A Framework for Architecture-driven Service Discovery

A. Kozlenkov  V. Fasoulas  F. sanchez  G. Spanoudakis
A. Zisman

Software Engineering Group
Department of Computing
City University
London - UK
Outline

• Motivation

• Framework Overview

• Example

• Query Specification

• Query Execution Engine

• Query Results

• Conclusion and Future Work
Motivation

- Development of *Service Centric Systems (SCS)*: construction of software systems based on the composition of autonomous web-services
- Need to extend software development practices with new processes, methods, and tools to assist the engineering of complex and dependable SCS
- Discovery and composition of services at different stages of the development life-cycle of a system

**Architecture-driven Service Discovery Framework**
Framework Overview

**Definition:** Identification of services that can provide the functionality and satisfy quality properties and constraints of SCS as specified by its design models

**Requirements/Challenges:**
- Extraction of queries from SCS architecture and design models
- Provision of a query language to support combination of prioritised functionality and quality properties
- Efficient matching of queries against service specifications and return of services with varying degrees of match
- Assistance to designers to select discovered services
- Integration of discovered services into an iterative design process (model re-formulation)

*Identified by industrial partners in the EU SeCSE project*
ASD Process

- **Iterative** architecture-driven service discovery process: queries are derived from SCS design models and discovered services are used to amend and re-formulate SCS models

(SyBM) **System Behavioural Model**: describes interactions between operations of SCS (UML sequence diagram)
(SySM) **System Structural Model**: describes the types of the operation parameters and constraints (UML class diagram)
ASD Framework

- **UML 2.0 Integration Module**
  - Extraction of queries expressing the functionality, properties, and constraints from design models
  - Integration of discovered candidate services into the design models

- **Query Execution Engine**
  - Search for services in different service registries based on similarities between queries and services (graph-matching algorithm)
  - Service specifications: *facets* describing different aspects of services - interface description (WSDL), behavioural description (BPEL4WS), semantic description (OWL, WSMO), other (cost, quality)
Example (SyBM)

Global Positioning Service Centric System

- **haptical**: Haptical
- **locationService**: ILocation Service
- **mappingService**: IMapping Service

1. **FindAddress** (address, zipCode, country)
   - <<return>>
   - 2. **FindAddress**: location

3. **DisplayMap** (center, size)
   - <<return>>
   - 4. **DisplayMap**: map

5. **FindPOI** (center, radius, typePOI)
   - <<return>>
   - 6. **FindPOI**: location
Example (SySM)

Global Positioning Service Centric System

<<Interface>>
ILocationService

FindAddress(addrss:String, zipCode:String, country:String):Location
FindPOI(center:Coordinates, radius:Double, typePOI:String):Location

<<Interface>>
IMappingServices

DisplayMap(center:Coordinates, size:Size):Map

Location
address : String
zipCode : String

Coordinates
posX : Double
posY : Double

Size
width : Integer
height : Integer
Query Specification (1/3)

• A query is specified by system designer by selecting messages that should be realised by operations of services to be discovered and specifying constraints

\[
\text{Query} = \text{copy of an interaction from SyBM (I')} + \text{selected messages in I'} + \text{constraints}
\]

• Query and results are specified by *ASD profile* - set of stereotypes for different UML elements found in:
  – Query interaction (messages)
  – Query results (messages, services)
  – SySM referenced by elements of the query interaction (operations, classes) or result parameters
Query Specification

• Stereotypes of interaction messages
  – <<query_message>>: service operations to be discovered
  – <<context_message>>: constraints for the query messages
  – <<bound_message>>: bound to concrete operations that have been discovered in previous iterations

• **Query Parameters**: scalar values that limit the search space and amount of information returned by the execution engine (number of services to be returned)

• **Query Constraints**: provide specific selection criteria (condition about services and operations)
  – Type: hard or soft
  – Body: OCL expression
    e.g.: `self.description.Provider.contains('nameProvider')`
  – Weight: optional weight if constraint is soft [0.0 and 1.0]
ASD Query Package = \( M_Q \cup DC_Q \cup IC_Q \)

\( M_Q \) : set of query and context messages

\( DC_Q \) : classes defining the type of the parameters of context and query messages, classes in the properties of the messages in \( M_Q \), and classes representing the services to be discovered

\( IC_Q \) : classes that are directly and indirectly references by classes in \( DC_Q \) (transitive closure)
Example

Global Positioning Service Centric System

1: FindAddress(address, zipCode, country)
   <<return>>
2: FindAddress: location
3: DisplayMap(center, size)
   <<return>>
4: DisplayMap: map
5: FindPOI(center, radius, typePOI)
   <<return>>
6: FindPOI: location
<<query-message>>

haptical: Haptical
locationService: ILocationService
mappingService: IMappingService
Example
(SySM)

Global Positioning Service Centric System

<<Interface>> ILocationService

FindAddress(address: String, zipCode: String, country: String): Location
FindPOI(center: Coordinates, radius: Double, typePOI: String): Location[*]

<<Interface>> IMappingServices

DisplayMap(center: Coordinates, size: Size): Map

Location
address : String
zipCode : String

Coordinates
posX : Double
posY : Double

Size
width : Integer
height : Integer
Query Execution Engine

• Two-stage process
  
  – *Filtering*: search for services with operations that satisfy the hard constraints of a query
  
  – *Best operation matching*: search through filtered services to identify operations that have the best match with the soft constraints of a query

• Best operation matching: *graph matching algorithm* based on the assignment problem
Best Operation Matching

