Managing Web Service Quality

James Skene, Franco Raimondi and Wolfgang Emmerich
London Software Systems
Dept. of Computer Science
University College London
http://sse.cs.ucl.ac.uk

Setting the scene

“Deutsche Bank AG has agreed to outsource two internal business processes to Accenture Ltd. as part of its ambitious program to cut costs and increase efficiency by moving non-core operations to external service providers. Under the service agreement announced Thursday, Deutsche Bank will outsource its worldwide corporate purchasing and accounts payable services to Accenture. The global consultancy and software development group, located in Hamilton, Bermuda, will provide IT systems and tools to manage the bank’s entire procurement-to-payment process.”

[Source: IDG, 30 Jan 2004]
Web Service Quality

- Current SOA standards mainly focus on functionality
- But organizations depend on quality of services provided by 3rd parties
- Their service needs to be delivered with agreed quality
  - Availability / Timeliness
  - Reliability
  - Confidentiality
  - Integrity, …
Quality Management

- Testing web services alone is insufficient because service quality determined by
  - Resource provision available in the run-time environment
  - Service usage profile
- For web services, we need to
  - have quality norms and standards
  - know how to measure quality
  - have continuous quality monitoring
  - use quality criteria for service selection
- These need to be reified at run-time

Managing quality in WS Architectures

- SLA
- Server Monitor
- Client Monitor
- Purchase Request Client
- Purchase Request Server
- SLA violations
- Component
- Message passing
- Generation
- Feedback Loop
Service Level Agreements

• Associate penalties to aberrant service behaviour
• Are often part of service delivery contracts
• Mitigate risk
• Previously mostly written in natural language
  – Ambiguous
  – Incomplete
  – Inconsistent
• We focus on SLAs in formal languages

Service Level Agreements

• Determine required and provided service quality
• Written in terms of
  – Non-functional requirements
  – Usage constraints
• Often annexed to a service provision contract
• Bi-lateral
• Bi-directional
SLAs determine conditions, e.g.
- Reliability
- Timeliness
- Availability
- Throughput
- Backup

May include terms determining
- Monitorability
- Penalties
- Administration
- Schedules of applicability

SLA Language Engineering

- Aim: defining precise and unambiguous SLAs language
- Use OMG’s Meta Object Facility (MOF) to define
  - Abstract syntax of SLA language
  - Service observation domain model
- Define semantics of SLA language in model denotational style
  - Behavioural constraints between syntax and domain model

Syntax definition for web service SLAs in MOF

SLA() {
    terms = ServiceTerms[terms]() {
        penalties = {
            ::slang::PenaltyDefinition[p1]("Pay client 100 dollars.")
        }
    }

    services = ServiceDefinition[service](Notification port)
    operations = {
        OperationDefinition[o1]("notify") {
        }
        OperationDefinition[o2]("subscribe") {
        }
    }

    failureModes = {
        FailureModeDefinition[f1]() {
            kind = OPERATION;
            operations = {OperationDefinition[o1]}
            maximumLatency = ::types::Duration(5, S)
        }
    }
}

See: http://uclslang.sourceforge.net

SLA in OMG Human readable Textual Notation

SLA() {
    terms = ServiceTerms[terms]() {
        penalties = {
            ::slang::PenaltyDefinition[p1]("Pay client 100 dollars.")
        }
    }

    services = ServiceDefinition[service](Notification port)
    operations = {
        OperationDefinition[o1]("notify") {
        }
        OperationDefinition[o2]("subscribe") {
        }
    }

    failureModes = {
        FailureModeDefinition[f1]() {
            kind = OPERATION;
            operations = {OperationDefinition[o1]}
            maximumLatency = ::types::Duration(5, S)
        }
    }
}

See: http://uclslang.sourceforge.net
SLA in HUTN (cont’d)

```cpp
conditions = ServiceConditions[conditions]() {
    inputThroughput = {
        InputThroughputClause[iTC1]() {
            inputWindow = ::types::Duration(1, min)
            inputConcurrency = 10
            operation = {OperationDefinition[01]}
        }
    }
    reliability = {
        ReliabilityClause[rC1]() {
            failureModes = {FailureModeDefinition[f1]} // When > 5 secs
            reliability = ::types::Percentage(0.9)
            window = ::types::Duration(1, min)
            penalties = {
                UnreliabilityPenaltyClause() {
                    penalty = ::slang::PenaltyDefinition[p1]
                }
            }
        }
    }
}
```

Further SLA syntax: Administration
Service observation domain model

Evidence
+date

ServiceUsageRecord
+duration
+outcome:Outcome

Outcome
+SUCCEEDED:int=1
+FAILED:int=2
+NO_RESPONSE:int=3
+DATAAGED:int=4

ReportRecord
+sent
+received

Report

DefectReport
+defectKind

DefectKind
+PARAMETER:int=1
+OPERATION:int=2
+SERVICE:int=3
+DATA:int=4

Semantics of input-throughput clause

class InputThroughputClause {
  invariant {
    conditions.sLA.admin->forall(
      a : ::services::Administration |
      violationFirstUsage(a.reconciliation.agreed)->forall(
        first : ::services::es::ServiceUsageRecord |
        a.calculation.violation->one(
          v : ::services::Violation |
          v.violator = conditions.sLA.terms.clientDefinition.party
          and v.violatedClause = self
          and v.penalty = penalty
          and v.evidence =
            violationEvidence(a.reconciliation.agreed, first)
        )
      )
    )
  }
}
Reminder

- SLA
- Purchase Request Client
- Server Monitor
- Client Monitor
- Purchase Request Server
- SLA violations

Generating SLA Monitors

- SLAs machine readable
- MOF gives standard representation
- Idea: Generate monitoring component from SLA
- Given service observation data, monitor decides whether actual service level complies with SLA
- Generator written using
  - Java Metadata Interface (Sun)
  - Eclipse Platform
Key idea

• SLAs concern many timeliness constraints:
  – Latency
  – Input and Output Throughput
  – Reliability
  – Availability

• Events can be intercepted and time stamped without changing web service requester and provider

• Monitors can be expressed using timed automata

• Detection of SLA violations reduces to acceptance of timed words that consist of timed events

Timed Automata

• A time sequence is a sequence of real numbers \( \tau = \tau_1 \tau_2 \ldots \tau_n \) such that \( \tau_i > \tau_{i-1} \).

• A timed word is a pair \((w, \tau)\) where \(w\) is a word of length \(n\) and \(\tau\) is a time sequence of length \(n\).

• Timed automata extend finite automata in the following way:
  – They introduce a set of clocks
  – They allow definition of time constraints over transitions
  – They allow to reset clocks.

• Timed automata accept timed words and recognize timed languages.

Expressing Web Service Reliability Constraints

- Negate constraint (i.e. timed automaton accepts timed word that indicates non-reliability)
- In this example, no more than one failure occurrence (fm) per minute.

   0 \rightarrow fm, x := 0 \rightarrow 1 \rightarrow fm, (x < 60)? \rightarrow 2

- Online monitoring per transition is efficient (constant in number of outgoing transitions per state).


On-line monitoring Architecture

Diagram showing the flow from SLA to Timed Automata, then to SLA Monitor, with evidence of SLA Violation, and interactions with Client and Provider via Interceptor.
Performance

Summary

SLA violations

Purchase Request Client

Server Monitor

Client Monitor

Purchase Request Server

accenture
Ongoing Work

- SLAs with Virtualization:
- SLAs and Orchestration:
  - SLAs with service providers
  - A service composition

What SLA can be offered for the composite service?