2013 7th IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)

Adaptive SLA Monitoring of Service Choreographies Enacted on the Cloud

Antonia Bertolino, Antonello Calabrò, Guglielmo De Angelis
Istituto di Scienza e Tecnologie della Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche
Pisa – Italy
Email : {antonia.bertolino, antonello.calabro, guglielmo.deangelis}@isti.cnr.it

***Abstract*—The deployment and the execution of applications on dynamic Cloud infrastructures introduces new requirements of adaptability with respect to monitoring. Specifically, the governance of service choreographies enacted over Cloud-based solutions relies on the observation and analysis of events happening at different abstraction layers. Adaptability requirements are even more evident when monitoring deals with Service Level Agreements (SLA) established among the choreography participants. In fact, as the Cloud paradigm offers on-demand solutions as a service, often monitoring rules cannot be completely defined off-line. Thus also the monitoring infrastructure must keep track of the continuous evolution of the underlying environment, and adapt itself accordingly. This paper proposes an adaptive multi-source monitoring architecture that can synthesize on-the-fly SLA monitoring rules following the evolution of the Cloud infrastructure. We demonstrate the idea on a case study and discuss limitations as well as planned further advancements.**

***Keywords*-Monitoring; Choreographies; Complex Event Processor; SOA; SLA; QoS.**

I. Introduction

Cloud paradigm has become a wide-spread, and attractive means for both business, and technical organizations that aim to provide reliable, customized, and on-demand computing environments [18]. Specifically, its application supports the perception from Cloud users side of an infinite amount of computational/storage resources available [21]. Nowadays, companies or institutions tend to federate the software solution they provide, so that increase the quality of the offered services and have a better control over their interactions. By leveraging the Cloud paradigm, they can easily evolve their solutions with respect to the configurations, the deployment, and the number of replicating instances as much as their needs scale up, without the need of planning far ahead for provisioning [1].

However, with respect to this last scenario, an effective Cloud solution can only be achieved by governing, and ensuring the applied reconfigurations maintain the proper functioning over the whole integrated system. In this sense, we consider that federated companies/institutions should explicitly express and deal with the coordination models they intend to adopt [9].

Service choreographies are a means for specifying the intended interaction protocol among a set of cooperating services at the application business level. In other words, they model the peer-to-peer externally observable interactions that the choreographed services should put in place [2]. Thus, these specifications candidate themselves as a very suitable choice for modeling cross-organizational interactions acting over the Cloud.

In addition, the reconfigurations supported by a Cloud-based solution usually aims at preventing quality issues from any of those interacting parties. Thus, service choreographies are also often augmented with notations expressing the non-functional properties that the choreographed service should abide by [3]. Such agreed levels of Quality of Service (QoS) between the involved parties are then formulated in terms of more technical and monitorable Service Level Agreements (SLAs) [10].

Within the context of Service Oriented Architectures (SOA) over the various layers implementing the Cloud paradigm, the effective detection of an undesirable quality leak from a service within a choreography generally relies on tracking, combining, and analyzing events occurring at different abstraction levels. Furthermore, to locate the origin of the issue, or even to predict some potential failures, more fine-grained information would be required. 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 integrated SLA monitoring solutions able to reveal or predict run-time anomalies due to the combination of phenomena originating from sources operating at different levels [15], namely the infrastructure, and the business layers. Specifically, the correlation of information belonging to different layers allows for holistic choreography governance and supports the effective enforcement of recovery policies in order to assure the agreed QoS levels.

Within the CHOReOS project[1], we developed a monitoring architecture supporting the SLA monitoring of service choreographies from multiple sources [4]. However, such a framework had not been originally conceived to deal with the unplanned evolution of the monitored system. In fact, the specification of monitoring rules was a-priori (statically) defined. Such a limitation prevented its application within scenarios relying on the Cloud paradigm.

[1] see at http://www.choreos.eu

978-1-4673-4889-8/13/$31.00 ©2013 IEEE

Indeed, the deployment and the execution of applications on highly dynamic Cloud infrastructures introduces requirements of adaptability with respect to monitoring. In fact, the monitoring infrastructure must keep track of the continuous evolution of the underlying environment hosting the choreographed services, and must adapt itself accordingly. Notably, as the Cloud paradigm offers on-demand solutions as a service, often those changes cannot be completely defined off-line. Moreover, in a context in which both services, and service instances may dynamically appear or disappear, and where they may be dynamically bound, it is reasonable to assume that the monitored SLA may itself evolve as well. This may be needed, for instance, as a reaction to some occurring event or situation that cannot be a-priori known.

