Flexible Quality of Service (QoS) Management of Web Services Orchestrations

Ajay Kattepur

DistribCom, INRIA, Rennes

PhD Thesis Defense

November 8, 2012
Outline

1. Overview
   - Web Services Orchestrations
   - QoS and SLAs

2. QoS Modeling
   - Monotonicity
   - QoS Theory
   - Weaving QoS

3. QoS Weaving
   - Causality and QoS Tracking
   - Weaving in Orc

4. QoS Management Framework
   - QoS Management
   - Upgrading bestQoS

5. Conclusion and Perspectives
Outline

1 Overview
   - Web Services Orchestrations
   - QoS and SLAs

2 QoS Modeling
   - Monotonicity
   - QoS Theory
   - Weaving QoS

3 QoS Weaving
   - Causality and QoS Tracking
   - Weaving in Orc

4 QoS Management Framework
   - QoS Management
   - Upgrading bestQoS

5 Conclusion and Perspectives
Web Services

- Web services:
  - Platform/Language agnostic
  - Describe functionality, Invocation mechanism
  - Pass control flow, data to other services

- Benefits:
  - Interoperability, SaaS
  - Run-time discovery and binding

---

Web Service Composition

- Composite web services: Control flow, data dependencies
- Workflow management: Heterogeneous/distributed parties; BPMN
- Web mashups: APIs combining data, business processes

---

Orchestrations and Choreographies

- **Orchestration**: Centralized control flow; Data dependencies
- **Choreographies**: Decentralized control of orchestrations
- **Data-dependent**: Control flow dependent on data returned

---

Specifying Orchestrations

Standards:

- WSDL: Service interface, data types exposed
- UDDI: Service directories; Semantic web discovery; Ontologies; Non-functional properties as types
- SOAP/REST: Protocols for message passing among services

\[\text{References:}\]


Specifying Orchestrations

- **Standards:**
  - **WSDL:** Service interface, data types exposed
  - **UDDI:** Service directories; Semantic web discovery; Ontologies; Non-functional properties as types
  - **SOAP/REST:** Protocols for message passing among services

- **Formal Models:** Statecharts; Process calculi; Workflow nets

- **Languages:** YAWL\(^4\), COWS, Orc; BPEL\(^5\) (industry)

- **Composition Synthesis**\(^5\): OWL-S, automated compositions, business artifacts

---


Orc

- Elegant, powerful concurrent programming language
- Fundamental declaration used in Orc: Site
- Combinators to create Expressions:
  - Parallel combinator $F | G$
  - Sequential combinator $F > x > G$ or $F \gg G$
  - Pruning combinator $F < x < G$ or $F \ll G$
  - Otherwise combinator $F ; G$
- Constructs for timeouts, recursions, semaphores, channels, data structures

---

QoS

- QoS: discovery, selection and substitution of services \(^7\)
- Multiple metrics \(^8\)
  1. Availability
  2. Accessibility
  3. Integrity
  4. Performance
  5. Reliability
  6. Regulatory
  7. Security

---


QoS

- QoS: discovery, selection and substitution of services
- Multiple metrics
  1. Availability
  2. Accessibility
  3. Integrity
  4. Performance
  5. Reliability
  6. Regulatory
  7. Security

- QoS metrics: Multi-dimensional; Partially ordered; Probabilistic
- QoS composition: Analytic/Simulations for end-to-end QoS

---


Service Level Agreements

- SLAs: QoS/Resource management when sub-contracting

- Contractual types:
  - Hard Contracts: Latency $\leq 2000$ ms. in 90% of cases (WSLA standards)
  - Soft Contracts: Compares distributions (percentiles in QML)

---

Service Level Agreements

- SLAs: QoS/Resource management when sub-contracting

- Contractual types:
  - Hard Contracts: Latency $\leq 2000$ ms. in 90% of cases (WSLA standards)
  - Soft Contracts: Compares distributions (percentiles in QML)

- Consequences: Negotiations, Monitoring, Optimized Resource Management

---

Focus

- **QoS Modeling**
  - *Data dependent* orchestrations
  - *Probabilistic* QoS behavior
  - *Monotonicity* and consequences for QoS

- **QoS Weaving**
  - Causality as a tool for QoS tracking
  - QoS Weaving in Orc

- **QoS Management Framework**
  - QoS Management Overview - Probabilistic Contracts, Simulation Techniques, Negotiations, Service Product Lines
  - Upgrade *bestQoS* - Optimization
Outline

