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

Ajay Kattepur

DistribCom, INRIA, Rennes

PhD Thesis Defense

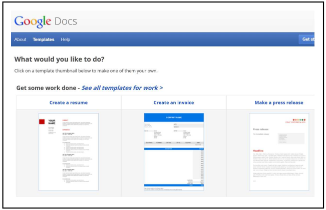
November 8, 2012

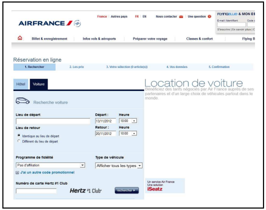
Outline

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

Web Services [1]



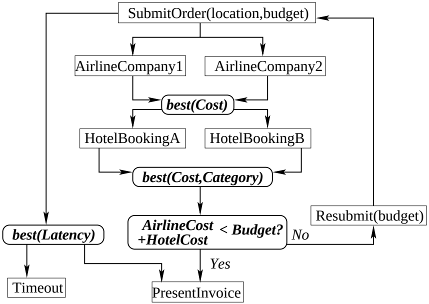


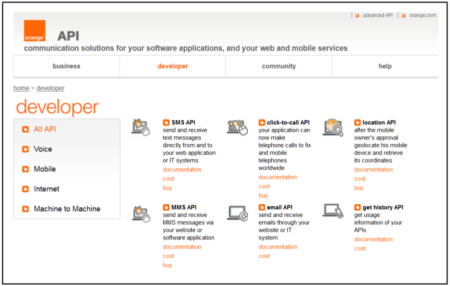
- Web services:
  - Platform/Language agnostic
  - Describe functionality, Invocation mechanism
  - Pass control flow, data to other services
- Benefits:
  - Interoperability, SaaS
  - Run-time discovery and binding

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. Web Services - Concepts, Architectures and Applications. Springer-Verlag, 2004.

Web Service Composition [2]

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

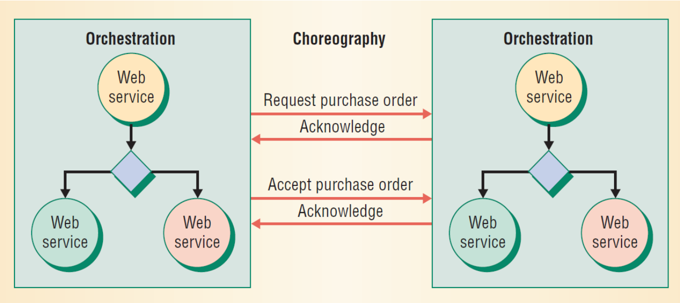




[2] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. Int. J. Web and Grid Services, 1:1-30, 2005.

Orchestrations and Choreographies [3]

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



[3] Chris Peltz, "Web Services Orchestration and Choreography", IEEE Computer, 3, pp. 46-52, 2003.

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

[4] A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell, editors. Modern Business Process Automation: YAWL and its Support Environment. Springer, 2010.

[5] Richard Hull and Jianwen Su. Tools for composite web services: A short overview. SIGMOD Record, 34:1-10, 2005.

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(industry)
- Composition Synthesis [5]: OWL-S, automated compositions, business artifacts

---

[4] A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell, editors. Modern Business Process Automation: YAWL and its Support Environment. Springer, 2010.

[5] Richard Hull and Jianwen Su. Tools for composite web services: A short overview. SIGMOD Record, 34:1–10, 2005.

Orc [6]

- 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

[6] J. Misra and W. R. Cook, "Computation Orchestration: A Basis for Wide-area Computing," *Springer J. of Software and Systems Modeling*, vol. 6, 1, pp. 83 – 110, 2007.

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

[7] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam and H. Chang, "QoS-Aware Middleware for Web Services Composition", IEEE Trans. Software Eng., vol. 30, 5, pp. 311-327, 2004.

[8] Ioan Toma and Douglas Foxvog. Non-functional properties in web services. Technical report, Web Services Modeling Ontology (WSMO) Final Draft, 2006.

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

[7] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam and H. Chang, "QoS-Aware Middleware for Web Services Composition", IEEE Trans. Software Eng., vol. 30, 5, pp. 311-327, 2004.

[8] Ioan Toma and Douglas Foxvog. Non-functional properties in web services. Technical report, Web Services Modeling Ontology (WSMO) Final Draft, 2006.

Service Level Agreements [9]

- 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)

[9] Li-jie Jin, Vijay Machiraju and Akhil Sahai, "Analysis on Service Level Agreement of Web Services", HP Laboratories, 2002.

