Adaptive SLA Monitoring of Service Choreographies Enacted on the Cloud

Antonia Bertolino, Antonello Calabrò, and Guglielmo De Angelis

CNR-ISTI
about this presentation

» context
» multi source monitoring framework
» issues coming with the Cloud paradigm
» proposed enhancements
» case study
» conclusions
SOA promises support for the definition of cooperating entities that can be independently
- developed
- assembled
- managed

Two major aggregation paradigm emerge in this context
- service orchestration
- service choreography
service choreographies

- decentralized model for the aggregation of services
  - the cooperation lacks a “single” coordination point (i.e. the orchestrator)
  - highlights the interaction protocol
  - usually refers to abstract specifications without coupling them to concrete implementations

- core assets:
  - roles (that can be played by a service)
  - tasks (interaction points among roles)
  - messages (produced during a task)
  - control flows
services are black-box entities that are discovered, chosen and bound at run-time. Binding takes place based on:
- the choreography specification
- the role the services play in the choreography
- the exposed functional interface
- the negotiations of some QoS levels

Negotiations and run-time bindings leave space to scenarios that were not (completely) specified at design time.
service monitoring becomes essential in the management of SOA-based applications

crucial assets:

- **functional monitoring**: no centralized coordination actor --> observation of distributed sequence of messages
- **non-functional monitoring**: to check the SLA offered by a service with reference to the distributed QoS parameters it relies on

event correlation results interesting mean supporting monitoring activities
multi source monitoring

- enhances resilient approaches dealing with SLA violations in service choreographies

- locates the origin, or even predicts potential SLA failures
  - correlates the messages monitored at business-service level
  - with lower-level observations captured from the resources hosting the service application
multi source monitoring – abstract architecture –
the rules within the CEP

1. `<ComplexEventRuleActionList ...`
2. `<Insert RuleType="drools">
3. `<RuleName>MyRule</RuleName`
4. `<RuleBody`
5. `...`
6. `when`
7. `  $aEvent : /* RECEIVED SLA VIOLATION FOR S1 HOSTED ON H1 */`
8. `  $bEvent : /* RECEIVED KEEP-ALIVE NOTIFICATION N1 FORM H1 */`
9. `  $cEvent : /* RECEIVED KEEP-ALIVE NOTIFICATION N2 FORM H1 AFTER N1 */`
10. `then`
11. `  /* CONSUME THE EVENTS */`
12. `  /* APPLY OTHER REACTIONS TO THIS SCENARIO */`
13. `end`
14. `</RuleBody>`
15. `</Insert>`
16. `</ComplexEventRuleActionList>`
issues coming with the Cloud paradigm -1-

» service choreographies enacted on IaaS/PaaS

» only the Cloud Provider can access the information about the nodes, the topology, or the deployment policies

» a Cloud Provider could exploit the Multi-source Monitoring Framework for:
  » understanding why a SLA violation occurred
  » indirectly contributing to detect non-functional leaks of the hosted software services

» properly reacting to a SLA violation
  » balancing the infrastructure resources
  » by adapting the deployment of the services
HOWEVER: new monitoring requirements emerge from the enactment of service choreographies on the Cloud

- the paradigm offers on-demand solutions
- monitoring infrastructures must deal with the evolution of the Cloud configurations
- monitoring infrastructures must deal with the redeployment of the choreographies over the Cloud
- monitoring infrastructures must adapt themselves accordingly
multi source monitoring && the Cloud
how to describe rule?

1. `<ComplexEventRuleActionList ...>`
2. `<Insert RuleType="drools">`
3. `<RuleName>`MyRule`</RuleName>`
4. `<RuleBody>`
5. `...`
6. `when`
7. `$aEvent :` /* RECEIVED SLA VIOLATION FOR S1 HOSTED ON H1 */
8. `$bEvent :` /* RECEIVED KEEP-ALIVE NOTIFICATION N1 FORM H1 */
9. `$cEvent :` /* RECEIVED KEEP-ALIVE NOTIFICATION N2 FORM H1 AFTER N1 */
10. `then`
11. `/* CONSUME THE EVENTS */`
12. `/* APPLY OTHER REACTIONS TO THIS SCENARIO */`
13. `end`
14. `</RuleBody>`
15. `</Insert>`
16. `</ComplexEventRuleActionList>"
CEP for the adaptive monitoring
interactions during rule synthesis
case study - scenario -
case study – objectives –