1 Overview
   • Web Services Orchestrations
   • QoS and SLAs

2 QoS Modeling
   • Monotonicity
   • QoS Theory
   • Weaving QoS

3 QoS Weaving
   • Causality and QoS Tracking
   • Weaving in Orc

4 QoS Management Framework
   • QoS Management
   • Upgrading bestQoS

5 Conclusion and Perspectives
QoS in Monotonic Orchestrations

- Monotonicity in Data dependent Orchestration design
- Abstract Algebra for QoS composition, Contracts
- Functional specifications in Orc with “weaved” QoS aspects

---

“If any service performs better, then so will the Orchestration”\textsuperscript{11}

Data-dependent orchestrations (local vs. global optimization \textsuperscript{12})

Large orchestrations can overlook conditions for monotonicity


# QoS Literature

<table>
<thead>
<tr>
<th>Paper</th>
<th>QoS Framework</th>
<th>Prob. QoS</th>
<th>Monotonicity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abundo et al. (2011)</td>
<td>MDP QoS Formulation</td>
<td>✓</td>
<td>−</td>
</tr>
<tr>
<td>Cardellini et al. (2010)</td>
<td>LP-based QoS Selection</td>
<td>✓</td>
<td>−</td>
</tr>
<tr>
<td>Cao et al. (2005)</td>
<td>Genetic Algorithms</td>
<td>×</td>
<td>−</td>
</tr>
<tr>
<td>Limam and Boutaba (2010)</td>
<td>QoS Simulations; Reputation</td>
<td>✓</td>
<td>−</td>
</tr>
<tr>
<td>Yu and Bouguettaya (2008)</td>
<td>QoS algebra; Dynamic Programming</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Bistarelli and Santini (2009, 2010)</td>
<td>Semiring Algebra; Analytic Composition</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Menascé et al. (2008)</td>
<td>Optimal QoS service selection</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Calinescu et al. (2011)</td>
<td>Probabilistic temporal logic; QoS-based design and reconfiguration</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zeng et al. (2004, 2008)</td>
<td>Integer programming - global vs. local optimization</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ardagna et al. (2005)</td>
<td>Mixed IP; local vs. global QoS guarantees</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Al rifai &amp; Risse (2009)</td>
<td>MMKP; local vs. global QoS guarantees</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Literature survey:** Data dependency, Probabilistic QoS, Monotonicity
Theoretical Model: OrchNets

- Petri-net based modeling advocated with WFnets \(^{13}\)
- Execution of a WFnet produces a partial order of events/actions: *configuration*

An *OrchNet* is a *safe, colored* Occurrence Net with *read-arcs*
- Token values \(c = (v, q) = (\text{data, QoS value})\)
- OrchNet transitions increment both data and QoS

---

Theoretical Model: OrchNets

- Petri-net based modeling advocated with WFnets \(^{13}\)
- Execution of a WFnet produces a partial order of events/actions: *configuration*

- An *OrchNet* is a *safe, colored* Occurrence Net with *read-arcs*
  - Token values \(c = (v, q) = (\text{data, QoS value})\)
  - OrchNet transitions increment both data and QoS

- Configuration \(\kappa\) produces end-to-end QoS:

\[
Q_\omega(\kappa, N) = \bigvee_{p \in \maxPlaces(\kappa)} q_p(\omega)
\]

- Extended to probabilistic settings using stochastic partial orders

---

QoS Algebra

- Tokens \((v, q) = \text{(data, QoS value)}\)
- Algebra \(Q = (D, \leq, \oplus, \triangleleft)\)
QoS Algebra

- Tokens \((\nu, q) = \text{(data, QoS value)}\)
- Algebra \(\mathbb{Q} = (\mathbb{D}, \leq, \oplus, \triangleleft)\)
- Incrementing QoS: \(q_1 = q_0 \oplus \delta q_1\)
QoS Algebra

- Tokens \((v, q) = (\text{data, QoS value})\)
- Algebra \(Q = (\mathbb{D}, \leq, \oplus, \triangleleft)\)
- Incrementing QoS: \(q_1 = q_0 \oplus \delta q_1\)
- Synchronizing tokens: Supremum associated with partial order \(\leq\): \(q_0'' \lor q_1\)
QoS Algebra

- Tokens \((v, q) = (\text{data}, \text{QoS value})\)
- Algebra \(Q = (\mathbb{D}, \leq, \oplus, \triangleleft)\)
- Incrementing QoS: \(q_1 = q_0 \oplus \delta q_1\)
- Synchronizing tokens: Supremum associated with partial order \(\leq: q''_0 \lor q_1\)
- Competition policy: If 
  \((q'_0 \oplus \delta q'_1) \leq (q'_0 \oplus \delta q''_1)\) implies that \(t'_1\) fires with competition 
  \(q'_1 = (q'_0 \oplus \delta q'_1) \triangleleft (q'_0 \oplus \delta q''_1)\)
QoS Domains

