

Toward Autonomic Web Services Trust and Selection

E. Michael Maximilien
IBM and NCSU
maxim@us.ibm.com

Munindar P. Singh
North Carolina State University
singh@ncsu.edu

New York

July 9, 07

Agenda

- **Service Selection using Trust**
- **Framework**
- **Algorithm Sketch**
- **How to Evaluate?**
- **Empirical Evaluation**
- **Related Work**
- **Contributions**
- **Enhancements since ICSOC submission**
- **Research Directions**

Service Selection using Trust

▪ Background

- *Service Discovery* – consumer finds a desired service interface
- *Service Selection* – consumer selects implementation
- *Service Binding* – consumer begins using the service instance

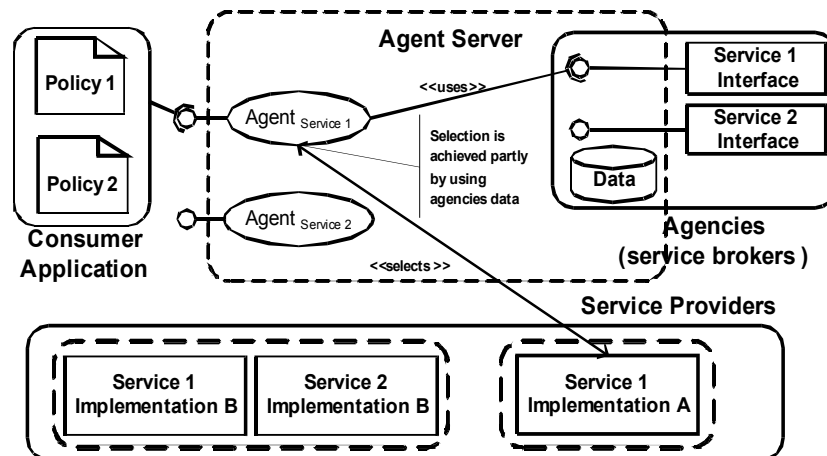
▪ Motivations

- Many implementations; how to select based on nonfunctional attributes
- *Runfiguration* – runtime and configuration are intertwined
- Adapt selection to behaviors of service providers and implementations

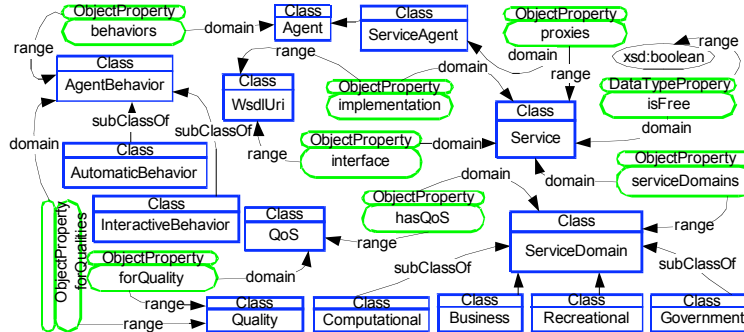
▪ Automated service selection

- Selection presupposes trust in provider and implementation
- Trust based on nonfunctional attributes (QoS) and reputation
 - Consumer preferences (needs) and provider advertisements

Framework – Architecture Overview

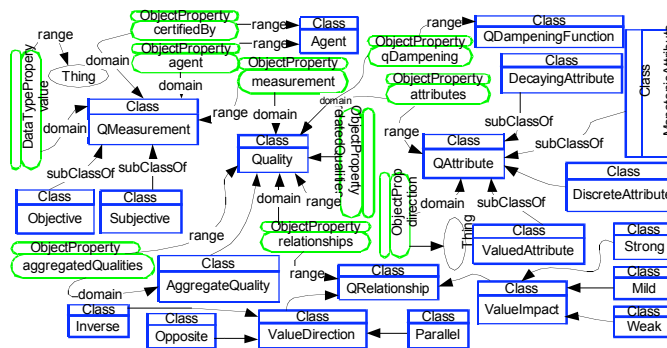


Framework – Service Ontology



- **Service**
 - Interface; 1+ implementations
- **ServiceAgent**
 - Proxies a service
 - Has AgentBehaviors
- **ServiceDomain**
 - Categorizes services
 - Associates indirectly with Quality
- **AgentBehavior**
 - AutomaticBehavior
 - InteractiveBehavior
- **Quality**
 - Nonfunctional attribute of a service in some domain