» the interaction between Airport and SecurityCompany is regulated by an SLA

» reaction to a SLA violation could depend on the different configurations about the monitored notifications:
   » SLA violation && node overload;
   » SLA violation && node not overloaded
case study – deployment -
<?xml version="1.0" encoding="UTF-8"?>
<ComplexEventRuleActionList xmlns="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule ./ComplexEventRule.xsd">

<Insert RuleType="drools">
  <RuleName>Generic MetaRule</RuleName>
  <RuleBody>
    declare GlimpseBaseEventChoreos
      @role( event )
      @timestamp( timeStamp )
    end

    rule "metaRule_WHENSLAVIOLATION"
      no-loop true
      salience 20
      dialect "java"
      when
        $aEvent : GlimpseBaseEventChoreos(this.isConsumed == false,
        this.isException == false, this.getEventName == "SLA Alert")
        ;
      then
        $aEvent.setConsumed(true);
        update($aEvent);
        ServiceLocatorImpl.GetMachineIP("BSM","a","b",
        RuleTemplateEnum.INFRASTRUCCETIARCHIETURALVIOLATION,
        (String)$aEvent.getEventData(), $aEvent.
        getTimestamp());
        ServiceLocatorImpl.GetMachineIP("BSM","a","b",
        RuleTemplateEnum.NOEVENTFROMINFRASTRUCTURE,
        (String)$aEvent.getEventData(), $aEvent.
        getTimestamp());
      end
  </RuleBody>
</Insert>
</ComplexEventRuleActionList>
<xml version="1.0" encoding="UTF-8"/>
<ComplexEventRule xmlns="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule ../ComplexEventRule.xsd">
  <Insert RuleType="drools">
    <RuleName>Generic MetaRule</RuleName>
    <RuleBody>
      declare GlimpseBaseEventChoreos
      @role( event )
      @timestamp( time Stamp )
      end

      rule "metaRule_WHENSLAVIOLATION"
      no-loop true
      salience 20
      dialect "java"
      when
      $aEvent : GlimpseBaseEventChoreos(this.isConsumed == false,
      this.isException == false, this.getEventName == "SLA Alert")
      then
        $aEvent.setConsumed(true);
        update($aEvent);

        ServiceLocatorImpl.GetMachineIP("BSM","a","b",
        RuleTemplateEnum.INFRASPACIOVIOLATION,
        (String)$aEvent.getEventData(), $aEvent.
        getTimestamp());

