A Generative Approach for the Adaptive Monitoring of SLA in Service Choreographies

Antonia Bertolino, Antonello Calabrò, and Guglielmo De Angelis

CNR–ISTI, Pisa, Italy
{antonia.bertolino,antonello.calabro,guglielmo.deangelis}@isti.cnr.it

Abstract. Monitoring is an essential means in the management of service-oriented applications. Here, event correlation results crucial when monitoring rules aim at checking the exposed levels of Quality of Service against the Service Level Agreements established among the choreography participants. However, when choreographies are enacted over distributed networks or clouds, the relevant monitoring rules might not be completely defined a-priori, as they may need to be adapted to the specific infrastructure and to the evolution of events. This paper presents an adaptive multi-source monitoring architecture synthesizing instances of rules at run-time and shows examples of use on a demonstration scenario from the European Project CHOReOS.

Keywords: Monitoring, Choreographies, Complex Event Processor, SOA, SLA, QoS.

1 Introduction

Service choreographies specify the intended interaction protocol among a set of cooperating services at the application business level [1]. With services becoming more and more pervasive and critical in everyday life and business, increasing importance assumes the quality exposed by those interactions. The agreed levels of Quality of Service (QoS) between the involved parties form the Service Level Agreements (SLAs). Hence, service choreographies are often augmented with notations expressing the non-functional properties that the choreographed service should abide by [2]. As a consequence, SLA monitoring and assessment become essential assets of any environment supporting choreography enactment.

Within the context of SOA, in order to effectively detect unexpected or undesirable behaviors of services, locate the origin of the issue, or even predict potential failures it is generally necessary to track, combine, and analyze events occurring at different abstraction levels. Therefore, in contrast with the use of more monitors operating in separate contexts, a promising strategy that is investigated in the literature is to architect SLA monitoring solutions able to reveal or predict run-time anomalies due to the combination of phenomena originated from sources operating at different levels [3]. We have recently developed [4] a monitoring architecture supporting the SLA monitoring of service choreographies from multiple sources, namely the infrastructure and the business layers, within the scope of the CHOReOS project[1].

The above mentioned multi-source monitoring solution, though, had not been conceived to deal with the continuous dynamic evolution that is typical of service compositions. In a context in which services may dynamically appear and disappear and are dynamically bound, it is reasonable to assume that the same SLA requirements to be monitored may also evolve, even as a reaction to some occurring event or situation that cannot be a-priori known. Moreover, the deployment and the execution of applications on highly dynamic Cloud infrastructures introduce further requirements of adaptability with respect to monitoring. Such requirements must be directly addressed by developers, providers, and maintainers of the choreography-based applications [5].

In this paper we present a monitoring infrastructure that improves on [4] by supporting the dynamic evolution of SLA monitoring rules. Specifically, our contribution is a *generative approach for the adaptive multi-source monitoring of SLAs in service choreographies.*

The rest of the paper is structured as follows: Section 2 introduces our adaptive multi-source monitoring framework including a generative module for the monitoring rules making the architecture adaptable at run-time; Section 3 reports about a case study demonstrating the application of the approach; finally Section 4 draws the conclusions.

2 Adaptive SLA Monitoring

In [4] we presented a monitoring architecture based on Glimpse [6] that supported the SLA monitoring of service choreographies from multiple sources. In the following we refer to such starting configuration as a Multi-source Monitoring Framework.

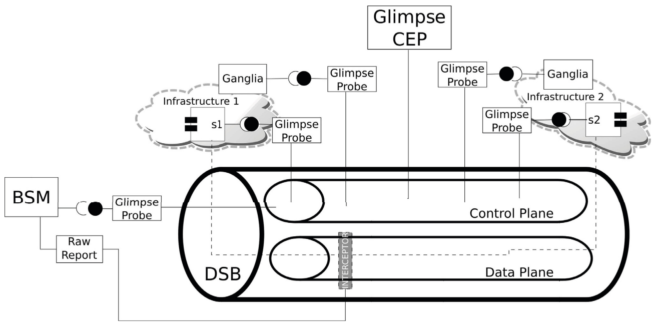

Fig. 1. Multi-source Monitoring Architecture

[1] See at `http://www.choreos.eu`

As shown in Figure 1, the configuration relies on a Distributed Service Bus (DSB) sharing distributed communication channels among the choreographed services. The DSB distinguishes between a set of channels on which both coordination and application messages flow (i.e. `Data Plane`), and another set dedicated to the monitoring activities (i.e., `Control Plane`). The data passing through the latter, can be correlated and analyzed by means of a Complex Event Processor (CEP).