- Rich algebra $\mathcal{Q} = (\mathbb{D}, \leq, \oplus, \triangleleft)$ instantiated for multiple domains

- 
  *Latency* $d$: $(\mathbb{R}_+, \leq, +, \text{minima})$

- 
  *Security level* $s$: ($\{\text{high, low}\}, \leq_s, \lor_s, "\text{best}"$)

- 
  *Cost* $c$: Multisets with ($\mathcal{Q} \mapsto \mathbb{N}, \subseteq, \cup, "\text{best}"$)
QoS Domains

- Rich algebra \( \mathcal{Q} = (\mathbb{D}, \leq, \oplus, \triangleleft) \) instantiated for multiple domains

- Latency \( d \): \( (\mathbb{R}_+, \leq, +, \text{minima}) \)

- Security level \( s \): \( \{\text{high, low}\}, \leq_s, \lor_s, \text{"best"} \) 

- Cost \( c \): Multisets with \( (\mathcal{Q} \mapsto \mathbb{N}, \subseteq, \cup, \text{"best"}) \)

- Composite Example: Lexicographic or weighted priority
  \( (s, d) \triangleleft (s', d') = \) if \( d \leq d' \) and \( s = \text{low} \) then \( (s, d') \) else \( (s, d) \)
Ensuring Monotonicity

- **Data-independent** ⇒ Monotonicity
- **Data-dependent:**
  - Orchestration consists of nested fork-joins
  - Competing threads: “best” winning the race policy

A) Service aggregation: \( \delta q'_{12} = \delta q'_1 \oplus \delta q'_2 \) and \( \delta q''_{12} = \delta q''_1 \oplus \delta q''_2 \);

B) Pessimistic QoS evaluation: \( \delta q_{12} = \delta q'_{12} \lor \delta q''_{12} \).
Weaving QoS

Response Time

```xml
<sequence>
  <flow>
    <invoke name = "site(-)"/>
    <sequence>
      <invoke "clock()"/>
      <receive "clock()"
        outputVariable = "clock"/>
      <assign>
        <$site.clock.store = $clock />
        <$orch.RTime = $orch.RTime + $site.RTime />
      </assign>
    </sequence>
  </flow>
  ...
  <flow>
    <receive name = "site(-)" />
    <sequence>
      <invoke "clock()"/>
      <receive "clock()"
        outputVariable = "clock"/>
      <assign>
        <$site.RTime = $clock - $site.clock.store />
        <$orch.RTime = $orch.RTime + $site.RTime />
      </assign>
    </sequence>
  </flow>
</sequence>
```
Role of Causality

- Orchestrations may be specified directly in Orc
- Petri-net modeling unavailable – *Causality* analysis needed
Outline

1 Overview
   - Web Services Orchestrations
   - QoS and SLAs

2 QoS Modeling
   - Monotonicity
   - QoS Theory
   - Weaving QoS

3 QoS Weaving
   - Causality and QoS Tracking
   - Weaving in Orc

4 QoS Management Framework
   - QoS Management
   - Upgrading bestQoS

5 Conclusion and Perspectives
Causality and QoS Tracking

- Causality in distributed systems: Debugging and examining traces
- Equip Orc programs with causal past along with publications
- Leverage causality to track QoS increments

---

Causality

- Orc events: publications, site calls and site returns
- Orc events produce causal dependencies
- A *causality* enabled event is a pair
  \[ e = (v(e), \downarrow e) \]
  - \( v(e) \) is the value of the Orc-event
  - \( \downarrow e \) is, recursively, a finite set of pairs of the causal past of \( e \)
  - \( \downarrow e = \emptyset \) for initial events
Causality - Transformation Rules

- values and variables
  \[
  \begin{align*}
  &[[v]] \rightarrow (v, c) \\
  &[[x]] \rightarrow x > (v, -) > (v, \{x\} \cup c)
  \end{align*}
  \]

- combinators
  \[
  \begin{align*}
  &[[f | g]] \rightarrow [[f]] | [[g]] \\
  &[[f > x > g]] \rightarrow [[f]] > x > [[g]]\{x\} \\
  &[[f < x < g]] \rightarrow [[f]] < x < [[g]] \\
  &[[f ; g]] \rightarrow [[f]] ; track(f) > x > [[g]]\{x\}
  \end{align*}
  \]

- site call
  \[
  \begin{align*}
  &[[v(x_1, \ldots, x_n)]] \rightarrow (x_1, \ldots, x_n) > ((v_1, \_), \ldots, (v_n, \_)) > \\
  &\quad track("v", \bigcup_{1 \leq i \leq n} x_i \cup c) > u > v(v_1, \ldots, v_n) > (v', X) > track((v', X \cup \{u\}))
  \end{align*}
  \]
Implementing Causality