Service Level Agreements [9]

- 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

[9] Li-jie Jin, Vijay Machiraju and Akhil Sahai, "Analysis on Service Level Agreement of Web Services", HP Laboratories, 2002.

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

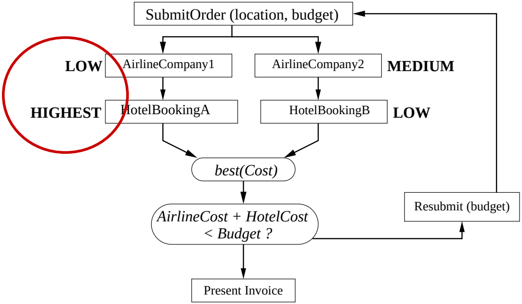
QoS in Monotonic Orchestrations [10]

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

[10] A. Benveniste, C. Jard, A. Kattepur, S. Rosario and J. A. Thywissen, “QoS-Aware Management of Monotonic Service Orchestrations”, *Formal Methods in System Design* (submitted), 2012.

Monotonicity

- "If any service performs better, then so will the Orchestration" [11]
- Data-dependent orchestrations (local vs. global optimization [12])



- Large orchestrations can overlook conditions for monotonicity

[11] Anne Bouillard, Sidney Rosario, Albert Benveniste, and Stefan Haar. Monotonicity in Service Orchestrations. Petri Nets, volume 5606, pages 263-282, 2009.

[12] Danilo Ardagna and Barbara Pernici. Global and Local QoS Guarantee in Web Service Selection. Business Process Management Workshops, volume 3812, pages 32-46, 2005.

QoS Literature

Paper	QoS Framework	Prob. QoS	Monotonicity
Abundo et al. (2011)	MDP QoS Formulation	✓	–
Cardellini et al. (2010)	LP-based QoS Selection	✓	–
Cao et al. (2005)	Genetic Algorithms	×	–
Limam and Boutaba (2010)	QoS Simulations; Reputation	✓	–
Yu and Bouguettaya (2008)	QoS algebra; Dynamic Programming	✓	×
Bistarelli and Santini (2009, 2010)	Semiring Algebra; Analytic Composition	✓	×
Cardoso et al. (2002, 2004)	Generic rules for QoS composition	✓	×
Hwang et al. (2004,2007)	Analytic techniques for QoS composition	✓	×
Menascé et al. (2008)	Optimal QoS service selection	✓	×
Calinescu et al. (2011)	Probabilistic temporal logic; QoS-based design and reconfiguration	✓	×
Zeng et al. (2004, 2008)	Integer programming - global vs. local optimization	✓	✓
Ardagna et al.(2005)	Mixed IP; local vs. global QoS guarantees	×	✓
Alrifai & Risse(2009)	MMKP; local vs. global QoS guarantees	×	✓
Rosario et al. (2007, 2008, 2009)	QoS Algebra, Probabilistic Contracts	✓	✓

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 $c = (v, q) =$ (data, QoS value)
  - OrchNet transitions increment both data and QoS

[13] Van Der Aalst, "The Application of Petri Nets to Workflow Management", 1998.

Theoretical Model: OrchNets

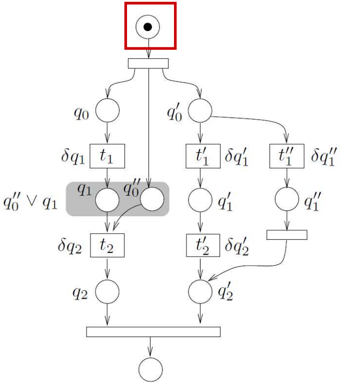
- Petri-net based modeling advocated with WFnets [13]
- Execution of a WFnet produces a partial order of events/actions: *configuration*
- An *OrchNet* is a *safe, colored* Occurrence Net with *read-arcs*
  - Token values $c = (v, q) =$ (data, QoS value)
  - OrchNet transitions increment both data and QoS
- Configuration κ produces end-to-end QoS:

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

- Extended to probabilistic settings using stochastic partial orders

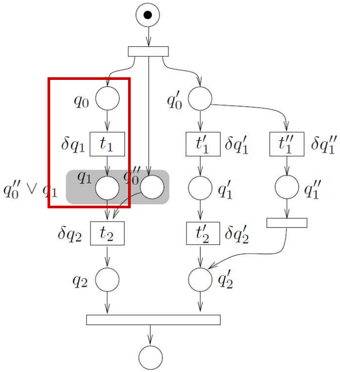
[13] Van Der Aalst, "The Application of Petri Nets to Workflow Management", 1998.