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*. A preliminary shorter description of this adaptive version of the monitor appeared in [7]. This paper improves on [7] in several respects: the description of the approach provides more details, and its application to a case study developed within the CHOReOS project is outlined. Also, a critical discussion about the open issues of the multi-source monitoring of service choreography enacted over the Cloud is included.

The rest of the paper is structured as follows: Section II introduces our Multi-source Monitoring Framework formerly presented in [4]; Section III extends the latter with a generative module for the monitoring rules making the architecture adaptable at run-time; Section IV reports about a case study demonstrating the application of the approach; Section V arguments the limitations and the potential side-effects that the current version of this monitoring solution may suffer. The paper concludes with Section VI presenting related work and Section VII drawing the conclusions.

II. SLA Monitoring of Choreographies

In this section we describe the existing solution on which the adaptive monitor proposed in this paper is based. Specifically, in [4] we presented a monitoring architecture based on Glimpse [8] 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.

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

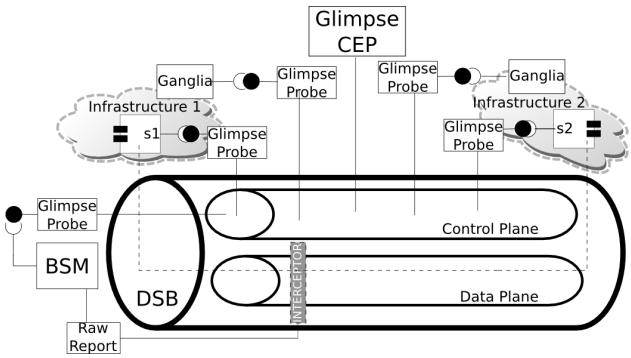

Figure 1. Multi-source Monitoring Architecture

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 Monitoring: 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 Monitoring: 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, the BSM analyzes the temporal sequence of those events, checks their compliance to the SLA in the choreography specification, and, if any violation is found, this is notified over the Control Plane.

Event Monitoring: 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 by which they are processed and correlated.

Even though the SLA violations could manifest themselves in similar ways, actually their causes can vary and may require different actions for their mitigation. In this sense, the Multi-source Monitoring Framework is quite useful, as it can combine information from different abstraction levels and infer the reason why the violation occurred.

Specifically, the violation could be due to either the current status of the infrastructure hosting some of the services involved in the interaction, or rather to the implementation of some services. In the former case, the Multi-source Monitoring Framework may reveal that the SLA has been violated due to either an overload, or a crash of the node hosting the service. A reaction to this scenario may foresee

the migration of the service to another (more powerful, more reliable) node. In the latter case, the Multi-source Monitoring Framework reveals that the SLA has been violated even though the node is available and is not overloaded. Here, the reaction can be the notifications of a request for the deployment of an updated revised version of some of the services involved.

III. Adaptive SLA Monitoring

As said, the Multi-source Monitoring Framework aims at correlating information from business and infrastructure layers, thus enabling different notifications according to whether the SLA violation occurred due to the business services, or to the hosting infrastructure.

Let us consider a canonical scenario in which the nodes composing the infrastructure level are known at pre-deployment time. In addition, the information about their topology and about what services each machine is hosting is also a priori available. In this context, a set of static (i.e., defined once and for all) monitoring rules for detecting the relevant events according to the deployment configuration can be defined off-line.

Let us now refer to a different scenario, such as the one that is emerging within the context of the Cloud paradigm: in this case 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 become 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 [14]. In this sense, with respect to the high-level hierarchical configuration of three sources presented in Section II, 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 [19]. There exist several rule engines that can be used for this task (like Drools Fusion, RuleML); for the sake of space we do not focus on the traditional aspects of a CEP [8]. We focus instead on the specific components that support adaptiveness: as depicted in Figure 2, we have extended the

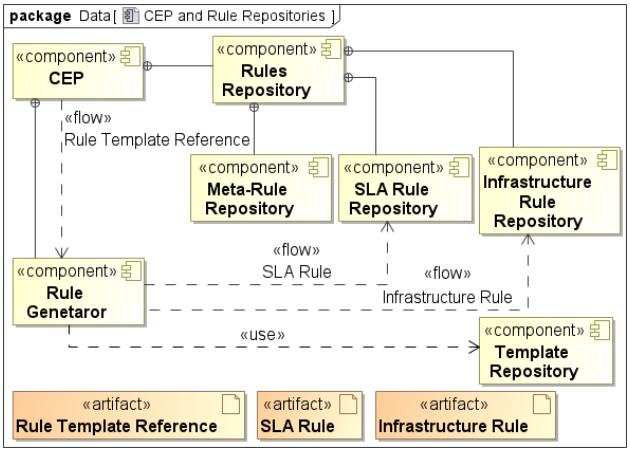

Figure 2. Components for the Adaptive Monitor

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.