Orc Expression
1

Core Orc
Calculus form (OIL)

<oil>
<constant>
<integer>1</integer>
</constant>
</oil>

Transformation Rules
<call>
<target>
<site>orcCauses.WrapConstant</site>
</target>
...
</call>

Transformed
OIL Output

Orc Compiler

Orc Output
1

Orc Compiler

Orc Output
Enhanced with

Causality
(1, Set((signal, Set())))
Orc with Causality

Orc Expression
1 >x> x

Output
(1, Set((signal, Set())),
(1, Set((signal, Set()))))

def f(x) = x

f(1)

(1, Set((1, Set((signal, Set()))))

((2 >> x) << (1 >> 3)) >> 4 | 5

(5, Set((signal, Set()))
(4, Set((3, Set((signal, Set())),
(3, Set((1, Set((signal, Set()))))))))
QoS with Causality

- A QoS-event is a tuple

\[ e = (v(e), q(e); \downarrow e, \#(e)) \]

- \( v(e) \) is an Orc-event (publication)
- \( q(e) \) is its QoS-increment
- \( \downarrow e \) is the set of causes of \( e \)
- \( \#(e) \) is the set of events in conflict with \( e \)

- Cumulated QoS at final publication using \( (\mathbb{D}, \leq, \oplus, \triangleleft) \):

\[
Q(e) = \left( \bigvee_{e' \rightarrow e} Q(e') \right) \oplus_q q(e) \triangleleft (Q(e') \mid e' \in \#(e))
\]
Orc with Causality and QoS

<table>
<thead>
<tr>
<th>Orc Expression</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>(2 \gg x) \ll x \ll (1 \gg 3)) \gg 4</td>
<td>5</td>
</tr>
</tbody>
</table>
Q-Orc: Weaving Orc with QoS

QoS EnhancedOrc Output (hiding redundant publications)

Monotonicity Assessment
- Checking for Monotonicity
- Enforcement Policies

Orc QoS weaving

SLA in Orc
ResponseTime().QoS(site)
ResponseTime.QoSOplus(site)
......
Cost().QoSCompete(c1,c2)

SLA in Scala
traitResponseTime{
def QoS
  def QoSOplus
  ....
  object QoS extends ResponseTime
}

Orc Intermediary Language (OIL) XML

Orc QoS weaving

Orc Program
Functional Specification

QoS "Weaved"OIL XML (verbose)

Orc Compiler

QoS EnhancedOrc Output (hiding redundant publications)

QoS "Weaved"OIL XML (concise)

QoS OIL XML weaving

Orc Compiler

http://qorc.googlecode.com/svn/trunk/QoS_weaving/
**Q-Orc: Weaving Orc with QoS**

Example: Functional declaration before QoS

```orc
-- SiteDefs.orc

def class f(x) =
    def function() = x
    stop

def class g(x) =
    def function() = x*x
    stop

def class h(x) =
    def function() = x+x
    stop

-- Goal Expression

1 >> 2 >> ((m) << (f(7).function() | (f(9).function())))
| g(12).function() >v1> g(13).function() >v2> (v2)
| h(25).function()

-- Output

50
7
169
```
Q-Orc: Weaving Orc with QoS

- **SLA / QoS Declaration**: types, domains, operations, units
- Library of QoS algebra: instantiated and re-used

```python
-- Types and Definitions for Latency
type ResponseTime = Number
def class ResponseTime(unit) =
    def QoS(String) :: Number
def QoS(sitex) =
    signal >> LatencyIncrement(sitex) >>(_,q)>
        (Ift(unit = Millisecond) >> q | Ift(unit = Second) >> q/1000)

    def QoSPlus(Number,Number) :: Number
def QoSPlus(rt1,rt2) = rt1+rt2

def QoSCompare(Number,Number) :: Number
def QoSCompare(rt1,rt2) = rt1 <= rt2

def QoSCompete(Number,Number) :: Number
def QoSCompete(rt1,rt2) = bestQoS(QoSCompare,[rt1,rt2])

def QoSVee(Number,Number) :: Number
def QoSVee(rt1,rt2) = max(rt1,rt2)
stop
```
**Q-Orc: Weaving Orc with QoS**