Via the DSB, the Multi-source Monitoring Framework integrates three different monitoring facilities, each relative to a specific data source:
Infrastructure Monitor (IM): focuses on the status of the environment, providing support for the monitoring of resources, both in terms of their utilization and health status.
Business Service-Oriented Monitor (BSM): is responsible for monitoring the coordination messages that the choreographed services exchange with each other on the Data Plane channels of the DSB, by means of distributed interceptors. Then, BSM analyzes the temporal sequence of those events, checks the compliance of the SLA in the choreography specification, and, if any violation is found, it notify over the Control Plane.
Event Monitor (EM): refers to a generic event-based monitoring infrastructure able to bridge the notifications coming from the other two sources. Specifically at this level the other two kind of sources are wrapped by means of Glimpse Probes that forward notifications to the Glimpse CEP where they are processed and correlated.

Let us now refer to a scenario, such as the one that is emerging within the context of the Cloud paradigm, in which a solution that relies on the off-line definition of the monitoring rules appears not effective, as it is not thinkable to foresee a-priori the actual instantiation of the configurations. In fact, in the Cloud computing model enterprises provide infrastructures (e.g. machines) on-demand by allocating the exact amount of resources the customers need to use. Therefore, the information about the nodes available, and the mapping of the services on them becomes available only at run-time. Both the monitoring infrastructure and the correlation rules should deal with such dynamic contexts and adapt themselves according to the evolution of the deployment context.

To address such need, in this paper we propose a novel adaptive configuration of the Multi-source Monitoring Framework that supports the definition of the monitoring rules at run-time: the latter are synthesized by means of techniques based on generative programming approaches [7]. In this sense, with respect to the high-level hierarchical configuration presented in Section 2, the main improvement toward adaptiveness at run-time is relative to the source Event Monitoring.

In any event-based monitor, a central element is the CEP, which is the rule engine that analyzes the primitive events, generated from some kind of probes, in order to infer complex events matching the consumer requests. There exist several rule engines that can be used for this task (like Drools Fusion, RuleML), and for the sake of space we do not focus on traditional aspects of a CEP [6].

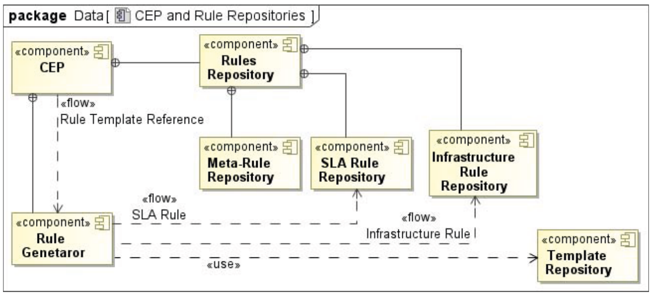
package Data [CEP and Rule Repositories]
«component» CEP
«component» Rules Repository
«component» Meta-Rule Repository
«component» SLA Rule Repository
«component» Infrastructure Rule Repository
«component» Rule Genetaror
«component» Template Repository
«flow» Rule Template Reference
«flow» SLA Rule
«flow» Infrastructure Rule
«use»

Fig. 2. Main Components of the CEP for the Adaptive Monitor

We focus instead on the specific components that support adaptiveness: as depicted in Figure 2, we have extended the CEP in its functionalities by including the sub-components: the Rules Repository, the Rule Generator, and the Template Repository.

The component Rules Repository abstracts the definition of three kind of repositories, each linking a dedicated kind of rule-set. Specifically, there is a repository storing the rules matching infrastructure events; a repository storing event rules about the SLA agreed among the choreographed business services; finally an additional repository storing the meta-rules enabling the run-time adaptation by means of generative procedures. A meta-rule is a special rule whose body implements the run-time synthesis procedure for populating both the SLA Rule Repository, and the Infrastructure Rule Repository.

