Flexible Matching and Ranking of Web Service Advertisements

Navid Ahmadi, Walter Binder

University of Lugano, Switzerland
Outline

• Problem statement
  – Current matchmakers’ limitations

• Proposed solution
  – A flexible matchmaker

• Architecture

• Service request

• Matchmaking process

• Conclusion and future work
Problem

• Matchmakers’ limitations
  – Specific service description languages
  – Bound to third-party repositories
  – Hard-coded ranking criteria

• Drawbacks
  – Service adaptation to matchmakers’ languages
  – Static third-party repositories
  – Static ranking algorithm
  – Specific service publisher
Solution

• A Flexible Matchmaker!
  – Supporting current service description languages
  – Openness to emerging service description languages
  – User-defined third-party repositories
  – User-defined service ranking
  – Independence from service publisher
Architecture

Client

Service Request

Service Retriever

Ranker

Integrated Description

Ranking

Description Selector and Integrator (DSI)

Service Descriptions

Property Extraction

Related Documents

Service ID

Third-Party Repositories

Related Documents

Service ID

UDDI

Matching Services

Functional Properties

Providers

Matchmaker
Sample Use Case

• Looking for a restaurant recommendation service
• Functional properties
  – Input: area, type of cuisine, date
  – Output: name, concrete address
• Non-functional properties
  – reputation, availability, cost
• Quality of service description documents
  – Web Service Level Agreements (wsla)
  – WS-Agreement (wsaq)
• Third-party repositories
  – Reputation repository
Service Request

- Repositories
  - UDDI Repositories
  - Third-party repositories
- XPath extractors
- Selectors
- Utility function
Service Request
UDDI Repositories

<Repository BindingAddress= "http://registry.marketplace1.com/uddi/inquiry">
  <find_service xmlns="urn:uddi-org:api_v3">
    <findQualifiers>
      <findQualifier>
        uddi:uddi.org:findQualifier:sql99:like
      </findQualifier>
    </findQualifiers>
    <name>restaurant</name>
    <name>recommend</name>
    <name>cuisine</name>
    ...
  </find_service>
</Repository>
Service Request
Third-Party Repositories

<Repository BindingAddress='http://www.repSite.com/rep'>
  <getReputation>
    <servicekey>
      $servicekey
    </servicekey>
  </getReputation>
</Repository>
Service Request

XPath Extractors

• Locators for requested properties
• Each XPath extractor consists of:
  – XML namespace
    • Distinguish between different service description languages
  – XPath expression
    • The spot to find requested properties in the specified language
XPath Extractor Example

• Extractor for WSLA language
  
  ```xml
  <Namespace>wsla</Namespace>
  <XPathExpression>
    /SLA/Obligations/SLAParameter[node()="$name"]/../Value[$condition]
  </XPathExpression>
  ```

• Under /SLA/Obligations
  
  ```xml
  <Predicate xsi:type="wsla:Less">
    <SLAParameter>Availability</SLAParameter>
    <Value>0.95</Value>
  </Predicate>
  ```

• `$name` and `$condition` will be defined in selectors
Indirect XPath Extractors

• WS-Agreement XPath extractor
  
  \[
  \text{<Namespace>wsag</Namespace>}
  \text{<XPathExpression>}
  \text{///wsag:Variable[@name="$name"]/wsag:location[$condition]}
  \text{</XPathExpression>}
  \]

• Example WS-Agreement document
  
  \[
  \text{<wsag:Variable name="Availability" metric="...">}
  \text{<wsag:Location>}
  \text{///JobDescription/Resources/Availability/Exact}
  \text{</wsag:Location>}
  \text{</wsag:Variable>}
  \]

• Referring to the property value in another namespace
Dereferencing

• Enclosing the XPath expression in parentheses

\((/wsag:Variable[@name="\$name"]/wsag:location)[\$condition]\)

• Indirect address

//JobDescription/Resources/Availability/Exact

• Dereferencing

<Availability>
  <Exact>0.95</Exact>
</Availability>
Selectors

• Selectors determine requested service properties

• Each selector consists of
  – Selector (property) name
  – Assigned variable name
  – Namespace list
  – Selection condition
  – Default value
Selector Example

- Selector element

  <Selector>
    <Name>availability</Name>
    <Variable>A</Variable>
    <Namespaces>
      <namespace>wsaq</namespace>
      <namespace>wsla</namespace>
    </Namespaces>
    <Condition>node()>=0.8</Condition>
    <Default>0.95</Default>
  </Selector>

- Similar to SQL structure

  Select Availability as A from wsaq,wsla where Availability>=0.8
Selection Considerations

• Namespace list
  – Service description transparency to client

• Selection condition
  – Hard requirements
  – Service list filtration
  – Narrowing down the search

• Default value
  – Determining property necessity
Utility Function

• Utility function is a mathematical formula
  – Used for ranking the selected services
  – Variables: properties retrieved by Selectors

• Example
  – Variables
    • “Availability” as A, “Reputation” as R, “Cost” as C
  – Utility function
    • $(A \times R) / C$

• Expressed by a mathematical markup language
  – MathML
Matchmaking Process

Client

XPath Generator

XSLT Generator

Service Request

XPath

//wsag:Variable[@name="Availability"]/wsag:location)[node() >= 0.8]

Service Retriever

Description Documents’ Information

Selection and Integration

Description Document

Sorted Service List

Ranker

Transformation Document

Integrated Description Document
Deployment Strategies

• Matchmaker on UDDI server
  – Fully centralized
• Matchmaker on an external server
  – Possibly replicated service
• Matchmaker on client side
  – Fully distributed
Conclusion

• Summary
  – Problems with current matchmakers
  – A flexible matchmaker
    • Supporting description languages
    • User-defined third-party repositories
    • User-defined ranking
    • Independence from service publisher

• Future Work
  – Interleaving document fetching, selection and ranking
  – Dynamic negotiation for the best SLA