- **QoS Registry**: QoS classes, metric units and specific handles
- **Handles**: cost, latency, security increments
- **Validating Registry Entries**: Permissive / Strict

```ocaml
val QoSRegistry =
  [ { . name = "f", QoSDom = ResponseTime, QoSUnit = Millisecond, Handle = LatencyIncrement },
    { . name = "f", QoSDom = Cost, QoSUnit = CurrencyDollars, Handle = CostValue },
    { . name = "h", QoSDom = ResponseTime, QoSUnit = Second, Handle = LatencyIncrement },
    { . name = "h", QoSDom = InterQueryTime, QoSUnit = Second, Handle = [] },
    { . name = "h", QoSDom = Cost, QoSUnit = CurrencyDollars, Handle = CostValue },
    { . name = "h", QoSDom = SecurityLevel, QoSUnit = Level, Handle = SecurityValue }
  ]
```

```ocaml
def QoSMatch(siteID) = each(QoSRegistry) >> Ift(M.name = siteID) >>
  (M.QoSDom,M.QoSUnit,M.Handle)

def QoSValidate(callersiteID, calleesiteID) =
  (collect(defer(QoSMatch,callersiteID)),
   collect(defer(QoSMatch, calleeSiteID))) >> (A,B) >>
   (Ift(A.QoSDom = B.QoSDom) >> signal |
    Iff(A.QoSDom = B.QoSDom) >> Println("Registry Entries Missing") >> stop)
```
**Q-Orc: Weaving Orc with QoS**

- **QoS Weaving**: Generates the tuple of `(Data, QoS)`
- Domains and handles are strictly checked

```ocaml
-- QoSWeaver.inc

def QoSWeaver(site, (lookup, unit, handle)) =
  def ResponseTimeCheck(competition) =
    Ift (lookup = ResponseTime && handle = LatencyIncrement) >>
    (Ambient (ResponseTime, competition) >> (ResponseTime (unit).QoS (site))) ; stop

  def CostCheck() =
    Ift (lookup = Cost && handle = CostValue) >>
    (NonAmbient (Cost) >> Cost (unit).QoS (site, CostValue())); stop

signal >> v<v<(ResponseTimeCheck (max) | CostCheck())

def QoS (site, identifier) =
  val Data = Ref()
  def QoSCollect (v) = collect (defer2 (QoSWeaver, Data?, v))

site >d> Data := d >> collect (defer (QoSMatch, identifier)) >v> (Data?, map (QoSCollect, v))
```
Q-Orc: Weaving Orc with QoS

- QoS weaving equips sites with QoS increments
- Functional declarations can make use of the QoS values

```
-- SiteDefs.orc
def class f(x) =
    def function() = x
    def QoSID() = "f"
    stop

def class g(x) =
    def function() = x*x
    def QoSID() = "g"
    stop

def class h(x) =
    def function() = x+x
    def QoSID() = "h"
    stop

def QoSsite(sitex) = QoS(sitex.function(),sitex.QoSID())

-- Goal Expression
1 >> 2 >> ((m,n) <(m,n)< (QoSsite(f(7)) | (QoSsite(f(9))))))
| QoSsite(g(12)) >(_,q1)> QoSsite(g(13)) >(_v,q2)> (v,append(q1,q2))
| QoSsite(h(25))
```
Q-Orc: Weaving Orc with QoS

- **Functional + QoS**

  Output:

  $$
  (50, \ [[3], \ [107], \ [[0, \ 8, \ 7]], \ ["High"]])
  (7, \ [[7], \ [[2, \ 3, \ 5]]])
  (169, \ [[2], \ [[7, \ 9, \ 1]], \ [1], \ [[0, \ 8, \ 5]]])
  $$
Q-Orc: Weaving Orc with QoS

--TravelAgent definition
def TravelAgent(SalesOrder, Budget, ResponseTime, Cost) =

def AirlineCompany(GenerateInvoice, Cost) = (bestQ(compareCost, defer(inquireCost, AirlineList))
  >q> GenerateInvoice.AirQuote := q >>
  Cost().QoS(AirlineCompany, [q]), Rclock().time() > (q, d)>
  (q, ResponseTime().QoS(AirlineCompany, d))

def HotelBooking(GenerateInvoice, Cost) = (bestQ(compareCategory, defer(inquireCategory, HotelList))
  >q> GenerateInvoice.HotelQuote := q >>
  Cost().QoS(AirlineCompany, [q]), Rclock().time() > (q, d)>
  (q, ResponseTime().QoS(HotelBooking, d))

SubmitOrder(SalesOrder, Budget) > ((GenerateInvoice, Budget), RT)>
(AirlineCompany(GenerateInvoice,
  Cost) > (q, RT1) > (q, ResponseTime().QoSplus(RT, RT1)) > (q, RT)>
HotelBooking(GenerateInvoice,
  Cost) > (q, RT2) > (q, ResponseTime().QoSplus(RT, RT2)) > (q, RT)>
) > (GenerateInvoice, RT) > (GenerateInvoice, RT)

Output:

