Enabling SLA Monitoring for Component-Based SOA Applications – A Component-Based Approach

Cristian Ruz, Françoise Baude, Bastien Sauvan
INRIA Sophia Antipolis, CNRS, I3S, Univ. de Nice Sophia Antipolis, France
{*cruz,fbaude,bsauvan*}*@inria.fr*

1. Motivation and General View

In service provisioning relationships, contracts relating to Quality of Service (QoS) are agreed in the form of a Service Level Agreement (SLA), which establishes conditions that must be met by the provider of the service. SLAs are often described as a set of Service Level Objectives (SLO). To watch the fulfilment of this contract, these SLOs must be monitored at runtime performing SLA monitoring [1].

The ability to collect QoS parameters is particularly important in Service Oriented Architectures (SOA), where the situation is that of a set of loosely coupled interacting heterogenous services from different providers. The monitoring and management task is thus intrinsically a distributed matter. Component-based approaches have emerged as a suitable alternative to traditional, purely service oriented ones (e.g. Web Services based orchestrations) to design and build SOAs, much strongly enforcing principles like reusability, encapsulation, composability, and interoperability. The Service Component Architecture (SCA) has emerged as a standard specification, technologically agnostic, for designing and building SOA applications, by modeling services as components which can be reached by other services or by external applications through the use of appropriate bindings. Runtime managements concerns, however, are not addressed by this specification and are left as a platform specific issue.

Monitoring the runtime behaviour of such applications and watch the compliance to an SLA is not an easy task. Online data must be extracted and QoS metrics must be kept up to date; the composing and bindings can change and new components may need to be monitored; the SLOs can also evolve at runtime. There is an overhead cost and a tradeoff must be made with respect to the amount of data to collect.

Several existing solutions are oriented to specific technologies like Web Services [2] or J2EE [3], or rely on measurements at the network level, and detect interactions by correlating events [4]. Other approaches while extensible and low intrusive can only be customized at design time [5].

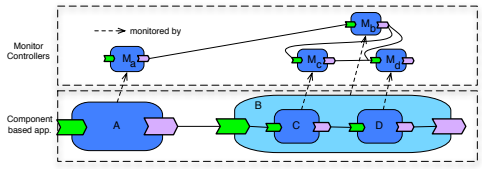

Figure 1. Components and Monitor Controllers.

In this work we intend to show that by implementing the monitoring task using a component-based approach we can provide a customizable, scalable and even adaptable architecture to exhibit the information required to fulfil SLAs.

2. Current Solution

We are proposing a solution that takes a component-based approach to obtain QoS parameter values. Our solution is built around a component that we call *Monitor Controller* that can be adapted to the monitoring requirements of an SLA. We have built an example where our solution can be used to obtain performance metrics and we plan to extend this approach to monitor other aspects of QoS that can be systematically computed, like dependability and trust.

At deployment time, each applicative component is deployed with a Monitor Controller attached, turning the component into a *monitored component*. The Monitor Controllers are interconnected forming a monitoring layer, to communicate the data they capture. This monitoring layer resembles the hierarchical structure of the application as shown on Figure 1.

The Monitor Controller leverages the information that can be obtained from the runtime platform. The collection of monitoring data can be based on interception of requests or by listening to events provided by the component platform.

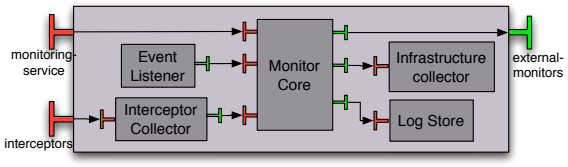

Figure 2. Internal Composition

2.1. Performance Monitoring

The Monitor Controller can be configured to collect performance metrics following, for example, the architecture shown on Figure 2. By storing the time taken by each request and communicating with the Monitor Controller of other components, it is possible to dynamically determine the components involved in the computation, the time spent on each one of them, as well as the time waiting for replies. From these measures, performance metrics can be computed for each component and a *request path* can be established for each request.

By measuring times both in the caller side and in the server side it is possible to determine also the latency of the calls. This differentiation is important in service selection when considering the QoS provided by a service. For example, a component can have a good average service time, but it may be accessible through a very slow network link, so it may not be a good choice against a similar component with bigger service time, but with lower latency.

2.2. SLA Monitoring

For monitoring the compliance to an SLA, another component called *SLA Monitor* con be connected to the Monitor Controller. The SLA Monitor uses the information collected to check the conditions specified by an specific SLO within an SLA, and can trigger an alarm if a condition is violated. The component-based approach allows to replace the SLA Monitor at runtime in case the SLOs evolve, to watch the new conditions. This configuration is exemplified on Figure 3.

For example, the SLA Monitor can be watching the SLO: "90% of the requests must be served within 2 secs.". At a certain moment, the SLO can change to, or add the condition: "No more than 1 sec. must be spent in network communication". The SLA Monitor component can be replaced for another one that is programmed to watch the new SLO.

2.3. Implementation

We have implemented a small example from Figure 4, using the GCM/ProActive middleware which

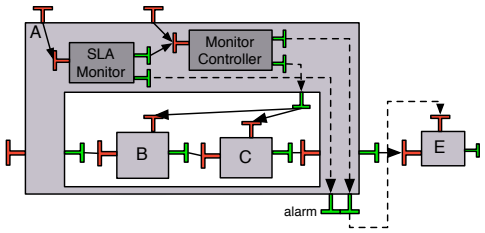

Figure 3. Location of the Monitor Controller

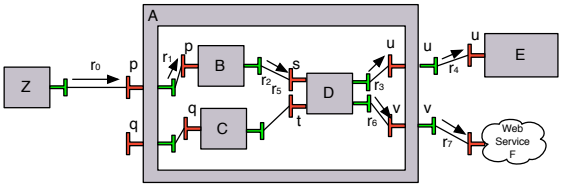

Figure 4. Request path. Client `Z` calls `p` on `A`

implements a component model with asynchronous communications, and which provides an SCA personality to interact with other SCA-based applications. In this example, we are able to obtain information from each component and reconstruct the service invocation path of any specific request, including the time spent in each component and determine a possible performance problem. The current experiments have shown that under a heavy load of applicative messages, the monitoring overhead is no bigger than 4%. We are working on extending this experiment with an SLA Monitor encompassing several and dynamically evolving SLOs.

References

[1] M. Schmid and R. Kröger, "Decentralised QoS-Management in Service Oriented Architectures," in *DAIS*, 2008.

[2] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar, "Comprehensive QoS monitoring of Web services and event-based SLA violation detection," in *MWSOC '09: Proc. of the 4th Int. Workshop on Middleware for Service Oriented Computing*. ACM, 2009, pp. 1–6.

[3] T. Parsons, A. Mos, and J. Murphy, "Non-intrusive end-to-end runtime path tracing for J2EE systems," *IEE Proceedings Software*, vol. 153, no. 4, p. 149, 2006.

[4] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling," in *OSDI'04: Proc. of the 6th Symposium on Operating Systems Design & Implementation*, 2004.

[5] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke, and W. Hasselbring, "Kieker: continuous monitoring and on demand visualization of java software behavior," in *SE '08: Proc. IASTED Int. Conf. on Soft. Engineering*, 2008.