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.

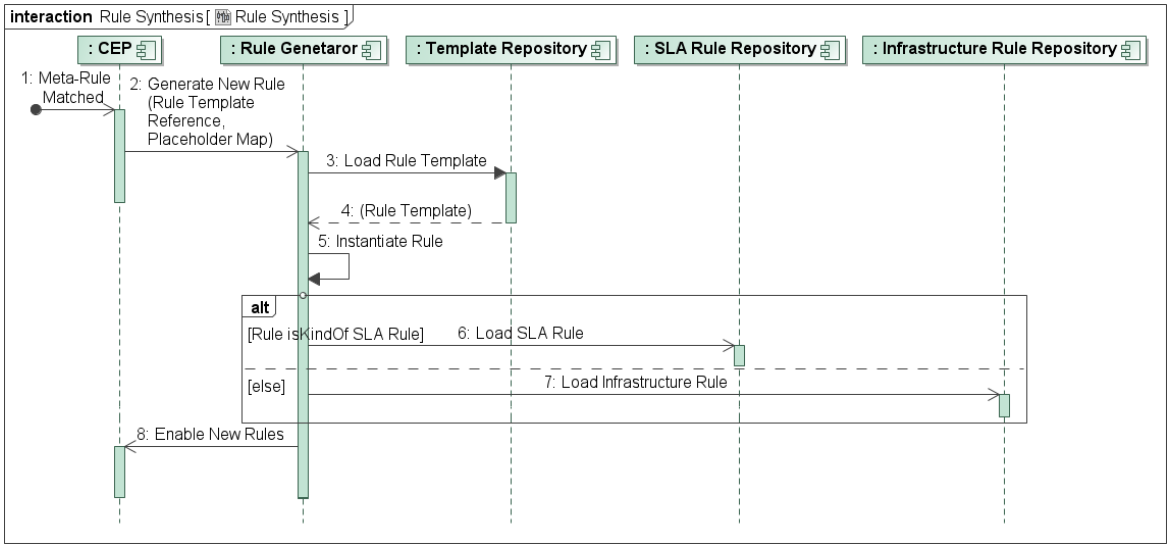

Figure 3. Diagram of Interactions during Rule Synthesis

IV. 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 problem detection on a scenario involving a realistic choreography. A live demo about such case study has been recorded in a video, which is available along with the CHOReOS Project Demos at http://www.choreos.eu/bin/Discover/videos (i.e., Monitoring and V&V Planning).

Note that the following demonstration scenario is supposed to be executed on some kind of private IaaS/PaaS solution [27]. In such a context, only the Cloud Provider is able to access the information about the available nodes, their topology, and the actual deployment policies.

The scenario hence refers to the case that a Cloud Provider is exploiting the features offered by the Multi-source Monitoring Framework in order to understand why a SLA violation occurred. The overall demonstration foresees that eventually, using the monitor feedback, the Cloud Provider can react to the SLA violation by balancing the infrastructure resources provided, or by adapting the configurations of the services deployed on top of the hardware nodes, or indirectly by contributing to detect some non-functional inadequacy of the hosted software services.

A. Scenario Description

In the following, we refer to a choreography from the "Passenger-Friendly Airport" use case developed within the CHOReOS project [11]. Specifically, this section focuses on the choreography "Manage Unexpected Arrival" that handles the arrival of passengers redirected from other planned destinations. Among the other tasks, the choreography models the reservation of facilities for managing the arrival of the plane.

For the sake of keeping the presentation simple, in order to illustrate the contents we are contributing in this paper, the case study focuses on the monitoring activities (both at infrastructure, and at business level) performed within a specific task of such choreography (i.e., `Book Amenities` in [11]). More in detail, after the confirmation of an unexpected landing from the `Air Traffic Control`, the task `Book Amenities` becomes active and the role `Airport` starts interacting with the other participants in the task (e.g. `Security Company`, etc.).

With respect to the interactions between the `Airport`, and the `Security Company`, the simulation shows 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.

B. Deployment and Configuration

A distributed demo of the above scenario has been designed. Specifically, as reported in Figure 4, the services participating in the choreography were deployed on different nodes: an instance of `Airport` on the node `lua.eclipse.ime.usp.br`, and an instance of `Security Company` on the node `vinhedo.eclipse.ime.usp.br`.