[[[3, 1, 8]]]
[[[4, 0, 5]]]
(1, [[[33], [[6, 0, 2]]]])
(1, [[[33], [[6, 0, 2]], [7, 1, 13]]])
Outline

1 Overview
   - Web Services Orchestrations
   - QoS and SLAs

2 QoS Modeling
   - Monotonicity
   - QoS Theory
   - Weaving QoS

3 QoS Weaving
   - Causality and QoS Tracking
   - Weaving in Orc

4 QoS Management Framework
   - QoS Management
   - Upgrading bestQoS

5 Conclusion and Perspectives
**QoS Management Overview**

- **Implement bestQoS**
- **Probabilistic Contract Composition**
- **Contract Negotiation**
- **Service Product Lines - QoS**
Sampling Probabilistic QoS Contracts

- Probabilistic Contracts:
  \[ G_Q \preceq F_Q \iff \forall x \in \mathbb{D}_Q, \; G_Q(x) \geq F_Q(x) \]

- Importance Sampling of heavy-tailed distributions
  \[ P_{IS} = \frac{1}{N} \sum_{i=1}^{N} H(Q_i) 1_{H(Q_i) > \Phi} \frac{F_Q(Q_i)}{G_Q(Q_i)} \]

- Precise QoS sampling, measurement and variance in SLAs

---

**Contract Negotiation**

- Strategic negotiation selection for end-to-end QoS improvement
- Integer programming formulation: best in monotonic cases

\[
\min \sum_{j=1}^{N} s_j c_j
\]

Subject to: \( F'_j(\delta)s_j \leq F_j(\delta)s_j \)

- Improved performance when compared to random selection

---

**QoS Modeling**

- QoS Aware Monotonic Orchestration Design
  - QoS Weaving

**QoS Management**

- Implement bestQoS
  - Late Service Binding
    - QoS Dependent Optimization

**Probabilistic QoS Contract Composition**

- Contract Negotiation
  - Product Lines of Services
  - Contract Monitoring
  - Reconfiguration

---

Variability in QoS of Service Product Lines: Dynamic behavior, Probabilistic QoS
Sampling through Combinatorial testing techniques: Pairwise Analysis
Efficient performance when compared to random, exhaustive search

---

Optimization Tools for Orchestrations \(^{19}\)

- Mathematical packages: Optimization on QoS (random variables)
- Generic techniques for total ordering QoS metrics
- Optimization libraries invoked from \textit{Orc}: Resource Management

\[\text{QoS Management Framework} \quad \text{Upgrading \textit{bestQoS}}\]

\[\begin{align*}
\text{QoS Modeling} \rightarrow \\
\begin{array}{c}
\text{QoS Aware Monotonic Orchestration Design} \\
\text{QoS Weaving}
\end{array}
\end{align*}\]

\[\begin{align*}
\text{QoS Management} \rightarrow \\
\begin{array}{c}
\text{Implement \textit{bestQoS}} \\
\text{Probabilistic QoS Contract Composition} \\
\text{Contract Negotiation} \\
\text{Product Lines of Services} \\
\text{Contract Monitoring} \\
\text{Reconfiguration}
\end{array}
\end{align*}\]

\[\begin{align*}
\text{Late Service Binding} \\
\text{QoS Dependent Optimization}
\end{align*}\]

Upgrading Orc - best QoS operator

- Traditional *pruning* operator: \( f < x < (E_1 \mid E_2 \mid \ldots \mid E_n) \) resolves conflict using Latency

- “Best” QoS: \( f < x < q (E_1 \mid E_2 \mid \ldots \mid E_n) \) resolves conflict using arbitrary QoS metric \( q \)

```scala
-- SLADeclaration.orc
def bestQoS(comparer, publisher) = head(sortBy(comparer, publisher))
def class ResponseTime() =
def QoS(sitex, d) = RTime() - d > q > q
def QoSPlus(rt1, rt2) = rt1 + rt2
def QoSCompare(rt1, rt2) = rt1 <= rt2
def QoSCompete(rt1, rt2) = bestQoS(QoSCompare, [rt1, rt2])
def QoSVee(rt1, rt2) = max(rt1, rt2)
stop
```
Analytical Hierarchy Process (AHP)\(^\text{20}\)

- Totally ordered cost functions in multi-criterion decisions
- Comparison done using subjective classification: (1 – equal importance, 5 – strong importance, 9 – extreme importance)

Analytical Hierarchy Process (AHP) \(^{20}\)

- Totally ordered cost functions in multi-criterion decisions
- Comparison done using subjective classification: (1 – equal importance, 5 – strong importance, 9 – extreme importance)
- Principal Eigenvector of the *positive reciprocal matrix*: weights