- **Graph G:**
  - \( V_Q = \text{Oper}(Q) \cup \text{DV}_k \quad V_S = \text{Oper}_S(Q) \)
  - \( E(V_Q, V_S) \)
  - Weighted edges: \( D(F, v^Q_i, v^S_j) \), overall distance between \( v^Q_i \) and \( v^S_j \) (values \([0,1]\))

\[
D(F, v^Q_i, v^S_j) = \sum_{f \in F} w_f d_f(v^Q_i, v^S_j)
\]

if \( v^Q_i \in \text{Oper}(Q), \ v^S_j \in \text{Oper}_S(Q) \)

\[
D(F, v^Q_i, v^S_j) = 1
\]

if \( v_i \in \text{DV}_k \)

\[
D(F, v^Q_i, v^S_j) = \infty
\]

if \( v_i \) should not be mapped onto \( v^S_j \)
Partial Distance Function

\[ d_{\text{f=signature}}(v^Q, v^S) = w_N \cdot d_L(\text{name}(v^Q), \text{name}(v^S)) + \]
\[ w_{\text{IN}} \cdot d_{\text{PS}}(\text{in}(v^Q), \text{in}(v^S)) + w_{\text{OUT}} \cdot d_{\text{PS}}(\text{out}(v^Q), \text{out}(v^S)) \]

\( d_L \): Linguistic distance based on WordNet lexicon
\( d_{\text{PS}} \): Distance between sets of input or output parameters
(best possible morphism between elements in the sets)

\[ d_{\text{PS}}(P1, P2) = \min_{pm} \left( \sum_{(x, y) \in pm} d_P(x, y) \right) \]
\( d_P(x, y) \): distance between two specific parameters
(best matching between the structures of the parameter types)
Best Operation Matching

- Graph G is constructed and D computed…
- Matching of $V_Q$ and $V_S$
  - Select subset $O(V_Q, V_S)$ of $E(V_Q, V_S)$ - total morphism between $V_Q$ and $V_S$ and minimises function $\Sigma_{(v^Q_i, v^S_j) \in O(V^Q, V^S)} D(F, v^Q_i, v^S_j)$
    
    (assignment problem algorithm)
  - Restrict $O(V_Q, V_S)$ - edges whose distance $D(F, v^Q_i, v^S_j)$ does not exceed a threshold value $D_t$. 
Query Results

• Results are specified by *ASD profile*

  ASD Result Package = Ref_I’ U SySM’ U service_packages

• Stereotypes of operations in service_packages
  – **<bound_operation>>**: operation with the best match to a query message or selected by the designer as best candidate
  – **<candidate_operation>>**: other possible results
  – **<service_operation>>**: remaining operations in the service WSDL

• Query messages are replaced by **bound messages**

• Designers can **analyse** the results and **select** candidate operations
Example

Global Positioning Service Centric System

1: FindAddress(address,zipCode,country)
   <<return>>
   2: FindAddress : location

3: DisplayMap(center,size)
   <<return>>
   4: DisplayMap : map

5: FindPOI(center,radius,typePOI)
   <<return>>
   6: FindPOI : location

<<query-message>>
# Example

Service registry with 45 services

**FindAddress**

<table>
<thead>
<tr>
<th>Provider</th>
<th>Service</th>
<th>Operation</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArcWeb</td>
<td>AdressManager</td>
<td>findAddresses()</td>
<td>0.138</td>
</tr>
<tr>
<td>cdyne.com</td>
<td>AddressLookup</td>
<td>AdvancedCheckAddress()</td>
<td>0.145</td>
</tr>
<tr>
<td>cdyne.com</td>
<td>AddressLookup</td>
<td>CheckAddress()</td>
<td>0.150</td>
</tr>
<tr>
<td>cdyne.com</td>
<td>AddressLookup</td>
<td>CheckAddressW2lines()</td>
<td>0.152</td>
</tr>
</tbody>
</table>

**FindPOI**

<table>
<thead>
<tr>
<th>Provider</th>
<th>Service</th>
<th>Operation</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViaMichelin</td>
<td>FindNearbyPOIService</td>
<td>getPoiList()</td>
<td>0.136</td>
</tr>
<tr>
<td>ViaMichelin</td>
<td>FindNearbyPOIService</td>
<td>getPoi()</td>
<td>0.140</td>
</tr>
<tr>
<td>ViaMichelin</td>
<td>FindNearbyPOIService</td>
<td>getCompactPoiList()</td>
<td>0.140</td>
</tr>
<tr>
<td>ArcWeb</td>
<td>PlaceFinderSample</td>
<td>findPlace()</td>
<td>0.154</td>
</tr>
</tbody>
</table>
Example

Global Positioning Service Centric System

1. findAddress(address: String, addressmanagerOptions: String, token: String): AddressmanagerInfo
2. findAddress(address: String, addressmanagerOptions: String, token: String): AddressmanagerInfo
3. DisplayMap(center, size)
4. DisplayMap(map)
5. getPoiList(request: FindNearbyPOIRequest, check: String): FindLocations
6. getPoiList(request: FindNearbyPOIRequest, check: String): FindLocations
Conclusions and Future Work

• Framework for architecture-driven service discovery integrated with UML-based system engineering design
• Service discovery is part of the design process of SCS
• Prototype tool as an Eclipse plug-in
• Experiments has demonstrated average precision of 62% (3 scenarios, 72 queries, 97 services, 1028 operations)

Currently…
• Extending and evaluating the prototype tool to support service behavioural specifications (IJWSR - to appear)
• Conducting large-scale experimentation with industrial partners
• Implementing the tool as web-services