Two important Questions

**Security**
- Web Services are using port 80 – no problems with firewalls
- How to protect against attacks?
- Encryption of exchanged data
- …

**Quality of Service**
- Building an electronic service market with Web Services
- The customer buys a service – he expects a certain service quality
- How can quality aspects be considered in service selection?
- How can quality aspects be guaranteed in service usage?
- Moreover: composition of web services makes the whole thing even more complex…
QoS for Web Services

Lot of approaches for enhancing Web Services with Quality of Service, e.g.:

- WSLA: Web Service Level Agreements (material from IBM)
- WSOL: Web Service Offerings Language (material from Carleton University, Ottawa, Canada)
- UDDIe: Extending UDDI to deal with QoS (material from Cardiff University)
Specifying QoS: WSLA

WSLA: enhancing the service description by defining performance characteristics:
- QoS metrics (mean response time)
- Service Level Objectives (mean response time < 2 s)
- Conditional actions
- Parties and interactions
WSLA Language Elements

Parties
- Action Interfaces
- Sponsors

Service Description
- Service Objects, refer to
  - WSDL
  - BPEL, ...
- SLA Parameters

Metrics
- Measurement Directives
- Functions

Obligations
- SLOs, Action Guarantees

// Service provider, consumer
// Interface descriptions
// e.g. Measurement Service
// Common view of the service
// Specification, link to serv. descr.
// e.g. WSDL service, port,
// binding, operation
// e.g. Response time
// e.g. Transaction counter
// e.g. Sampling interval
// e.g. \((\text{metric1} + \text{metric2}) / 2\)
// What is actually promised
// e.g. Notify management service

Also: some syntactic help, e.g., Metric Macros, Operation Groups, Pricing
Defining SLA Parameters and Metrics

Assignment of Metric to SLA Parameter

Who Communicates with whom? And how?

Define the Metric: How many Values (in %) of a "Utilization" Time Series are over a Threshold of 80%?

Create the Time Series: - probe every 5 Minutes - keep the last 12 Values
Service Level Objectives

SLOs:

**ACMEProvider guarantees the SLO**

- **The SLO is valid for 1 Day**
- **Time Format: RFC 3060**

**Precondition:**
- OverloadPercentage < 30%

**Guarantee:**
- Average Throughput > 1000

**Evaluate when a new SLA Parameter Value is available.**

```xml
<ServiceLevelObjective name="SLO_for_AvgThroughput">
  <Obliged>ACMEProvider</Obliged>
  <Validity>
    <Start>2001-11-30T14:00:00.000-05:00</Start>
    <End>2001-12-31T14:00:00.000-05:00</End>
  </Validity>
  <Expression>
    <Implies>
      <Expression>
        <Predicate xsi:type="Less">
          <SLAParameter>OverloadPercentage</SLAParameter>
          <Value>0.3</Value>
        </Predicate>
      </Expression>
    </Implies>
  </Expression>
  <Expression>
    <Predicate xsi:type="Less">
      <SLAParameter>AvgResponseTime</SLAParameter>
      <Value>1000</Value>
    </Predicate>
  </Expression>
  <Expression>
    <EvaluationEvent>NewValue</EvaluationEvent>
  </Expression>
</ServiceLevelObjective>
```
Contracting Process

- SLA negotiation and provisioning additional steps in the binding process at the outset.
- Can be skipped later in umbrella agreements.
Party Roles and Evaluation Model

Primary Role

Service Customer
  Management
  Client Application

Sponsored Roles

Condition Evaluation Service
  Condition Evaluation
  Measurement
  Measurement Service
  Metrics
  Service Operation

Actions

Primary Role

Service Provider
  Management
  Condition Evaluation
  SLA Parameters
  Metrics
  Service
WSLA Monitoring Deployment

WSLA Contract Management

Deployment

WSLA

Signatory Party (Provider/Consumer)

WSLA Service Deployment Information (SDI)

Service Deployment

Mealurement

Sponsored Party 1

Specific Configuration Information

Service Deployment

Condition Eval.