Perron Frobenius Theorem

*For a positive matrix \(W\), the only positive vector \(v\) and only positive constant \(c\) that satisfy \(Wv = cv\):*

- \(v\) is a positive multiple of the principle Eigenvector of \(W\)
- \(c\) is the principal Eigenvalue of \(W\)

- Consistency check: Perturbation of Eigenvalue

---

Upgrading bestQoS

Specialized site Optima to upgrade bestQoS:

```python
def class Optima():
    type Latency = Number
    type Cost = Number

    val Latency = Ref()
    val Cost = Ref()
    val QoS = (Latency, Cost)
    val AHPWeight = (0.3, 0.7)
    val Constraint = ((Latency, "<:"), 0.5), (Cost, "<:"), 0.8))
    val Routine = "bin"
    def Optimization(QoS, AHPWeight, Constraint, Routine) = lpsolve
    stop
```

---

Upgrading bestQoS

Specialized site Optima to upgrade bestQoS:

```python
def class Optima():
    type Latency = Number
    type Cost = Number

    val Latency = Ref()
    val Cost = Ref()
    val QoS = (Latency,Cost)
    val AHPWeight = (0.3,0.7)
    val Constraint = ((Latency, "<:"), 0.5), (Cost, "<:"), 0.8)
    val Routine = "bin"
    def Optimization(QoS, AHPWeight, Constraint, Routine) = lpsolve
    stop
```

Example:

```python
signal >> (Site1(), Site2(), Site3()) ->(s1,s2,s3)->
Optima().Optimization(
    [s1.latency?, s1.cost?], [s2.latency?, s2.cost?], [s3.latency?, s3.cost?]),
    AHPWeight, Constraint, Routine)
```

COIN-OR (COmputational INfrastructure for Operations Research) 21

---

# Outline

1. **Overview**
   - Web Services Orchestrations
   - QoS and SLAs

2. **QoS Modeling**
   - Monotonicity
   - QoS Theory
   - Weaving QoS

3. **QoS Weaving**
   - Causality and QoS Tracking
   - Weaving in Orc

4. **QoS Management Framework**
   - QoS Management
   - Upgrading bestQoS

5. **Conclusion and Perspectives**
Conclusions

- Accurate QoS modeling
  - Probabilistic QoS
  - Data dependency – Monotonicity
  - “Weaving” QoS into functional specs


- Causality in Orc, QoS tracking

Conclusions

- Optimization within orchestrations

- Improved Contractual Agreements

- Negotiation Strategies

- Product Lines of Services
Perspectives

- Implementing the “best” operator in Scala using appropriate data structures
- Feasibility in Industry settings – QoS/SLAs in choreographies, clouds
- Subjective QoS aspects: security, reliability
- Diagnosis / Re-configuration during run-time monitoring
Thank you.

**Orc Syntax**

\[ D \in \text{Definition} \quad := \quad \text{def} \quad y(\bar{x}) = f \]

\[ f, g, h \in \text{Expression} \quad := \quad p \mid p(\bar{p}) \mid ?k \mid \]
\[ f \mid g \mid f > x > g \mid f < x < g \mid f ; g \mid D \mid f \]

\[ v \in \text{Orc Value} \quad := \quad V \mid D \]

\[ w \in \text{Response} \quad := \quad V \mid D \mid \text{stop} \]

\[ p \in \text{Parameter} \quad := \quad V \mid D \mid \text{stop} \mid \tau \]

\[ n \in \text{Non-publication Label} \quad := \quad V_k(\bar{v}) \mid k?w \mid \tau \mid \bot \]

\[ l \in \text{Label} \quad := \quad !v \]

Abstract syntax of the Orc Calculus.
Orc: Structural Operational Semantics

\[ \frac{k \text{ fresh } \bar{v} \text{ closed}}{V(\bar{v}) \quad \forall_k(\bar{v}) \quad ?k} \quad \text{(SiteCall)} \]

\[ ?k \quad \frac{k}{{\to} w} \quad \text{(SiteReturn)} \]

\[ \frac{\bar{v} \text{ closed}}{\bar{v} \quad {!} \bar{v} \quad \text{stop}} \quad \text{(Publish)} \]

\[ D \text{ is } \text{def } y(\ldots) = \ldots \]

\[ \frac{f \xrightarrow{\tau} [D/y] f}{D \quad f \xrightarrow{\tau} [D/y][\bar{p}/\bar{x}] g} \quad \text{(DefDeclare)} \]

\[ D \text{ is } \text{def } y(\bar{x}) = g \]

\[ \frac{D(\bar{p}) \xrightarrow{\tau} [D/y][\bar{p}/\bar{x}] g}{f \in \frac{\perp}{f}} \quad \text{(DefCall)} \]