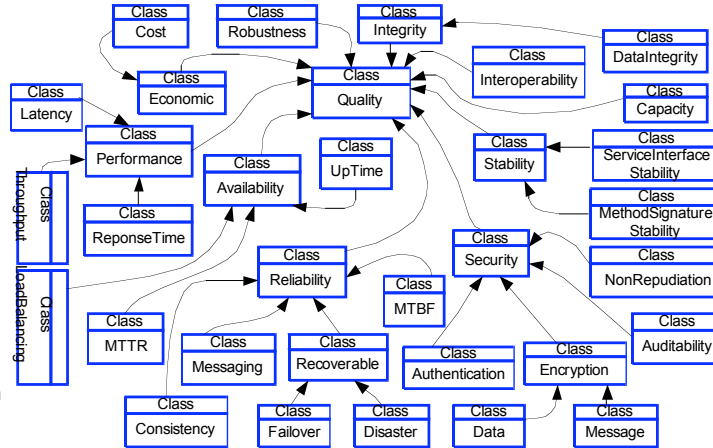
Framework – QoS Upper Ontology



- **QAttribute**
 - Represents a measurable aspect of a quality, i.e., percentage, uptime in days
 - Subclasses indicate the general characteristic and distribution of attribute values
- **QMeasurement**
 - Can be objective or subjective
 - Value determined by an agent
- **QRelationship**
 - Relation between Quality instances

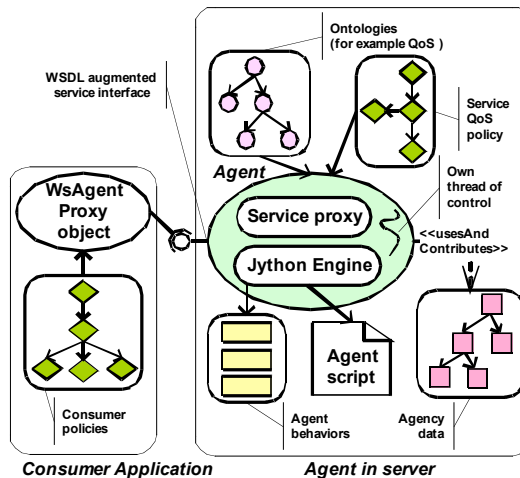
Framework – QoS Middle Ontology

- **Performance**
 - ResponseTime
 - Throughput
- **Reliability**
 - MTBF
 - Recoverability
- **Availability**
 - UpTime
 - TTR
 - LoadBalancing
- **Economic**
 - Cost
- **Security**
 - Authentication
 - NonRepudiation
 - Auditability
- **Others**
 - Stability
 - Interoperability



Framework – Selection Agent

- **Dynamically**
 - Create and deploy a Web service with an augmented agent interface
 - Create a proxy to selected service
- **Policies**
 - Service advertisements
 - Consumer preferences
- **Ontologies: Service and QoS**
- **Service registries**
 - Contain descriptors of service interfaces and implementations
- **Agencies**
 - Consulted for collected data and reputation
- **Script in Jython (Python in Java)**
 - Allows for dynamic behaviors to be loaded from ontology
 - Consumer can specify extensions at runtime



Policies – Consumer Preferences

```

<WsPolicy ... name="Consumer1"
2     type="consumer">
  <Services>
4   <Service name="Service1"
      interface="http://.../s1?wsdl"/>
6   <Service name="Service2"
      interface="..."/>
8 </Services>
  <Ontologies>
10  <Ontology name="QoSOnt"
      uri="http://.../owl/qos.owl"/>
12 </Ontologies>

<QoSPolicy ontology="QoSOnt"
14   methods="*"
      services="Service1, Service2">
16  <QoS name="#ResponseTime">
    <qValue>
18   <preferred>60</preferred>
      <max>100</max>
20   <unit>ms</unit>
    </qValue>
22 </QoS>
  </QoSPolicy>
</WsPolicy>