Within the configuration of the scenario both nodes were equipped on-purpose with means (i.e., the "Load Knob" answering on port 34567) for injecting artificial disruptions at the infrastructure level by overloading the node hosting them.

The components of the Multi-source Monitoring Framework were also deployed in distributed way.

Specifically: the DSB has been deployed on the node `atlantis.isti.cnr.it`, the CEP on the node `reposto.isti.cnr.it`, the BSM on the node `avalon.isti.cnr.it`. According to the configuration presented in Section II, 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.

C. Execution and Adaptation

For this demonstration 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

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

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

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 III, 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 11, where the `$aEvent` represents the SLA Alert event sent by the BSM to the CEP. It is identified by the timestamp, a parameter checking if the event has been already managed by the CEP (i.e. `isConsumed`), and the name of the event. The second part begins at line 12, and represents the infrastructure event the Multi-source Monitoring Framework looks for matching. Notably, this second part specifies a parameter called `getMachineIP` containing the IP address of the node that generated the infrastructure-level notification, which would be matched with the IP address retrieved from the SLA notification during the generation of the rule. In addition, such a declaration refers to a filter on the window frame within which the correlation should be considered valid (see at line: 12). Specifically Listing 1 specifies that two events can be correlated if `$bEvent` occurred within a 10 seconds interval after `$aEvent`.

```
1  <ComplexEventRuleActionList xmlns="http://labse.isti.cnr.
       it/glimpse/xml/ComplexEventRule"...>
2   <Insert RuleType="drools">
3    <RuleName>
4    SLA_violation_Auto_SecutityCompanyService
5    </RuleName>
6    <RuleBody>
7       ... ...
8       rule "SecurityCompanyServiceNOINFRASTRUCTURE"
9       ... ...
10    when
11     $aEvent : GlimpseBaseEventChoreos(this.isConsumed ==
            true, this.getTimeStamp == 1360752984631, this.
            getEventName == "SLA Alert -
            SecurityCompanyService") not(
            GlimpseBaseEventChoreos(this.isConsumed == false,
            this.getEventName == "load_one", this.getMachineIP
             == "67.215.65.132" , this after[0,10s] $aEvent));
12    then
13     ResponseDispatcher.LogViolation("...","
            auto_generated_rule", "\nSLA violation\noccurred
            on: SecurityCompanyService");
14    retract($aEvent);
15   end
16  </RuleBody>
17 </Insert>
18 </ComplexEventRuleActionList>
```

Listing 2. Generated Rule : SLA violation due to an anomaly at Business Service level

Similarly, the meta-rule synthesized an additional rule matching the case an SLA notification has been received but no notification came from the infrastructure layer within a given time-frame (see Listing 2).

In the simulation case that no artificial overload is injected, the rule at Listing 2 applies, and a notification is dispatched to the service provider/administrator as po-

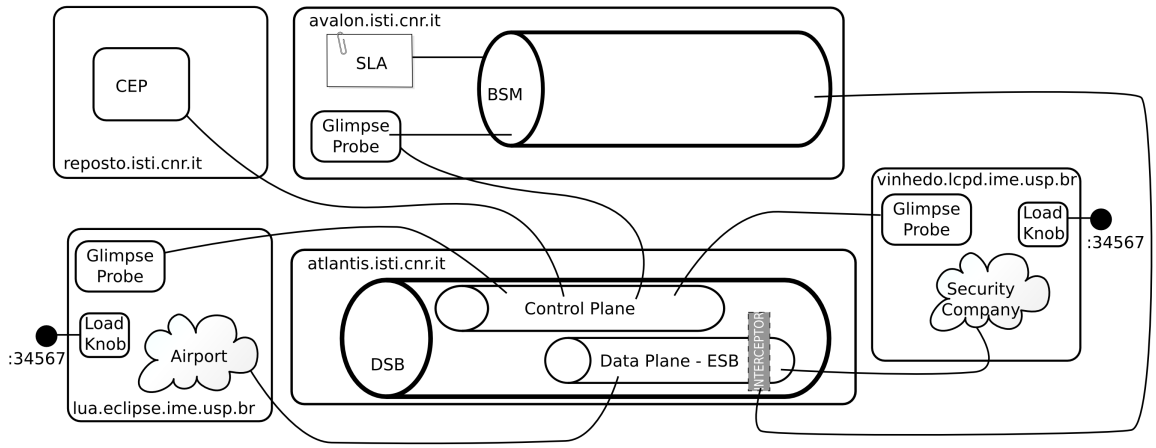

Figure 4. Deployment Configuration

tentially the violation may be due to the service itself. On the other hand, by querying the "Load Knob" on the node `vinhedo.eclipse.ime.usp.br`, 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 on the machine hosting 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.