\[ \frac{f \xrightarrow{\tau} f'}{f \mid g \xrightarrow{\tau} f' \mid g} \quad \text{(Par)} \]

\[ \frac{f \xrightarrow{\tau} f'}{f > x > g \xrightarrow{\tau} f' > x > g} \quad \text{(SeqN)} \]

\[ \frac{f \xrightarrow{\tau} f'}{f > x > g \xrightarrow{n} f' > x > g} \quad \text{(SeqV)} \]

\[ \frac{\frac{f \xrightarrow{n} f'}{f > x > g \xrightarrow{n} f' > x > g}}{\text{OtherN}} \]

\[ \frac{\frac{f \xrightarrow{n} f'}{f > x > g \xrightarrow{n} f' > x > g}}{\text{OtherV}} \]

\[ \frac{f \xrightarrow{n} f'}{f > x > g \xrightarrow{n} f' > x > g} \quad \text{(OtherStop)} \]

\[ \frac{f \xrightarrow{n} f'}{f > x > g \xrightarrow{n} f' > x > g} \quad \text{(PruneLeft)} \]

\[ \frac{f \xrightarrow{\tau} f'}{f < x < g \xrightarrow{n} f' < x < g} \quad \text{(PruneN)} \]

\[ \frac{f \xrightarrow{\tau} f'}{f < x < g \xrightarrow{n} f' < x < g} \quad \text{(PruneV)} \]

\[ \frac{f \xrightarrow{n} f'}{f < x < g \xrightarrow{n} f' < x < g} \quad \text{(OtherStop)} \]
**QoS with Causality**

- QoS Algebra: \( Q = (D_q, \leq_q, \oplus_q, \triangleleft_q) \)
- The \( \triangleleft_q \) operator should track competition
- Ambient vs. non-Ambient metrics:

  **type Ambient:**
  \[
  \begin{align*}
  D_q &= D_{q_1} \times D_{q_2} \\
  \leq_q &= \leq_{q_1} \times \leq_{q_2} \\
  \oplus_q &= \oplus_{q_1} \times \oplus_{q_2} \\
  q \triangleleft_q (q(1), \ldots, q(k)) &= (q_1, \max_{i \in I} q_2(i))
  \end{align*}
  \]

  **type Non-Ambient:**
  \[
  \begin{align*}
  D_q &= D_{q_1} \times D_{q_2} \times \cdots \times D_{q_n} \\
  \leq_q &= \leq_{q_1} \times \leq_{q_2} \times \cdots \times \leq_{q_n} \\
  \oplus_q &= \oplus_{q_1} \times \oplus_{q_2} \times \cdots \times \oplus_{q_n} \\
  \triangleleft_q &= q \triangleleft_q (q(1), \ldots, q(k)) = q
  \end{align*}
  \]
### Probabilistic Comparison \(^{22}\)

- Random variables \(X, X' \in \mathbb{D}\), downward closed ideals \(I \subseteq \mathbb{D}\), distributions \(F, F'\):

\[
X \preceq X' \text{ iff for any ideal } I \text{ of } \mathbb{D} \Rightarrow F(I) \geq F'(I)
\]

---

#### Theorem

*For two distribution functions \(F, F'\), there exists a probability space \(\Omega\), probability \(P\) over \(\Omega\) and two real valued random variables \(Y, Y'\) over \(\Omega\) such that:*

1. \(Y, Y'\) have \(F, F'\) as respective distribution functions
2. \(Y \preceq Y'\) if \(Y(\omega) \leq Y'(\omega)\) holds \(\forall \omega \in \Omega\)

- Reduce stochastic comparison of random variables to ordinary comparison as functions endowed with similar probabilities

---

Web Services’ Negotiation

\[
\begin{align*}
\text{min} & \quad \sum_{j=1}^{N} s_j c_j \\
& \quad (f_{ij}' - f_{wj})s_j \leq M_j z_{iwj} s_j \\
\text{Latency Constraint:} & \quad s_j \sum_{w=1}^{W} p_{wj} z_{iwj} \leq s_j \sum_{k=1}^{i-1} q_{kj} \\
& \quad z_{iwj} \in \{0, 1\}; \quad f_{ij}', f_{wj} \in X \\
\text{Cost Constraint:} & \quad c_j \frac{1}{m} \sum_{i=1}^{m} f_{ij}' = K_j, \text{ where } K_j \text{ are Constants} \\
\text{Selection Constraint:} & \quad \sum_{j=1}^{N} s_j = 1, \quad s_j \in \{0, 1\} \\
& \quad i = 1, \ldots, m; \quad w = 1, \ldots, W; \quad j = 1, \ldots, N
\end{align*}
\]