        ServiceLocatorImpl.GetMachineIP("BSM","a","b",
        RuleTemplateEnum.NOEVENTFROMINFRASPACIO,
        (String)$aEvent.getEventData(), $aEvent.
        getTimestamp());
      end
  </RuleBody>
</Insert>
</ComplexEventRuleActionList>
generated rule – SLA && machine overload–

```xml
<ComplexEventRuleActionList
 xmlns="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule" ... >
 <Insert RuleType="drools">
   <RuleName>SLA violation_overload_Autogenerated_SecutiryCompanyService</RuleName>
   <RuleBody>
     import it.cnr.isti.labse.glimpse.event.GlimpseBaseEventChoreos;
     import it.cnr.isti.labse.glimpse.manager.ResponseDispatcher;
     import it.cnr.isti.labse.glimpse.utils.NotifierUtils;
     import it.cnr.isti.labse.glimpse.impl.ServiceLocatorImpl;
     import it.cnr.isti.labse.glimpse.impl.RuleTemplateEnum;

     declare GlimpseBaseEventChoreos
     @role( event )
     @timestamp( timeStamp )
     end

     rule "SecutiryCompanyService_INFRASTRUCTUREVIOLATION"
     no-loop
     salience 10
     dialect "java"
     when
     $aEvent : GlimpseBaseEventChoreos(this.isConsumed == true,
     this.getTimeStamp == 1360752708858,  
     this.getEventName == "SLA Alert");
     $bEvent : GlimpseBaseEventChoreos(this.isConsumed == false,
     this.getEventName == "load_one",
     this.getMachineIP == "172.215.65.132",
     this after[0,10s] $aEvent);
     then
     $bEvent.setConsumed(true);
     update($bEvent);
     ResponseDispatcher.LogViolation("drools.getRule().getName()","
     auto_generated_rule",
     "\nSLA and Infrastructure violation by
service: SecurityCompanyService"
+ "\npossibly due to an overload on
machine: 
+ $bEvent.getMachineIP()";
     retract($aEvent);
     retract($bEvent);
   </RuleBody>
 </Insert>
</ComplexEventRuleActionList>
```
generated rule - SLA && machine overload-

```xml
<ComplexEventRuleActionList
  xmlns="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule" ... >
  <Insert RuleType="drools">
    <RuleName>SLA violation_overload_Auto-generated_SecutiryCompanyService</RuleName>
    <RuleBody>
      import it.cnr.isti.labse.glimpse.event.GlimpseBaseEventChoreos;
      import it.cnr.isti.labse.glimpse.manager.ResponseDispatcher;
      import it.cnr.isti.labse.glimpse.utils.NotifierUtils;
      import it.cnr.isti.labse.glimpse.impl.ServiceLocatorImpl;
      import it.cnr.isti.labse.glimpse.impl.RuleTemplateEnum;

      declare GlimpseBaseEventChoreos
        @role( event )
        @timestamp( timeStamp )
      end
      rule "SecutiryCompanyService_INFRASTRUCTUREVIOLATION"
        no-loop
        salience 10
        dialect "java"
        when
          $aEvent : GlimpseBaseEventChoreos(this.isConsumed == true, this.getHostName == "host1", this.getTimestamp == 1360752708858, this.getResponseCode == 100, this.getEventName == "SLA Alert!");
          $bEvent : GlimpseBaseEventChoreos(this.isConsumed == false, this.getHostName == "host2", this.getTimestamp == 1360752708858, this.getResponseCode == 100, this.getEventName == "load one!");
        then
          $bEvent.setConsumed(true);
          update($bEvent);
          ResponseDispatcher.LogViolation("drools.getRule().getName()", "auto_generated_rule", "SLA and Infrastructure violation by service: SecurityCompanyService" + "impossibly due to an overload on machine: " + $bEvent.getMachineIP());
        end
      </RuleBody>
  </Insert>
</ComplexEventRuleActionList>
```
generated rule – SLA violation @ service level–

```
<ComplexEventRuleActionList xmlns="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule">
    <Insert RuleType="drools">
        <RuleName>SLA_violation_Autogenerated_SecurityCompanyService</RuleName>
        <RuleBody>
            rule "SecurityCompanyService_NOEVENTFROMINFRASTRUCTURE"
            when
            $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] $aEvent));
            then
                ResponseDispatcher.LogViolation("...","auto_generated_rule","\nSLA violation\noccurred on: SecurityCompanyService");
                retract($aEvent);
            end
        </RuleBody>
    </Insert>
</ComplexEventRuleActionList>
```
generated rule – SLA violation @ service level–

```xml
<ComplexEventRuleActionList xmlns="http://labse.isti.cnr.it/glimpse/xml/ComplexEventRule"
  >
  <Insert RuleType="drools">
    <RuleName>SLA_violation_Autogenerated_SecurityCompanyService</RuleName>
    <RuleBody>
      ........
      rule "SecurityCompanyService_NOEVENTFROMINFRASTRUCTURE"
      ........
      when
      $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] $aEvent));
      then
      ResponseDispatcher.LogViolation("...","auto_generated_rule", "\nSLA violation\noccurred
on: SecurityCompanyService");
      retract($aEvent);
      end
    </RuleBody>
  </Insert>
</ComplexEventRuleActionList>
```
the enactment of service choreography over the Cloud poses interesting challenges wrt monitoring:

- the paradigm foresees that the service interaction protocol is on-purpose under specified
- unexpected scenarios may take place at run-time

adaptability is a crucial asset in monitoring SLA violations on the Cloud

- for understanding the causes of run-time anomalies

our proposed adaptive Multi-source Monitoring framework

- can combine and analyze events occurring at different abstraction levels, and
- supports the adaptive generation of the monitoring rules at run-time
Adaptive SLA Monitoring of Service Choreographies Enacted on the Cloud

THANK YOU

Antonia Bertolino, Antonello Calabrò, and Guglielmo De Angelis

CNR-ISTI