For the sake of completeness, we remark that in a real-life application scenario, it could happen that a node gets stuck so that it cannot send any infrastructure notification to the rest of the monitoring Framework, even within the "huge" window frame we set (e.g. 10 seconds). In those scenarios, a more complex set of meta-rules and rule templates should be considered, but this is left out of the illustrative scope of this scenario.

V. Discussion

The essential goal of monitoring is the run-time checking of relevant properties, either functional, or qualitative. As described above, such run-time checking involves the collection, analysis, and display of interactions among managed objects so to timely detect any possible deviations from the expected behavior.

However, it is intuitive that any kind of on-line intervention on a running system could adversely impact its characteristics. The precise impact clearly depends on the specific instantiation of the infrastructure, and on the strategy that would be applied on-line [6]. Hence, when presenting monitoring infrastructures, the overhead caused by the observation/checking mechanism itself, which in fact may consume valuable resources, must be considered. This concern become especially relevant when the monitoring infrastructure aims at observing the non-functional properties exposed by a system.

Accordingly, in this section we analyze the potential costs introduced by our adaptive Multi-source Monitoring Framework, and even though we do not provide yet a numerical assessment, we start a critical discussion by presenting some dimensions, issues, and mitigation strategies associated with the overhead of the proposed solution. We first focus on general aspects of a Multi-source Monitor, and then consider specific concerns for the newly added aspect of generative rules.

A. How intrusive is the approach?

Usually, monitoring the interactions among distributed and multi-tenant systems imposes some kind of additional requirements from the observed entities, for the purpose of allowing the collection of relevant data. For example, such requirements would concern exposing evidences of the performed activities, with respect to those tasks implementing the overall cooperation between participants as specified from a global viewpoint.

Even though in some domains the implementation of additional requirements may be perceived as intrusive, within the context of service choreographies that this paper refers to, we consider such aspect as unavoidable towards the achievement of cross-organizational business goals and hence we do not see it as a major issue. Specifically, service choreographies model explicit interactions that cooperating entities should put in place [2]. Thus, such multi-party collaboration is natively built on top of externally observable events describing the "public process" [25] each participant implements.

In this sense, similarly to other works [25], in our framework we assume that monitored parties are somehow willing to support such an activity, for example by adopting shared communications channels (i.e. the `Data Plane` in Figure 1), or by enabling local probes feeding the informa-

tion considered useful with respect to the objectives of the monitoring session.

Hence, a kind of "powerful" monitor such as the one we propose here can be seen as intrusive, however in a dynamic and loose coupled eco-system such as a service choreography some form of discipline and governance is intrinsic and necessary [26] [22], therefore we believe the implied burden will be acceptable.

B. What is the estimated overhead?

In tackling this question, different important considerations can be done. We thus decompose the question in subtopics addressing different facets.

First of all, the number of monitoring messages exchanged during an enactment of a choreography could be many and could somehow impact the performance of the monitored system. Even though we have not yet mature empirical results assessing how the Multi-source Monitoring Framework scales up with respect to the number of monitored elements, we can argument that, as described in Section II, the messages originated/consumed by the monitoring framework are managed separately from the messages exchanged among the choreography participants.

In this sense, the risk of monitoring overhead can be mitigated by decoupling the dimensioning of the DSB according to the estimated workloads foreseen on either the `Data Plane`, or the `Control Plane`, respectively. Such dimensioning would be actualized by means of the proper replication, interconnection, and configuration of several instances of the DSB. As detailed in [4], the communication backbone has been designed as a transmission layer referring to the publish-subscribe paradigm. Off-the-shelf middleware implementing the functionalities of an ESB[2] would support this aspect as they natively feature the federations of distributed set of entities, and the proper flowing of information among them.

Another aspect concerns the business messages intercepted on the `Data Plane`. These originate events that are forwarded to the BSM for checking whether the specified SLAs have been honored. The interception mechanism provides the minimal information about the messages exchanged. In our reference implementation, it takes place at the transport level within the ESB, and has been efficiently implemented so to add a constant computational effort that is usually negligible with respect to the other transportation steps adopted within an ESB (e.g. flooding each available channel with the messages it is supposed to collect) Hence, the overhead BSM interception introduces is at most linear with the number of the messages flowing in the communication bus.

One further aspect potentially impacting the performance of the enacted choreography concerns the collection of the data by the IM. Specifically, on the one hand a massive usage of such kind of messages may overload the resources assigned to the `Control Plane`. Nevertheless, as described above, those communication channels are implemented on dedicated resources different from the `Data Plane`. This aspect could be tackled by a proper dimensioning of those communication channels.