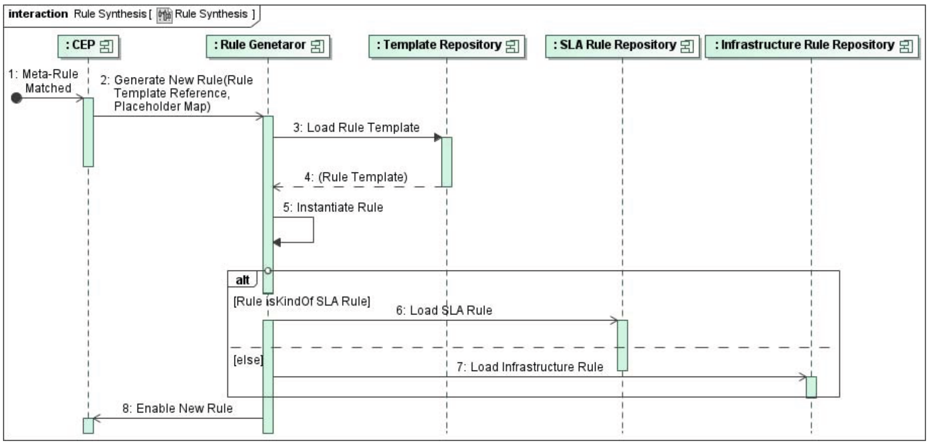

Fig. 3. Diagram of Interactions during Rule Synthesis

Figure 3 depicts a UML Sequence Diagram modeling the interaction schema that takes place among the traditional CEP and its new sub-components. Specifically, the rule generation is done in two steps. First, whenever a meta-rule within the CEP matches, it triggers the synthesis by the Rule Generator component. This will refer to the entries of the Template Repository relative to the kind of rules to be generated: precisely, a rule template is a rule skeleton, the specification of which has to be completed at run-time by instantiating a set of template-dependent placeholders. The Rule Generator will instantiate the latter with appropriate values inferred at run-time. Second, once the run-time synthesis of the new set of rules is completed, the Rule Generator loads the new rules into their corresponding repository (either SLA Rule Repository or Infrastructure Rule Repository) and enables them by refreshing the CEP's rule engine.

For the sake of completeness, we remark that both the SLA Rule Repository, and the Infrastructure Rule Repository can obviously also include sets of static rules that do not depend on the generative process discussed above.

3 Demonstration Scenario

The presented monitoring framework provides the facilities to adaptively detect and correlate events generated by different layers. In this section we show how this can help problems detection on a scenario referred by a choreography developed within the CHOReOS Project.

3.1 Scenario Description

In the following, the paper refers to the choreography "Manage Unexpected Arrival" from the "Passenger-Friendly Airport" [8]. For the sake of presentation with respect to the main contribution of the paper, the case study focuses on the monitoring activities of the task `Book Amenities` [8], and more specifically when the role `Airport` starts interacting with the other participants in the task (e.g. `Security Company`, etc.).

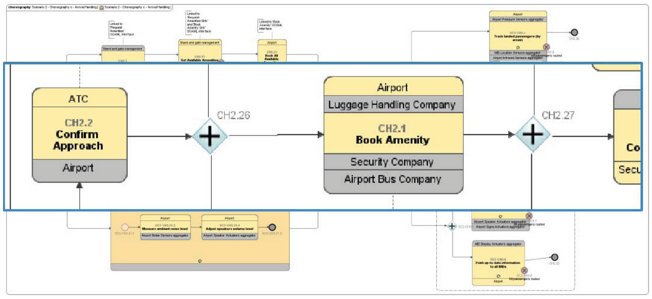

Fig. 4. The Passenger-Friendly Airport Use Case

Specifically, with respect to the interactions between the `Airport`, and the `Security Company`, the paper reports how to combine the run-time assessment of the QoS by the BSM with the information provided by the IM referring to the status of the nodes hosting the services.

Within the configuration of the scenario the infrastructural nodes were equipped on-purpose with means (i.e. "Load Knob") for injecting artificial disruptions by overloading them.