```

Algorithm Sketch

- **findBestService**
 - Returns first service from findBestServices call or throws exception
- **findBestServices**
 - Filter services matching interface
 - Do policy match
 - Do semantic match
 - Sort matches on degree of match (trust value)
- **iMatch** (*not shown*)
 - Return services that match the interface

#@return the 'best' matching service

```

def findBestService(intf, policy, services):
    matches = findBestServices(intf, policy,
                               services)
    if len(matches) == 0:
        raise NoServiceMatchException()
    return matches[ 0 ]

```

#@return a sorted list of matching services

```

def findBestServices(i, p, s):
    iMatches = iMatch(s, i)
    pMatches = pMatch(iMatches, p)
    spMatches = sMatch(pMatches, p)
    matches = []
    if len(spMatches) != 0:
        matches = sortMatches(spMatches) # sort
        on service.dMatch
    else:
        sMatches = sMatch(iMatches, p)
        matches = sortMatches(sMatches)
    return matches

```

Algorithm Sketch (Cont.)

- **pMatch**
 - Match services with consumer policy
- **pMatchAdvert**
 - Match qualities from provider advertisement and consumer policy (preferences)
- **qualityMatch**
 - Match a provider quality advert and consumer quality preference
- **calculateDegree**
 - Determines trust value

#@return list of matching services

```
def pMatch(services, policy):
    matches = []
    for s in services:
        dMatch = pMatchAdvert(s.advert, policy)
        if dMatch > 0:
            s.dMatch += dMatch; matches.append(s)
    return matches
```

#@return the degree of policy match

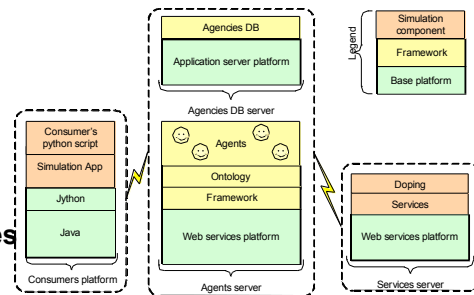
```
def pMatchAdvert(advert, policy):
    dMatch = 0
    for qos in policy.qosList:
        for aQoS in advert.qosList:
            dMatch += qualityMatch(aQoS.quality,
                                   qos.quality):
    return dMatch
```

#@return degree of match between q1 and q2

```
def qualityMatch(q1, q2):
    degree = 0
    if compatible( q1, q2 ):
        dataSet = [q.min, q.max, q1.typical,
                   q2.preferred, reputation(q2)]
        degree = calculateDegree(q2.preferred, dataSet)
    else:
        degree = -100
    return degree
```

How to Evaluate?

- **Flexible simulation component**
 - Configure pools of consumers and providers
 - Specify consumers' execution sequences
 - Sequential or parallel
 - Predetermined or random
 - Multiple iterations and runs of simulations
- **Added means to "dope" services**
 - Modify desired quality characteristics of a service, i.e., make service "bad"
- **Collect simulation data from consumer and agent**



Empirical Evaluation – Trust and Autonomy

Experiment Design

- Service interface is *SortService*
- Three types of providers: Bubble, Merge, and Quick
- Three qualities: Availability, Reliability, and ResponseTime
- Three types of consumers: Mellow, Careful, and Rushed
- Three pools of five consumers each
 - Consumers of each pool biased to one consumer type
- Three pools of five providers
 - All providers are doped except last member of each pool
- Five simulations (10 iterations per) with adjusted doping and agency data

Expected Results

- Without agency data, agents randomly select services
- With doping, emergence of autonomic characteristic
- Delay in doping should also delay convergence

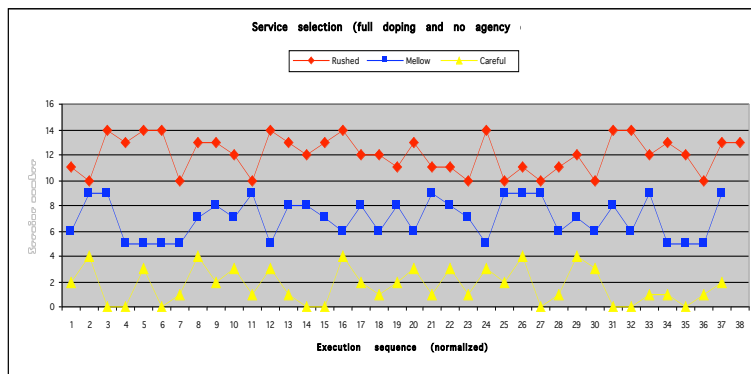
Policy match selects correct providers

Service instances 0, ..., 14

- Bubble = {0, ..., 4} with {4} clean
- Merge = {5, ..., 9} with {9} clean
- Quick = {10, ..., 14} with {14} clean