On the other hand, the observation on the infrastructure layer could be wrongly considered as a possible contributor to the overhead introduced by the monitoring framework. In fact, the monitoring of both resources usage and its health status of each node is widely acknowledged to be very low, and may be considered negligible [20].

C. Is the synthesis process affordable?

This section discusses some potential issues relative to the automatic synthesis of the rules that can be loaded on the CEP at run-time.

A first consideration refers to the latency of the rule generation process. Since the very instant in which a meta-rule matches within the CEP, until the instant when a set of newly generated rules become active in the Rules Repository, some time elapses and some effort is spent. Clearly this is a critical aspect with respect to the sustainability of the approach we are proposing. In fact, a generative process that did not react sufficiently fast, would imply that when the correlation mechanism we are advocating would start to monitor unforeseen events, it could be too late since they may have already happened.

More precisely, the model-to-code generation process itself should not rise relevant issues with respect to the delays it may introduce; the main concern refers to the interaction with the external repositories in order to query and extract the information relevant for the generation process. This would be especially true if the number of SLAs and the infrastructure nodes scale up.

The mitigation of this concern would also require the optimization of the interactions with the rule repositories. Nevertheless, borrowing an idea from the community working on real-time systems [13], a more interesting solution could rely on the adoption of the explicit definition of the deadlines for the generation process. In such a way it would be possible to implement the rule generation process with real-time scheduling strategies for the purpose of properly handling the execution of those activities that would spend too much time and miss their deadline.

A second consideration affects the CEP when the number of the rules generated at run-time grows. Specifically, as the number of services increases, the number of SLAs increases. Similarly, also the size of the cloud infrastructure would increase. All those dimensions would in turn increase the number of rules loaded on the CEP, thus the performance of the rule matching process may drop. A candidate solution could adopt a deployment strategy relying on a distributed

[2]In our reference implementation, it is the Linagora's EasyESB (see https://research.linagora.com/display/easyesb/)

topology of several CEP instances coordinated by means of publish/subscribe notifications on the `Control Plane` of the DSB.

In conclusion, the run-time generation of the rules might become an issue as the various above discussed dimensions of the monitored system grow up. However, various countermeasures could be adopted. For the sake of completeness, we remark that even though these issues are perceived to be crucial assets for our Multi-source Monitoring Framework, the prototype we report on in this paper does not yet include such features that will be part of our future work.

VI. Related Work

Self-adaptation mechanisms refer to those parts of a system that aim at modifying its behavior by dynamically addressing goals that may continuously emerge or evolve. Nevertheless, the authors of [23] noticed that most of the self-adaptation approaches proposed in the literature act only on the controlled system and do not also adapt themselves (as we do with the monitoring infrastructures).

In this sense, the DYNAMICO reference model [24], and its implementation [22] contributed to a context-aware framework realizing the adaptation of both the target monitored system, and the dynamic monitoring infrastructure itself. Specifically, the overall idea proposed in [22] aims at providing a SLA monitoring framework that is able to deal with evolving agreements, and thus with adaptation strategies in order to monitor and to guarantee them.

Even though DYNAMICO addresses interesting, and related topics, the goal of our research is slightly different. In fact, we mainly investigate how correlation of monitored events can justify the fact that a SLA violation occurred. In this sense, the adaptation layer we introduced within the Multi-source Monitoring Framework can be seen as an alternative instantiation of the monitoring feedback loop (i.e. M-FL) introduced in [22].

In parallel, also [12] considered that the monitoring activity can be enhanced with self-adaptation means. Specifically, the authors in [12] presented PRadapt as a framework for dynamic monitoring of adaptable service-based systems supporting the run-time synthesis of monitoring rules as a reaction to changes in the context and the system. Beyond the common idea, the two works mostly differ in the instantiation of the monitoring architecture. In fact, PRadapt explicitly models conceptual elements such as the `Execution Engine`, or the `Process Engine`, while our architecture mainly relies on the distributed communication channels used by both the different services participating in the composition and the monitoring infrastructure itself. In this sense, the framework in [12] seems mainly referring to orchestrated service compositions while we focus on decentralized, and message oriented scenarios that are typical in service choreographies.

Similarly to our work, the authors of [15] also report that in SOA monitoring should not address layer-specific issues separately. In fact, problems that are transmitted from one layer to the others cannot be captured and understood. Nevertheless, the work in [15] focused on the monitoring and adaptation of service orchestrations (i.e. BPEL processes) that are deployed onto a dynamic infrastructure. In contrast, our approach refers to choreographies where service aggregation is coordinated in a decentralized way and hence it cannot rely on any entity that is specifically responsible of enacting the choreography, or of restructuring and adapting any running instance.