QoS Algebra



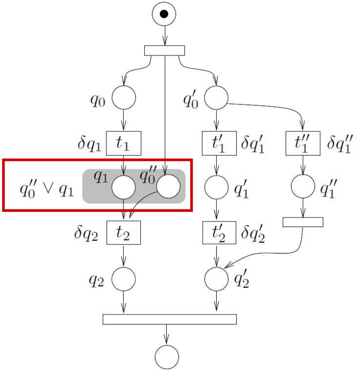
- Tokens $(v, q) = $ (data, QoS value)
- Algebra $\mathbb{Q} = (\mathbb{D}, \leq, \oplus, \lhd)$

QoS Algebra



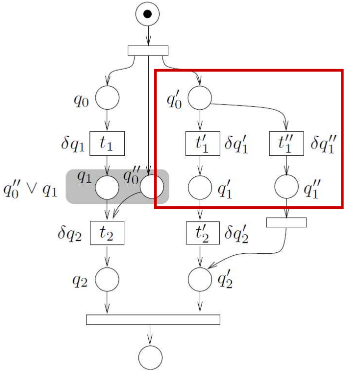
- Tokens (v, q) = (data, QoS value)
- Algebra $\mathbb{Q} = (\mathbb{D}, \leq, \oplus, \triangleleft)$
- Incrementing QoS: $q_1 = q_0 \oplus \delta q_1$

QoS Algebra



- Tokens $(v, q) = (\text{data, QoS value})$
- Algebra $\mathbb{Q} = (\mathbb{D}, \leq, \oplus, \lhd)$
- Incrementing QoS: $q_1 = q_0 \oplus \delta q_1$
- Synchronizing tokens: Supremum associated with partial order $\leq$: $q''_0 \vee q_1$

QoS Algebra



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

QoS Domains

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

- *Latency d*: $(\mathbb{R}_+, \leq, +, \mathit{minima})$
- *Security level s*: $(\{\text{high}, \text{low}\}, \leq_s, \vee_s, \text{"}\mathit{best}\text{"})$
- *Cost c*: Multisets with $(\mathbf{Q} \mapsto \mathbb{N}, \subseteq, \cup, \text{"}\mathit{best}\text{"})$

QoS Domains

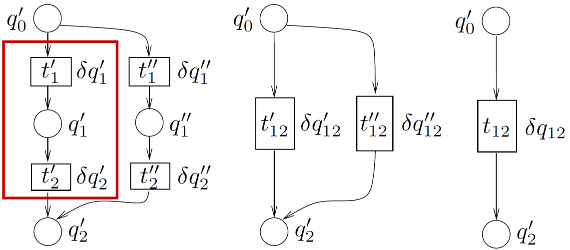
- Rich algebra $\mathbb{Q} = (\mathbb{D}, \leq, \oplus, \triangleleft)$ instantiated for multiple domains

- *Latency d*: $(\mathbb{R}_+, \leq, +, \mathit{minima})$
- *Security level s*: $(\{\text{high}, \text{low}\}, \leq_s, \vee_s, \text{“}\mathit{best}\text{''})$
- *Cost c*: Multisets with $(\mathbf{Q} \mapsto \mathbb{N}, \subseteq, \cup, \text{“}\mathit{best}\text{''})$

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

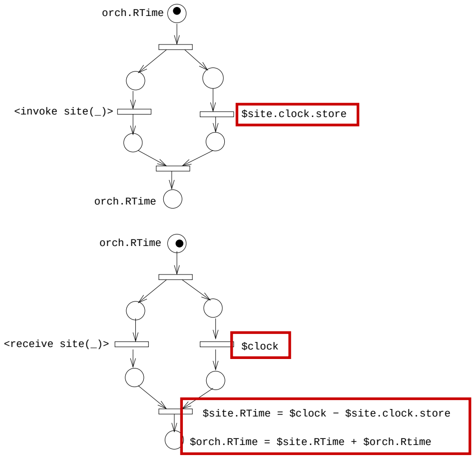
Ensuring Monotonicity

- *Data-independent* $\Rightarrow$ 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: $\delta q_{12} = \delta q'_{12} \vee \delta q''_{12}$.

Weaving QoS



Response Time

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

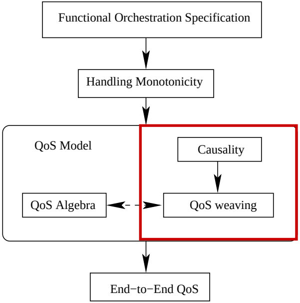
Role of Causality

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

Outline

Causality and QoS Tracking [14]

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



[14] C. Jard, A. Kattepur, J. Thywissen and A. Benveniste, "Leveraging Causality for QoS tracking in Service Oriented Systems", (under preparation), 2012.