Sponsored Party 2

Service Deployment

Management

Sponsored Party 3

Chapter 8: Middleware
WSOL: an offering language

Basing of Service classes:
- Discrete variations of service and QoS
- Same functionality, but differ in constraints
- Different prices
- For different consumer Web Services
- With different utilization of the underlying resources

Classes of service can differ in:
- Usage privileges
- Depth and emphasis of the analysis
- Verbosity and formatting of results
- Guaranteed response time
- Rate and priority of notification
- Payment models (pay-per-use, subscription)
- Price
- ...
Web Service Offerings Language

- WSDL does not formally describe important management information
  - classes of service, SLAs, other contracts
  - relationships between classes of service
  - different types of constraint (e.g., QoS)
  - other management statements (prices/penalties, management parties), policies

- XML-based and WSDL-compatible specification of service offerings

- Formal specification of (current status):
  - Functional constraints
  - QoS constraints
  - Simple access rights
  - Price
  - Management responsibility
  - Relationships between service offerings
WSOL Language Constructs

- Service Offering (SO) - class of service
  - constraints (functional, QoS, access rights)
    - expressions (Boolean, arithmetic, …)
  - statements (prices/penalties, management responsibility)
  - reusability constructs (extension, constraint groups, inclusion, instantiation of constraint group templates, …)
- Service Offerings Dynamic Relationships (SODRs) - can change during run-time
WSDL and WSOL: Example

```
buyStock Web Service

buyStock.wsdl

buyStockPortType
buySingleStockOperation
buySingleStockRequest
  symbol
  quantity
buySingleStockResponse
  totalStockBuyingCost
...
buyStockBinding
buyStockPort

buyStock.wsol

ServiceOffering1
  accounting party: provider
  Constraint1: preCondition
    domain: buyStockOperation
    quantity > 0
  Constraint2: QoSConstraint
    domain: buyStockOperation
    ResponseTime < 0.5 s
  Price: 0.5 $
ServiceOffering2 ...
```
WSOL Examples - Service Offering

```xml
<wsol:offering name="buyStockSO1"
service="buyStock:buyStockService" port="buyStockPort">
  ...
  <wsol:managementResponsibility>
    <supplierResponsibility scope="tns:buyStockSO1"/>
    <independentResponsibility
      scope="tns:MaxResponseTime" entity="http://..."/>
  </wsol:managementResponsibility>
  <wsol:relatedSOsList>
    <wsol:relatedSO name="tns:buyStockSO2"
      dimension="tns:MaxResponseTime"/>
  </wsol:relatedSOsList>
</wsol:offering>
```
<wsol:postcondition operation="buyStockOperation">
  <wsol:comparisonExpression>
    <wsol:arithmeticExpression>
      <wsol:variableName vName="buyStockResult"/>
    </wsol:arithmeticExpression>
    <wsol:comparator type="\lt;="/>
    <wsol:arithmeticExpression>
      <wsol:variableName vName="number"/>
      <wsol:arithmeticOperator type="\times"/>
      <wsol:variableName vName="maxPrice"/>
    </wsol:arithmeticExpression>
  </wsol:comparisonExpression>
</wsol:postcondition>
Usage of WSOL

- WSOL enables formal specification of classes of service, constraints, and management statements for Web Services
- Dynamic selection and negotiation of Web Services and classes of service
- Specification of management third parties and the specific accounting party
- Reusability constructs determine static relationships between service offerings
- Dynamic adaptation and management of Web Service compositions
  - Using manipulation of WSOL service offerings
- Special notation for dynamic relationships between service offerings
- Can be used for Web Service management, Web Service composition management, and selection of Web Services and their QoS
- Web Service management: monitoring, measurement, evaluation, accounting, control
  - Simple contracts and SLAs between Web Services
- Distinctive characteristics: lower run-time overhead, reusability constructs, and support for management applications
• An Extension to UDDI v.2

• Support the notion of “Blue Pages”: Information that describes the service known as the service properties. This information allows other to discover web services based upon its properties (such as Quality of Service attributes).

• A new way to describe and discover Services: dynamic meta-data:
  – Each service has one or more properties.
  – Service Properties could describe any thing, i.e. QoS attributes.
  – Discover services based on Service Properties.