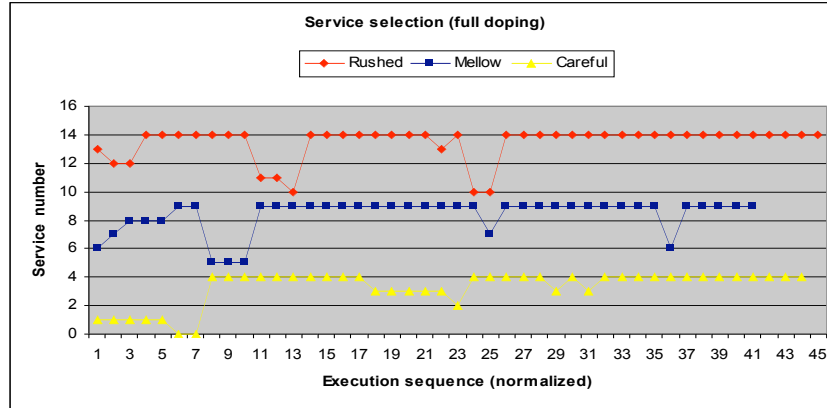
No agency data

- X: time; Y: selected instance
- Agent randomly selects instances; no agency data; no learning



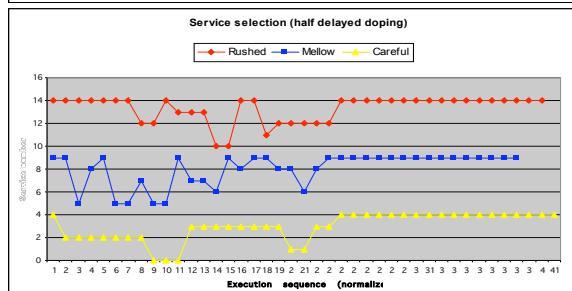
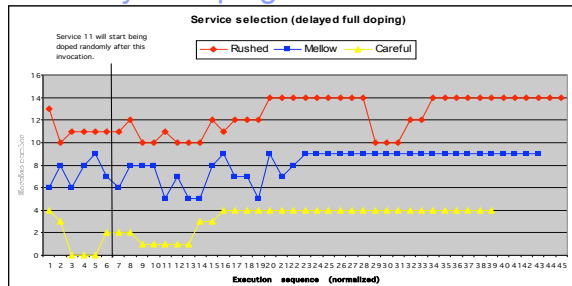
Convergence to clean provider using agency data

- **Dope Availability, Reliability, and ResponseTime of all services but last of each pool**
 - Gradual convergence to sole clean instance of each pool



Delayed convergence with delayed doping

- **Delayed service doping**
 - Delayed convergence of consumer pools to sole clean instance
 - Doping delay is after 4th selection for any service and its occurrence is random
- **Half-service doping**
 - ½ of services of each pool are doped at beginning then other ½ is doped—last service of each pool still remains clean
 - Convergence shifted to right since services of each pool are doped incrementally