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

$$\begin{array}{rcl} [\![v]\!]_c & \rightarrow & (v, c) \\ [\![x]\!]_c & \rightarrow & x >(v, _)> (v, \{x\} \cup c) \end{array}$$

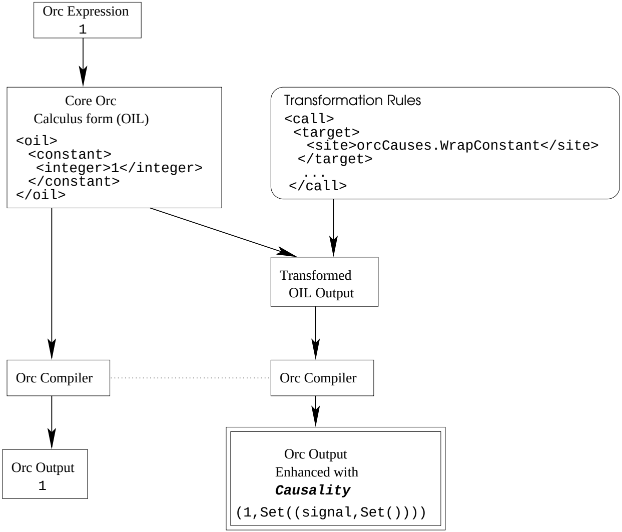
– combinators

$$\begin{array}{rcl} [\![f \mid g]\!]_c & \rightarrow & [\![f]\!]_c \mid [\![g]\!]_c \\ [\![f >x> g]\!]_c & \rightarrow & [\![f]\!]_c >x> [\![g]\!]_{\{x\}} \\ [\![f <x< g]\!]_c & \rightarrow & [\![f]\!]_c <x< [\![g]\!]_c \\ [\![f \; ; \; g]\!]_c & \rightarrow & [\![f]\!]_c \; ; \; track(f) >x> [\![g]\!]_{\{x\}} \end{array}$$

– site call

$$\begin{array}{rcl} [\![v(x_1, \ldots, x_n)]\!]_c & \rightarrow & (x_1, \ldots, x_n) >((v_1, _), \ldots, (v_n, _))> \\ & & track(("v", \bigcup_{1 \leq i \leq n} x_i \cup c)) >u> v(v_1, \ldots, v_n) \\ & & >(v', X)> track((v', X \cup \{u\})) \end{array}$$

Implementing Causality



Orc with Causality

Orc Expression	**Output**
$1 > x > x$	`(1, Set((`**`signal`**`, Set()),` `(1, Set((`**`signal`**`, Set()))))))`
def $f(x) = x$ $f(1)$	`(1, Set((1, Set((`**`signal`**`, Set()))))))`
$((2 \gg x) < x < (1 \gg 3)) \gg 4 \mid 5$	`(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

$$e = (v(e), q(e); \downarrow e, \#(e))$$

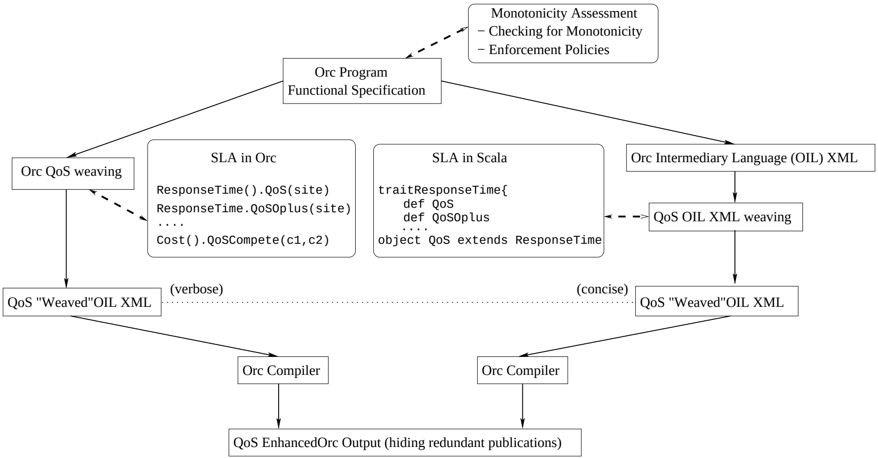
  - $v(e)$ is an Orc-event (publication)
  - $q(e)$ is its QoS-increment
  - $\downarrow e$ is the set of *causes* of e
  - $\#(e)$ is the set of events in *conflict* with e
- Cumulated QoS at final publication using $(\mathbb{D}, \leq, \oplus, \lhd)$:

$$Q(e) \quad = \quad \left(\left(\bigvee_{e' \to e} Q(e')\right) \oplus_q q(e)\right) \lhd \left(Q(e') \mid e' \in \#(e)\right)$$

