Adaptation Techniques Entities (WCAT05).

[2] O. Kopp, and F. Leymann, "Choreography Design Using WS-BPEL" IEEE Data Eng. Bull., Vol. 31, Nr. 3 (2008) , p. 31-34.

[3] A, Barros, M. Dumas and P. Oaks, "A Critical Overview of the Web Services Choreography Description Language (WS-CDL)". BPTrends Newsletter.

[4] K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman, "Grid Information Services for Distributed Resource Sharing". In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181--184. IEEE Press, New York (2001)

[5] K. S. Chan, J. Bishop, J. Steyny, L. Baresi and S. Guinea, "A Fault Taxonomy for Web Service Composition", Service-Oriented Computing - ICSOC 2007 Workshops: ICSOC 2007, pp. 363-375, 2007.

[6] Gang Huang, Xuanzhe Liu and Hong Mei, "SOAR: Towards Dependable Service-Oriented Architecture via Reflective Middleware". International Journal of Simulation and Process Modelling (IJSPM), Volume 3, Issue 1/2, pp. 55-65, 2007.

[7] R. B. Halima, E. Fki, K. Drira and M. Jmaiel, "Experiments results and large scale measurement data for web services performance assessment". Computers and Communications, 2009. ISCC 2009. IEEE Symposium, pp. 83-88, 2009..

[8] WS-Diamond, "WS-Diamond, IST-516933, Deliverable D4.3, Specification of diagnosis algorithms for Web Services – phase 2", Version 0.5, 2008.

[9] IBM, "An architectural blueprint for autonomic computing. IBM Autonomic Computing White Paper, 2005..

[10] OWS2 Consortium, "FraSCAti project, http://frascati.ow2.org/, 2011.

[11] J. Adamek, T. Bures, P. Jezek, J. Kofron, V. Mencl, P. Parizek, P and F. Plasil, "Component Reliability Extensions for Fractal Component Model", http://kraken.cs.cas.cz/ft/public/public_index.phtml, 2006.

[12] P. Hnetynka, L. Murphy and J. Murphy, John, "Comparing the Service Component Architecture and Fractal Component Model", The Computer Journal, Vol. 54 Issue 7, July 2011.

[13] P. Maes, "Concepts and Experiments in Computational Reflection", In Proceedings of ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA' 87), Orlando, FL USA, October 1987, pp.147-155.

[14] S. Poonguzhali, R. Sunitha, G. Aghila, "Self-Healing in Dynamic Web Service Composition". In International Journal on Computer Science and Engineering, Vol. 3, No. 5. (2011), pp. 2054-2060.

[15] Liliana Ardissono, Luca Console, Anna Goy, Giovanna Petrone, Claudia Picardi, Marino Segnan, "Enhancing Web Services with Diagnostic Capabilities". In ECOWS '05 Proceedings of the Third European Conference on Web Services.