Also in [16], a multi-layered service-oriented monitoring framework that focuses on both the platform and the infrastructure layer is presented. The primary goal of that approach is collecting and aggregating monitored information with regard to specific performance constraints. Two are the main differences with our framework. A first difference is that our Multi-source Monitoring Framework explicitly supports a correlation technique that makes use of complex event processing, while the approach in [16] mainly focuses in collecting and storing the monitored indexes in suitable repositories. As a consequence, the second difference: the solution in [16] aims at providing monitoring mechanisms where the adjective "adaptable" is mainly referred to the possibility of customizing the frequency of monitoring collection. A similar idea had been previously proposed also in [5]. In contrast, our architecture refers to "adaptiveness" as a mean to deal with configurations/scenarios that cannot be completely specified either at design, or deployment time. As described in Section III, in this paper the Multi-source Monitoring Framework has been extended in order to support the generation of monitoring-rules at run-time.

In [17] the authors proposed a monitoring approach that enables autonomous service provisioning in federated clouds. Among the others, the framework was mainly conceived to either support the deployment, or delegate the execution of the requested services as virtual machines on a specific IaaS. Thus the main difference with our approach is that such solution conceives the sources of the monitoring mainly as layers of the same kind. In our approach we combine events happening at different abstraction layers.

Another related work is [25], where an event-based approach to cross-organization monitoring based on service choreography descriptions is presented. Similarly to the CEP in our architecture, it relies on means for aggregating and processing events that have been distributively generated. The major difference with our approach is that they do not address the adaptation of the monitoring activity to the evolution of the context in which the choreography is executing. Finally, even though also this solution relies on the notion of SLAs, it does not refer to them as contracts focusing on QoS measurements (i.e. technical characteristics such as availability and response time), but rather as a means

for expressing a set of cross-organizational events that the cooperating participants are willing to provide, and that can be referred as process metrics for monitoring service choreographies.

VII. Conclusion and Future Work

Adaptability is a key requirement for the distributed and dynamic solutions implemented on the Cloud. Specifically, as the Cloud paradigm offers on-demand resources as a service, most often the changes it may subsume cannot be completely defined/predicted off-line.

As unexpected events or scenarios may take place at run-time, the need arises to rely on adequate monitoring solutions. In this sense, adaptability becomes also a crucial asset for monitoring infrastructures that have to correlate phenomena originated from sources operating at different abstraction layers. For example, complex event analyzers have to understand the causes of run-time anomalies such as the SLA violations among the participants of a choreography.

This paper extended the Multi-source Monitoring Framework as introduced in [4] with features supporting the adaptive generation of the monitoring rules at run-time for choreographies enacted on the Cloud.

The work and the case study have been developed as part of the demonstrators of the CHOReOS project. Future work will include the refinement of the implementation, and its further validation on improved versions of the use cases that are going to be released by the project.

An interesting aspect of [12] 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. Furthermore, we are planning to investigate how our research on monitoring and correlation of layered events can be combined with other adaptive frameworks such as the related approach presented in [22]. Finally, future work will also concern the adoption of mitigation strategies dealing with the issues discussed in Section V-C.

Acknowledgments

This paper describes work undertaken in the context of the European Project FP7 IP 257178: CHOReOS. We would like to acknowledge the collaboration with the CHOReOS partners in the development of the CHOReOS IDRE, and in particular with colleagues from Linagora and University of Sao Paulo for the contribution of some components of the Multi-source Monitoring Framework as described in Section II.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] A. Barker, C. D. Walton, and D. Robertson, "Choreographing Web Services," *Transaction on Services Computing*, vol. 2, no. 2, pp. 152–166, 2009.

[3] C. Bartolini, A. Bertolino, A. Ciancone, G. De Angelis, and R. Mirandola, "Non-Functional Analysis of Service Choreographies," in *Proc. of the Workshop on Principles of Engineering Service Oriented Systems*. IEEE-CS, June 2012.

[4] A. Ben Hamida, A. Bertolino, A. Calabrò, G. De Angelis, N. Lago, and J. Lesbegueries, "Monitoring Service Choreographies from Multiple Sources," in *Proc. of the Workshop on Software Engineering for Resilient Systems*, ser. LNCS, vol. 7527. Springer, 2012, pp. 134–149.

[5] A. Bertolino, G. De Angelis, S. Elbaum, and A. Sabetta, "Scaling up SLA Monitoring in Pervasive Environments," in *Proc. of the Workshop on the Engineering of Software Services for Pervasive Environments*. ACM, September 2007.