Orc with Causality and QoS

Orc Expression	**Output**
$1 \mid 2$	(1, 8, Set((**signal**, 0, Set(), Set())), Set((**signal**, 0, Set(), Set()))) (2, 5, Set((**signal**, 0, Set(), Set())), Set((**signal**, 0, Set(), Set())))
$((2 \gg x) <x< (1 \gg 3)) \gg 4 \mid 5$	(5, 1, Set((**signal**, 0, Set(), Set())), Set((**signal**, 0, Set(), Set()))) (4, 7, Set((3, 5, Set((**signal**, 0, Set(), Set()), (3, 5, Set((1, 5, Set((**signal**, 0, Set(), Set())), Set((**signal**, 0, Set(), Set()))))), Set((1, 5, Set((**signal**, 0, Set(), Set())), Set((**signal**, 0, Set(), Set())))))))), Set((**signal**, 0, Set(), Set())))), Set((3, 5, Set((**signal**, 0, Set(), Set()), (3, 5, Set((1, 5, Set((**signal**, 0, Set(), Set())), Set((**signal**, 0, Set(), Set()))))), Set((1, 5, Set((**signal**, 0, Set(), Set())), Set((**signal**, 0, Set(), Set())))))))), Set((**signal**, 0, Set(), Set())))))

Q-Orc: Weaving Orc with QoS [15]



[15] http://qorc.googlecode.com/svn/trunk/QoS_weaving/

Q-Orc: Weaving Orc with QoS

- Example: Functional declaration before QoS

```
-- 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) <(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

```
-- 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 QoSOplus(Number,Number) :: Number
  def QoSOplus(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*

```
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 .}
 ]
```

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

def QoSValidate(callersiteID,caleesiteID) =  (collect(defer(QoSMatch,callersiteID)),
  collect(defer(QoSMatch,caleesiteID))) >(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

```
-- 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().QoSOplus(RT, RT1)  ) >(q, RT  )>
  HotelBooking(GenerateInvoice,
   Cost ) >(q, RT2 )> (q, ResponseTime().QoSOplus(RT, RT2) ) >(q, RT )>
   ) >(GenerateInvoice, RT)> (GenerateInvoice, RT)
```