• A new way to control Services: dynamic service life period
UDDIe Characteristics

- **Service Leasing**
  - register services with UDDI for a limited time period
- **Query and advertisement of service based on service properties**
  - user defined properties within a “propertyBag”
  - properties can be a set of parameters, methods available, sub-routine calls available etc
- **Extend the find_service method**
  - enable range based and logical (AND/OR) queries
- **Support for content replication within private registries**
- **Can access UDDI and UDDIe in the same way**
UDDIe Architecture

- UDDIe request
  - Servlet
  - SOAP Parser
  - Lease Manager
  - Checks
    - Syntax check
    - Lease check
    - Service name/type check
  - Java Classes
  - Database
  - Response

Chapter 8: Middleware
Service Leasing

- Services published for limited time periods
- Deals with services which change often or missing services
- Finite Lease:
  - Must define exact time periods for which service may be discoverable
  - Maximum duration defined by UDDIe admin.
- Infinite Lease:
  - Service made available as persistent services
  - Based on a ratio of finite/infinite leases -- defined by the UDDIe admin.
  - No guarantees of infinite leases
- By altering (1) lease duration, (2) finite/infinite leases, an admin can alter discovery performance
Service Leasing

- **Future Lease**
  - Allow lease period to start at a future point in time (i.e. service discoverable only after this time period)

- **Immediate Lease**
  - Standard with UDDI - service discoverable immediately on publication

![UDDI lease diagram](image)

- **DD/MM/YYYY hh:mm:ss**
- **Number of times lease renewed**
Lease Manager

- Checks registry at periodic intervals
  - responsible for ensuring leases obtained for duration (or multiples) specified by admin.
  - Services request lease based on this
- Checks services based on service expiry times
- Lease manager is independent of database/registry
PropertyBag

- List of service properties that may be searched

```
<propertyBag>
  <property><propertyName> CPU </propertyName>
    <propertyType> number </propertyType>
    <propertyValue> 800 </propertyValue></property>
  <property><propertyName> Memory </propertyName>
    <propertyType> number </propertyType>
    <propertyValue> 512 </propertyValue></property>
</propertyBag>
```

User Defined -- may use some predefined ontology or metadata format (can be strings or number)

Additional find qualifiers
- exactPropertyMatch (services which have exactly the requested properties)
- exactMatch (also uses keyedReference and categoryBag)
PropertyBag

```
PropertyBag
  UDDI key
  PropertyName
  PropertyType
  PropertyValueNumber
  PropertyValueString

∞  1

BusinessService
  ServiceKey
  BusinessKey
  ServiceName

1

Lease
  UDDI key
  LeaseAppliedOn
  LeaseExpirationDate
  LeaseRenewalTime
  LeaseLastRenewal
  LeaseStartFrom
  isInfinity

Binding Template
CategoryBag
```
PropertyBag

```xml
<find_service businessKey="*****" generic="2.0" xmlns="urn:uddi.org:api_v2">
  <name>*****</name>
  <categoryBag>
    <keyedReference tModelKey="*****" keyName="*****" keyValue="*****" />
  </categoryBag>
  <tModelBag>
    <tModelKey>*****</tModelKey>
  </tModelBag>
  <propertyBag>
    <property>
      <propertyFindQualifier>******</propertyFindQualifier>
      <propertyName>******</propertyName>
      <propertyType>******</propertyType>
      <propertyValue>******</propertyValue>
    </property>
  </propertyBag>
</find_service>
```

Properties can be combined:

- GREATER_THAN
- GREATER_OR_EQUAL
- LESS_THAN_OR_EQUAL
- EQUAL_TO
- NOT_EQUAL_TO
- AND/OR
Logical AND/OR Search

FOR EACH element in the find_service message DO
    Fetch the services which match with the element value
    Add the services’ keys into the element result set
    Add the element result set into total result set
End For Loop

IF Logical OR is required THEN
    final result set = Union all element result set in the total result set
ELSE IF
    final result set = Intersect all element result set in the total result set
END IF

- **element result set**: A set which contains all the service keys which matched the value of the element.
- **total result set**: A set which contains all the element sets
- **final result set**: The final result set which contains the wanted result.
Conclusion

• Lots of problems in implementing distributed systems: synchronisation, coordination, replication, transactions, heterogeneity, …

• Useful concept: Middleware

• CORBA as supporting infrastructure for application as well as application programmers
  ➢ Lots of supporting services
  ➢ But: overloaded, new protocols, non-performant

• Web Services shall help
  ➢ Integrated with WWW
  ➢ Something new or just a new buzzword?
  ➢ Lots of features known from CORBA are missing

There is no ideal middleware – but searching for such a thing, old concepts are repeated frequently with new names.