[6] A. Bertolino, G. De Angelis, S. Kellomäki, and A. Polini, "Enhancing service federation trustworthiness through online testing," *IEEE Computer*, vol. 45, no. 1, pp. 66–72, 2012.

[7] A. Bertolino, A. Calabrò, and G. De Angelis, "A Generative Approach for the Adaptive Monitoring of SLA in Service Choreographies," in *Proc. of the International Conference on Web Engineering*, ser. Lecture Notes in Computer Science, vol. 7977. Springer, 2013, pp. 408–415.

[8] A. Bertolino, A. Calabrò, F. Lonetti, A. Di Marco, and A. Sabetta, "Towards a Model-Driven Infrastructure for Runtime Monitoring," in *Proc. of the Workshop on Software Engineering for Resilient Systems*, ser. LNCS, vol. 6968. Springer, 2011, pp. 130–144.

[9] A. Bertolino, G. De Angelis, and A. Polini, "Governance policies for verification and validation of service choreographies," in *WEBIST (Selected Papers)*, ser. LNBIP, J. Cordeiro and K.-H. Krempels, Eds., vol. 140. Springer, 2012, pp. 86–102.

[10] A. Bertolino, G. De Angelis, A. Polini, and A. Sabetta, "Trends and research issues in SOA validation," in *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*, V. Cardellini, E. Casalicchio, K. Regina, J. L. C. Branco, J. C. Estrella, and F. J. Monaco, Eds. IGI Global, 2011, pp. 98–115.

[11] P. Chatel and H. Vincent, Eds., *Passenger Friendly Airport Services Choreographies Design*. The CHOReOS Consortium, 2012, no. Del. D6.2.

[12] R. Contreras, A. Zisman, A. Marconi, and M. Pistore, "PRadapt: A Framework for Dynamic Monitoring of Adaptable Service-based Systems," in *Proc. of the Workshop on Principles of Engineering Service Oriented Systems*, June 2012, pp. 50–56.

[13] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A Real-time Service-Oriented Architecture for Industrial Automation," *Transaction on Industrial Informatics*, vol. 5, no. 3, pp. 267–277, 2009.

[14] K. Czarnecki and U. W. Eisenecker, *Generative Programming – Methods, Tools and Applications*. Addison-Wesley, 2000.

[15] S. Guinea, G. Kecskemeti, A. Marconi, and B. Wetzstein, "Multi-layered Monitoring and Adaptation," in *Proc. of the International Conference on Service Oriented Computing*, ser. LNCS, vol. 7084, 2011, pp. 359–373.

[16] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A Self-adaptive Hierarchical Monitoring Mechanism for Clouds," *JSS*, vol. 85, no. 5, pp. 1029 – 1041, 2012.

[17] A. Kertész, G. Kecskemeti, C. A. Marosi, M. Oriol, X. Franch, and J. Marco, "Integrated Monitoring Approach for Seamless Service Provisioning in Federated Clouds," in *Proc. of International Conference on Parallel, Distributed and Network-Based Processing*. IEEE, 2012, pp. 567–574.

[18] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the Cloud? An architectural map of the Cloud landscape," in *Proc. of the Workshop on Software Engineering Challenges of Cloud Computing*, 2009, pp. 23–31.

[19] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison Wesley, 2002.

[20] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler, "Wide Area Cluster Monitoring with Ganglia," in *Proc. of the International Conference on Cluster Computing*. IEEE-CS, 2003, pp. 289–298.

[21] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *Communications Surveys and Tutorials*, vol. 13, no. 3, pp. 311–336, 2011.

[22] G. Tamura, N. M. Villegas, H. A. Müller, L. Duchien, and L. Seinturier, "Improving context-awareness in self-adaptation using the DYNAMICO reference model," in *Proc. of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. IEEE / ACM, 2013, pp. 153–162.

[23] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Proc. of the Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. ACM, 2011, pp. 80–89.

[24] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas, "DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems," in *Software Engineering for Self-Adaptive Systems II*, ser. Lecture Notes in Computer Science, R. Lemos, H. Giese, H. Müller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 265–293.

[25] B. Wetzstein, D. Karastoyanova, O. Kopp, F. Leymann, and D. Zwink, "Cross-organizational process monitoring based on service choreographies," in *Proc. of the Symposium on Applied Computing*. ACM, 2010, pp. 2485–2490.

[26] P. Windley, "SOA governance: Rules of the game," on line at http://www.infoworld.com, Jan. 2006.

[27] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *J. Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.