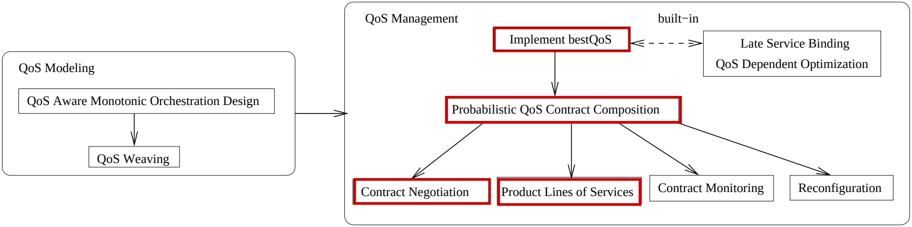
```
Output:

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

Outline

QoS Management Overview

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



Sampling Probabilistic QoS Contracts [16]

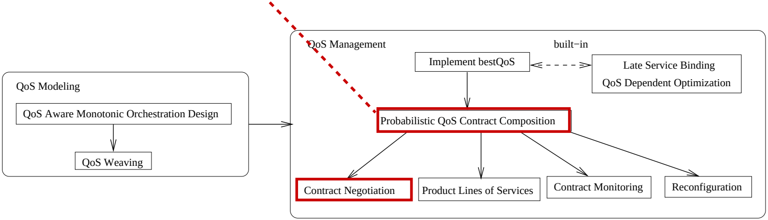
- Probabilistic Contracts:

$$G_Q \preceq F_Q \iff \forall x \in \mathbb{D}_Q, \quad G_Q(x) \geq F_Q(x)$$

- *Importance Sampling* of heavy-tailed distributions

$$\mathbb{P}_{IS} = \frac{1}{N} \sum_{i=1}^{N} H(Q_i) \mathbf{1}_{H(Q_i) > \Phi} \frac{F_Q(Q_i)}{G_Q(Q_i)}$$

- Precise QoS sampling, measurement and variance in SLAs



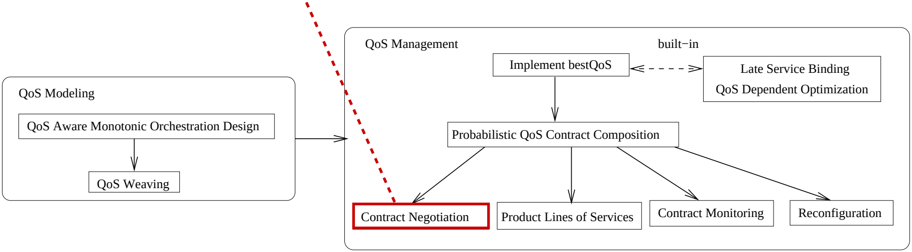
[16] A. Kattepur, "Importance Sampling of Probabilistic Contracts in Web Services", *International Conference on Service Oriented Computing (ICSOC)*, 2011.

Contract Negotiation [17]

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

$$\min \quad \sum_{j=1}^{N} s_j c_j$$
$$\text{Subject to:} \quad F'_j(\delta) s_j \preceq F_j(\delta) s_j$$

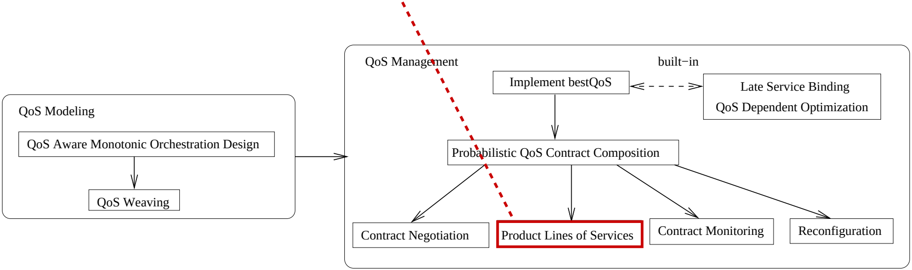
- Improved performance when compared to random selection



[17] A. Kattepur, A. Benveniste and C. Jard, "Negotiation Strategies for Probabilistic Contracts in Web Services Orchestrations", *International Conference on Web Services (ICWS)*, 2012.

Service Product Lines [18]

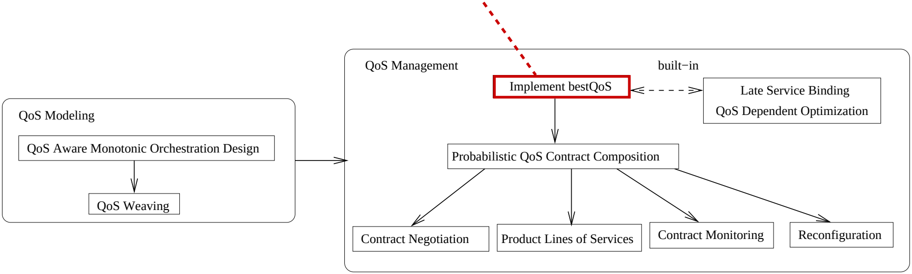
- Variability in QoS of Service Product Lines: Dynamic behavior, Probabilistic QoS
- Sampling through Combinatorial testing techniques: Pairwise Analysis
- Efficient performance when compared to random, exhaustive search



[18] A. Kattepur, S. Sen, B. Baudry, A. Benveniste and C. Jard, "Variability Modeling and QoS Analysis of Web Services Orchestrations", *International Conference on Web Services (ICWS)*, 2010.
A. Kattepur, S. Sen, B. Baudry, A. Benveniste and C. Jard, "Pairwise testing of dynamic composite services", *6th international symposium on Software engineering for adaptive and self-managing systems (SEAMS)*, 2011.

Optimization Tools for Orchestrations [19]

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



[19] A. Kattepur, A. Benveniste and C. Jard, "Optimizing Decisions in Web Services Orchestrations", *International Conference on Service Oriented Computing (ICSOC)*, 2011.

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 : $f <x<_{\mathbf{q}} (E_1 \mid E_2 \mid \ldots \mid E_n)$ resolves conflict using arbitrary QoS metric q

```
-- SLADeclaration.orc
def bestQoS(comparer, publisher) = head(sortBy(comparer, publisher))

def class ResponseTime() =
  def QoS(sitex, d) = RTime()-d >q> q
  def QoSOplus(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)[20]

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

[20] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *Eur. J. of Operational Research*, vol. 48, 1, pp. 9 – 26, 1990.

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 $\mathbf{W}$*, the only positive vector* υ *and only positive constant* c *that satisfy* $\mathbf{W}\upsilon = c\upsilon$*:*
- υ *is a positive multiple of the principle Eigenvector of* $\mathbf{W}$
- c *is the principal Eigenvalue of* $\mathbf{W}$

- Consistency check: Perturbation of Eigenvalue

[20] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *Eur. J. of Operational Research*, vol. 48, 1, pp. 9 – 26, 1990.