Related Work

- **Web services QoS and QoS models**
 - Early and incomplete QoS taxonomy by SRI [Sabata et al., 1997]
 - Mediator service broker from CSIRO Australia [Ran et al., 2003]
 - GlueQoS by IBM Research [Wohlstadter, Tai, Mikalsen, Rouvellou, and Devanbu 2004]
 - UML QoS profile [Aagedal et al., 2004]
- **Web services selection**
 - QoS-aware Web service composition [Zeng and Liu et al., 2004]
 - Dynamic configuration of resource aware services [Poladian, Garlan, and Shaw, 2004]
- **Trust, trustworthiness and Reputation in Web services**
 - Trust management via reputation [Zacharia and Maes, 2002]
 - Reputation an endorsement for Web services [Maximilien and Singh 2001]
 - Verity Reputation System [Ran, 2003]
 - Web Service Level Agreement by IBM Research [Keller, 2002; Ludwig et al., 2003]
- **Semantic matchmaking**
 - Semantic matchmaking of service capabilities using DAML-S [McIlraith et al. 2001; Trastour et al. 2001]
 - OWL-S (née DAML-S) from SRI, CMU and others, 2004
- **Agent for Web services**
 - Service proxy agent architecture [Maximilien and Singh, 2001]
 - WfMS at University of Georgia [Cardoso and Sheth, 2002]
 - Agent lab at CMU [Sycara et al., 2002]
- **Autonomic computing (AC)**
 - AC manifesto [Horn, 2001]
 - Vision of AC [Kephart and Chess, 2002]
 - IBM's Systems Journal special issue on AC in January 2003
 - IBM's AC Toolkit (part of emerging technology toolkit) 2004

Summary of Contributions

- **Software engineering**
 - Agent framework for services (evaluated via selection agent)
 - QoS ontology (evaluated via selection agent and policies)
 - QoS policy
- **Trust and Trustworthiness**
 - Beginnings of a trust model for service selection and binding
 - Service quality reputation
- **Autonomic Computing**
 - Emergence of *self-adjusting trust* in ecosystem of service consumers and providers

Enhancements since ICSOC submission

▪ **Software Engineering**

- Selection algorithm using quality policies, structural properties of quality, and enhanced consumer quality preferences
- Heuristic determination of interaction styles between consumers and providers that yield trustworthiness and self-adjusting trust

▪ **Trust and Trustworthiness**

- Service trust model using quality reputations, quality relationships, and consumer tradeoff preferences for qualities
- Endorsements and provider honesty
- Reputation algorithm for service quality
 - Comparison with others reputation algorithms

▪ **Autonomic Computing**

- Complete emergence of self-adjusting trust in ecosystem of service consumers and providers
 - Adding explorer agents to help build reputation of service

Research Directions

▪ **Software Engineering, Knowledge Engineering, and MAS**

- Discovering service capabilities at runtime, i.e., semantic matchmaking
- Provider agents and automated QoS negotiation
- Standardization of QoS ontologies

▪ **Trust and Trustworthiness**

- Expand policy language to allow for trust negotiation
- Local trust model to complement global trust (reputation)

▪ **Autonomic Computing**

- How to reach higher AC levels: self-configuration of explorer agents?

Thank You

E. Michael Maximilien

maxim@us.ibm.com

<http://maximilien.org>

Munindar P. Singh

singh@ncsu.edu

<http://www.csc.ncsu.edu/faculty/mpsingh/>

Selection using Trust (*Backup*)

- **Automated service selection**

- Selection presupposes trust in provider and implementation
- Trust assignment based on nonfunctional attributes (QoS) and reputation
 - Consumer preferences (needs) and provider advertisements

- **Trust Model Sketch**

- Service, interface, domain, and implementation $s = (\iota, d, i)$ $\iota \in \Psi_s$, $d \in \Delta$, and $i \in I_s$ where $I_s \subseteq \Psi_s$
- Quality reputation $R_Q = \frac{1}{n} \sum_{k=0}^n q_k \delta^{-t(q_k)}$, where $Q = \{q_k\}_{k=0}^n$
- Selection problem $i = \arg \max_{i \in I_s} \{trust(i, \Phi_d)\}$

Framework – Key Phases (Backup)

- **1) Providers setup (1 - 4)**
 - Service providers implementations are added to WSAF registries
 - Each implementation can advertise a QoS policy
- **2) Consumer setup (5 - 6.1)**
 - Specify service interface
 - Specify its QoS and binding policy
- **3) Agent setup (7 - 7.3)**
 - Bootstrap with QoS ontology
 - Loads objective (automatic) and subjective (interactive) behaviors
- **4) Agent selection (8 - 8.3)**
 - Consult agencies for available current QoS and reputation data
 - Runs selection algorithm and selects service implementation
- **5) Consumer service usage (9 - 13)**
 - Use service by invoking same methods on agent
 - Agent monitors all interactions
 - Agent records some QoS data (objective behaviors)
 - Agent interacts with consumers for subjective QoS feedback