The components of the Multi-source Monitoring framework were distributedly deployed on dedicated nodes (i.e. hosting the DSB, the CEP, and the BSM). According to the configuration presented in Section 2, the scenario included a set of probes (i.e. Glimpse Probes) notifying either violations of SLAs at business service level, or information about the status of the nodes in the cloud hosting the services.

In addition, the BSM has been configured to intercept events on the `Data Plane`, while the CEP and the Glimpse Probe were bound to the `Control Plane`. Finally, an SLA regulating the latency of the interactions between the participants `Airport`, and `Security Company` has been loaded and activated within the BSM.

3.2 Execution and Adaptation

Within this case study we assumed that the rule knowledge base of the CEP has been instructed with a meta-rule specifying the action/countermeasure to activate if an SLA violation message occurs. We are assuming that the action depends on the specific machine where the violation occurred, and moreover it varies for the two different configurations about the monitored notifications: 1) SLA violation && node overload; 2) SLA violation && node not overloaded.

When the BSM reveals that an SLA violation has occurred, its associated Glimpse Probe sends a warning to the CEP. According to the generative process described in Section 2, the CEP first interacts with an internal registry associated with the `Data Plane` of the DSB in other to identify the IP address of the machine running the specific instance of the service that violated the SLA; then, its Rule Generator component synthesizes and enables a new rule looking for issues on the node hosting that service.

Listing 1 reports the auto-generated rule after an SLA violation of the service `Security Company` is raised to the CEP.

The generated rule is composed by two parts: the first begins at line 7, where the `$aEvent` represents the SLA Alert event sent by the BSM to the CEP. It is identified by the timestamp, a parameter checking if the event has been already managed by the CEP (i.e. `isConsumed`), and the name of the event. The second part begins at line 8, and represents the infrastructure event the Multi-source Monitoring framework looks for matching. Notably, this second part specifies a parameter called `getMachineIP` containing the IP address of the node that generated the infrastructure-level notification, which would be matched with the IP address retrieved from the SLA notification during the generation of the rule. In addition, such a declaration refers to a filter on the window frame within which the correlation should be considered valid (see at line: 8). Specifically Listing 1 specifies that two events can be correlated if `$bEvent` occurred within a 10 seconds interval after `$aEvent`.

```
1   <ComplexEventRuleActionList xmlns="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule"...>
2    <Insert RuleType="drools"><RuleName>
          SLA_violation_overload_Autogenerated_SecutiryCompanyService</RuleName>
3     <RuleBody>
4      rule "SecurityCompanyService_INFRASTRUCTUREVIOLATION"
5          ...  ...
6      when
7       $aEvent : GlimpseBaseEventChoreos( this.isConsumed == true, this.getTimeStamp == 1360752708858,
              this.getEventName == "SLA Alert - SecurityCompanyService");
8       $bEvent : GlimpseBaseEventChoreos(this.isConsumed == false, this.getEventName == "load_one",
             this.getMachineIP == "67.215.65.132", this after[0,10s] $aEvent);
9      then
10      $bEvent.setConsumed(true); update($bEvent);
11      ResponseDispatcher.LogViolation("...","auto_generated_rule", "\nSLA and Infrastructure violation by
              service: SecurityCompanyService" + "\npossibly due to an overload on machine: " + $bEvent.
             getMachineIP());
12      retract($aEvent); retract($bEvent);
13     end
14    </RuleBody>
15   </Insert>
16  </ComplexEventRuleActionList>
```

Listing 1. Generated Rule : SLA violation due to the overload of the hosting node

In the simulation case that no artificial overload is injected, the rule at Listing 1 applies, and a notification is dispatched to the service provider/administrator as potentially the violation may be due to the service itself. On the other hand, by querying the "Load Knob" on the node hosting the `Security Company` service , it is possible to inject some artificial disruption at the infrastructure level. In this case, when both the SLA violation on `Security Company` and a notification of an overload peak from the machine hosting it occur, the rule at Listing 1 matches. The assigned countermeasure is to dispatch a notification for redistributing some of the services active on that specific node onto some others nodes of the cluster.