Upgrading bestQoS

- Specialized site `Optima` to upgrade `bestQoS`:

```
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
```

[21] R. Fourer, J. Ma, and K. Martin, "Optimization Services: A Framework for Distributed Optimization," *COIN-OR*, 2008.

Upgrading bestQoS

- Specialized site `Optima` to upgrade `bestQoS`:

```
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:

```
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]

[21] R. Fourer, J. Ma, and K. Martin, “Optimization Services: A Framework for Distributed Optimization,” *COIN-OR*, 2008.

Outline

Conclusions

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

A. Benveniste, C. Jard, A. Kattepur, S. Rosario and J. A. Thywissen, "QoS-Aware Management of Monotonic Service Orchestrations", *Formal Methods in System Design (FMSD)* (submitted), 2012.

- Causality in Orc, QoS tracking

C. Jard, A. Kattepur, J. Thywissen and A. Benveniste, "Leveraging Causality for QoS tracking in Service Oriented Systems", (under preparation), 2012.

Conclusions

- Optimization within orchestrations

  A. Kattepur, A. Benveniste and C. Jard, “Optimizing Decisions in Web Services Orchestrations”, *International Conference on Service Oriented Computing (ICSOC)*, 2011.

- Improved Contractual Agreements

  A. Kattepur, “Importance Sampling of Probabilistic Contracts in Web Services”, *International Conference on Service Oriented Computing (ICSOC)*, 2011.

- Negotiation Strategies

  A. Kattepur, A. Benveniste and C. Jard, “Negotiation Strategies for Probabilistic Contracts in Web Services Orchestrations”, *International Conference on Web Services (ICWS)*, 2012.

- Product Lines of Services

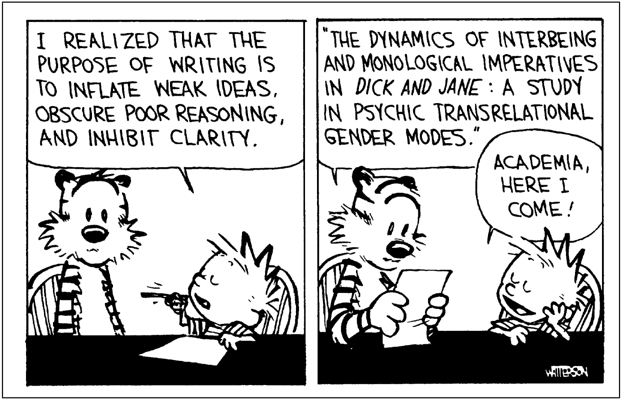
  A. Kattepur, S. Sen, B. Baudry, A. Benveniste and C. Jard, “Variability Modeling and QoS Analysis of Web Services Orchestrations”, *International Conference on Web Services (ICWS)*, 2010.

  A. Kattepur, S. Sen, B. Baudry, A. Benveniste and C. Jard, “Pairwise testing of dynamic composite services”, *6th international symposium on Software engineering for adaptive and self-managing systems (SEAMS)*, 2011.

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.



Bill Watterson, The Calvin and Hobbes Tenth Anniversary Book, 1995.

Orc Syntax

$$
\begin{array}{llll}
D \in \text{Definition} & ::= & \textbf{def}\ \ y(\bar{x}) = f \\
f, g, h \in \text{Expression} & ::= & p \mid p(\bar{p}) \mid ?k \mid \\
 & & f \mid g \mid f >x> g \mid f <x< g \mid f \ \textbf{;}\ g \mid D\ f \\
v \in \text{Orc Value} & ::= & V \mid D \\
w \in \text{Response} & ::= & V \mid D \mid \textbf{stop} \\
p \in \text{Parameter} & ::= & V \mid D \mid \textbf{stop} \mid x \\
n \in \text{Non-publication Label} & ::= & V_k(\bar{v}) \mid k?w \mid \tau \mid \bot \\
l \in \text{Label} & ::= & !v
\end{array}
$$

Abstract syntax of the Orc Calculus.