4 Conclusion and Future Work

Adaptability is a key problem in the distributed and dynamic environments subsumed by the paradigm of the service choreographies. Specifically, as choreographies are abstract specifications, they may include interaction schema that can evolve after the design phase, so that unexpected events or scenarios may actually take place at run-time.

In addition, adaptability is a crucial asset for monitoring infrastructures correlating phenomena originated from sources operating at different abstraction layers; for example trying to understand the causes of run-time anomalies such as the SLA violations among participants of a choreography. In these contexts the dynamicity is even more evident when the participants are executing in a distributed cloud-based infrastructure.

In this work we extended the Multi-source Monitoring framework originally introduced in [4] with features supporting the adaptive generation of the monitoring rules at run-time. Other works already exist, e.g. [3], and [5], arguing that SOA monitoring cannot address separately layer-specific issues. Moreover, the authors of both [9], and [10] previously considered that the monitoring activity can be enhanced with adaptation. On the one hand, these frameworks mainly refer to orchestrated service compositions while we focused on decentralized and message-oriented scenarios that are typical of service choreographies. On the other hand, our architecture refers to "adaptiveness" as a mean to deal with configurations/scenarios that cannot be completely specified either at design, or deployment time. The work and the application case study have been developed as part of the demonstrators of the CHOReOS project.

An interesting aspect of [9] that our work did not consider yet concerns the verification of the consistency between the run-time generated rules and the ones already loaded within the CEP. We are interested in supporting means ensuring such kind of consistency for the Multi-source Monitoring framework.

Acknowledgments. This work is part of the European Project FP7 IP 257178: CHOReOS. We thank the colleagues from Linagora and University of São Paulo for their contribution to some components of the Multi-source Monitoring Framework.

References

1. Barker, A., Walton, C.D., Robertson, D.: Choreographing Web Services. IEEE T. Services Computing 2(2), 152–166 (2009)
2. Bartolini, C., Bertolino, A., Ciancone, A., De Angelis, G., Mirandola, R.: Non-Functional Analysis of Service Choreographies. In: Proc. of the Workshop on Principles of Engineering Service Oriented Systems. IEEE-CS (June 2012)
3. Guinea, S., Kecskemeti, G., Marconi, A., Wetzstein, B.: Multi-layered Monitoring and Adaptation. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2012. LNCS, vol. 7084, pp. 359–373. Springer, Heidelberg (2011)
4. Ben Hamida, A., Bertolino, A., Calabrò, A., De Angelis, G., Lago, N., Lesbegueries, J.: Monitoring service choreographies from multiple sources. In: Avgeriou, P. (ed.) SERENE 2012. LNCS, vol. 7527, pp. 134–149. Springer, Heidelberg (2012)
5. Katsaros, G., Kousiouris, G., Gogouvitis, S.V., Kyriazis, D., Menychtas, A., Varvarigou, T.: A Self-adaptive hierarchical monitoring mechanism for Clouds. JSS 85(5), 1029–1041 (2012)
6. Bertolino, A., Calabrò, A., Lonetti, F., Di Marco, A., Sabetta, A.: Towards a Model-Driven Infrastructure for Runtime Monitoring. In: Troubitsyna, E.A. (ed.) SERENE 2011. LNCS, vol. 6968, pp. 130–144. Springer, Heidelberg (2011)
7. Czarnecki, K., Eisenecker, U.W.: Generative programming - methods, tools and applications. Addison-Wesley (2000)
8. Chatel, P., Vincent, H. (eds.): Passenger Friendly Airport Services Choreographies Design. Number Del. D6.2. The CHOReOS Consortium (2012)
9. Contreras, R., Zisman, A., Marconi, A., Pistore, M.: PRadapt: A framework for dynamic monitoring of adaptable service-based systems. In: Proc. of the Workshop on Principles of Engineering Service Oriented Systems, pp. 50–56 (June 2012)
10. Wetzstein, B., Karastoyanova, D., Kopp, O., Leymann, F., Zwink, D.: Cross-organizational process monitoring based on service choreographies. In: Proc. of the Symposium on Applied Computing, pp. 2485–2490. ACM (2010)