Orc: Structural Operational Semantics

$$\frac{k \text{ fresh} \quad \bar{v} \text{ closed}}{V(\bar{v}) \xrightarrow{V_k(\bar{v})} ?k} \text{ (SiteCall)}$$

$$?k \xrightarrow{k?w} w \text{ (SiteReturn)}$$

$$\frac{v \text{ closed}}{v \xrightarrow{!v} \textbf{stop}} \text{ (Publish)}$$

$$\frac{D \text{ is } \textbf{def } y(\ldots) = \ldots}{D\ f \xrightarrow{\tau} [D/y]\, f} \text{ (DefDeclare)}$$

$$\frac{D \text{ is } \textbf{def } y(\bar{x}) = g}{D(\bar{p}) \xrightarrow{\tau} [D/y]\,[\bar{p}/\bar{x}]\, g} \text{ (DefCall)}$$

$$\frac{f \xrightarrow{l} f'}{f \mid g \xrightarrow{l} f' \mid g} \text{ (Par)}$$

$$\frac{f \xrightarrow{n} f'}{f > x > g \xrightarrow{n} f' > x > g} \text{ (SeqN)}$$

$$\frac{f \xrightarrow{!v} f'}{f > x > g \xrightarrow{\tau} f' > x > g \mid [v/x]\, g} \text{ (SeqV)}$$

$$\frac{f \xrightarrow{l} f'}{f < x < g \xrightarrow{l} f' < x < g} \text{ (PruneLeft)}$$

$$\frac{g \xrightarrow{n} g'}{f < x < g \xrightarrow{n} f < x < g'} \text{ (PruneN)}$$

$$\frac{g \xrightarrow{!v} g'}{f < x < g \xrightarrow{\tau} [v/x]\, f} \text{ (PruneV)}$$

$$\frac{f \xrightarrow{n} f'}{f \,;\, g \xrightarrow{n} f' \,;\, g} \text{ (OtherN)}$$

$$\frac{f \xrightarrow{!v} f'}{f \,;\, g \xrightarrow{!v} f'} \text{ (OtherV)}$$

$$\frac{f \xrightarrow{\bot} \textbf{stop}}{\textbf{stop} \,;\, g \xrightarrow{\tau} g} \text{ (OtherStop)}$$

QoS with Causality

- QoS Algebra: $\mathbb{Q} = (\mathbb{D}_q, \leq_q, \oplus_q, \lhd_q)$
- The $\lhd_q$ operator should track competition
- Ambient vs. non-Ambient metrics:

$$\text{type } \textit{Ambient}: \begin{cases} \mathbb{D}_q &= \mathbb{D}_{q_1} \times \mathbb{D}_{q_2} \\ \leq_q &= \leq_{q_1} \times \leq_{q_2} \\ \oplus_q &= \oplus_{q_1} \times \oplus_{q_2} \\ q \lhd_q (q(1), \ldots, q(k)) &= (q_1, \max_{i \in I} q_2(i)) \end{cases}$$

$$\text{type } \textit{Non-Ambient}: \begin{cases} \mathbb{D}_q &= \mathbb{D}_{q_1} \times \mathbb{D}_{q_2} \times \cdots \times \mathbb{D}_{q_n} \\ \leq_q &= \leq_{q_1} \times \leq_{q_2} \times \cdots \times \leq_{q_n} \\ \oplus_q &= \oplus_{q_1} \times \oplus_{q_2} \times \cdots \times \oplus_{q_n} \\ \lhd_q &= q \lhd_q (q(1), \ldots, q(k)) = q \end{cases}$$

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 Ω, probability $\mathbf{P}$ over Ω and two real valued random variables Y, Y' over Ω such that:

1. *Y, Y' have F, F' as respective distribution functions*
2. *$Y \leq Y'$ 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

[22] T. Kamae, U. Krengel, and G.L. O'Brien, "Stochastic inequalities on partially ordered spaces", The Annals of Probability, 5(6), pp. 899-912, 1977.

Web Services' Negotiation [23]

$$\min \quad \sum_{j=1}^{N} s_j c_j$$

$$(f'_{ij} - f_{wj}) s_j \leq M_j z_{iwj} s_j$$

Latency Constraint: $$s_j \sum_{w=1}^{W} p_{wj} z_{iwj} \leq s_j \sum_{k=1}^{i-1} q_{kj}$$

$$z_{iwj} \in \{0, 1\}; \;\; f'_{ij}, f_{wj} \in \mathbf{X}$$

Cost Constraint: $$c_j \frac{1}{m} \sum_{i=1}^{m} f'_{ij} = K_j$$, where K_j are Constants

Selection Constraint: $$\sum_{j=1}^{N} s_j = 1, \quad s_j \in \{0, 1\}$$

$$i = 1, \ldots, m; \;\; w = 1, \ldots, W; \;\; j = 1, \ldots, N$$

[23] N. Noyan, G. Rudolf and A. Ruszczynski, "Relaxations of linear programming problems with first order stochastic dominance constraints", *Operations Research Letters*, vol. 34, pp. 653 – 659, 2006.