Efficient optimization techniques for automatic composition of Web services

Doctoral Meeting

Pablo Rodríguez Mier
Supervisors: Manuel Lama, Manuel Mucientes

Centro Singular de Investigación en Tecnoloxías da Información
Universidade de Santiago de Compostela

February 5, 2015
# Outline

1. Introduction
2. Problem Formalization
3. Algorithms
“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.”

— W3C Web Services Architecture Working Group

- Simple I/O Web Service Model:

Examples: Payment services (e.g. Paypal WS), Geolocation (e.g. Google Maps), IaaS (e.g. Amazon WS), E-commerce (e.g. Ebay WS), Delivery services (e.g. FedEx WS)...)
Web services are the most common realization of Service Oriented Architectures. Why?

- **Loosely-coupled components**: well defined interfaces and functionality
- **Distributed components**: can be deployed and accessed through the network
- **Interoperability**: built on standardized protocols and technologies
- **Composability**: can be combined to create new functionality by reusing services
- ...

February 5, 2015
A key feature of Web services is that they can be composed to create new services with new functionality by reusing the existing ones:
Web service composition is a highly complex task

- Huge amount of Web services
- Highly dynamic nature of the Web
  - Services are constantly updated, created and destroyed
- Many possible combinations, hard to find the best one

Need for efficient **automatic composition techniques**
Question

Given a input/output description of the composition goal, how can we obtain **optimal compositions** (fast) that satisfy the goal?
State-of-the-art current problems

- Elevated time to compute good compositions
- Poor scalability with the number of services
- Inefficient / sub-optimal compositions
- Lack of support for automatic service discovery
PhD Tasks & Research goals

- Define a model for composing services by connecting their inputs/outputs (semantics)
- Develop efficient algorithms for automatic composition
  - Minimizing the length of the composition
  - Minimizing the number of services in the composition
  - Optimizing non-functional aspects (QoS)
- Define optimizations to improve the scalability
- Integrated framework for automatic composition and discovery
Applications

Automatic Composition: Applications

- E-commerce
- E-business
- Internet of Things
- Smart Cities
Semantic Web Services (SWS)

How can we “match” inputs and outputs of services?

- Semantic annotations of WS enables logic reasoning of services

We define a semantic Web service as a tuple

\[ w = \{ In_w, Out_w \} \]

where:

- \( In_w = \{ i_1, i_2, \ldots, i_n \} \) is the set of required inputs
- \( Out_w = \{ o_1, o_2, \ldots, o_n \} \) is the set of generated outputs
- \( In_w, Out_w \subseteq O \) are concepts from an ontology \( O \)

\[ W_1 \quad \overset{\text{exact match}}{\longrightarrow} \quad W_2 \]

\[ o^{w_1}_1 \quad \overset{\text{exact match}}{\longrightarrow} \quad o^{w_2}_1 \]

\[ i^{w_1}_2 \quad \overset{\text{plugin match}}{\longrightarrow} \quad i^{w_2}_2 \]
Semantic Matching

When can we invoke a service?

Types of match [Paolucci 2002]:

- **Exact** ($\equiv$): $o_{w1} \equiv i_{w2} \iff$ same concepts
- **Plugin** ($\sqsubseteq$): $o_{w1} \sqsubseteq i_{w2} \iff$ $o_{w1}$ subclass of $i_{w2}$
- **Subsume** ($\sqsupseteq$):
  $o_{w1} \sqsupseteq i_{w2} \iff$ $o_{w1}$ superclass of $i_{w2}$
- **Fail** ($\bot$): no match between concepts

Service invokability:

- Given $C_1, C_2 \subseteq O$, we define $\otimes : O \times O \rightarrow O$ such that $C_1 \otimes C_2 = \{c_2 \in C_2 | \text{match}(c_1, c_2), c_1 \in C_1 \}$
- $\text{match}(c_1, c_2)$ is true $\iff c_1 \equiv c_2 \lor c_1 \sqsubseteq c_2$
- $w = \{\text{In}_w, \text{Out}_w\}$ is invokable with a set of concepts $C \subseteq O$ $\iff$ $C \otimes \text{In}_w = \text{In}_w$
How can we model a valid composition for a request?

Given a composition request $r = \{\text{In}_r, \text{Out}_r\}$, a composite service $w_c = \{\text{In}_{w_c}, \text{Out}_{w_c}, P = \{W, \prec\}\}$ satisfies $r$ if:

- $\text{In}_r \otimes \text{In}_{w_c} = \text{In}_{w_c}$ (invokable with the available inputs)
- $\text{Out}_{w_c} \otimes \text{Out}_r = \text{Out}_r$ (returns all the requested outputs)
- Every service $w \in W$ in the composite service is invokable with the preceding output concepts according to a partial order $P$ imposed by the match dependencies relations between inputs/outputs
The partial order of the services in the composition can be seen as a **directed graph**.

There are many topological orderings of the services (**many ways of invoking the composition: sequence, parallel...**)

February 5, 2015
Optimizing length & services

How to generate good compositions?

- Minimize length $\Rightarrow$ maximize parallel execution

- Minimize num. of services $\Rightarrow$ more interpretable & reliable solutions

Finding the optimal composition with the minimum number of services is NP-Hard!
Service minimization is NP-Hard

**Set Cover Problem** \( \leq_P \) **Service Minimization**

Every instance of the SCP can be trivially represented as an instance of the Service Minimization Problem
Genetic Algorithm for Automatic Composition (I)

Context-Free Grammar

- \( V = \{ \text{initialProcess, process, compositeProcess} \} \)
- \( \Sigma = \{ \text{atomicProcess, choice, sequence, split, splitJoin} \} \)
- \( S = \text{initialProcess} \)
- Rules:
  - \( \langle \text{initialProcess} \rangle ::= \langle \text{compositeProcess} \rangle | \langle \text{atomicProcess} \rangle \)
  - \( \langle \text{process} \rangle ::= \langle \text{compositeProcess} \rangle \langle \text{process} \rangle | \langle \text{compositeProcess} \rangle | \langle \text{atomicProcess} \rangle \)
  - \( \langle \text{compositeProcess} \rangle ::= \text{choice} \langle \text{process} \rangle | \text{sequence} \langle \text{process} \rangle | \text{split} \langle \text{process} \rangle | \text{splitJoin} \langle \text{process} \rangle | \langle \text{process} \rangle \)

Evolutionary Approach

- Initial Population
- Selection
- Replacement
- Recombination
- End

Generation of random compositions using the context-free grammar

- Sequence(AtomicProcess(A), Split(AtomicProcess(B), AtomicProcess(C)))
Context-Free Grammar

- \( V = \{ \text{initialProcess}, \text{process}, \text{compositeProcess} \} \)
- \( \Sigma = \{ \text{atomicProcess}, \text{choice}, \text{sequence}, \text{split}, \text{splitJoin} \} \)
- \( S = \text{initialProcess} \)
- Rules:
  - \( \langle \text{initialProcess} \rangle ::= \langle \text{compositeProcess} \rangle | \langle \text{atomicProcess} \rangle \)
  - \( \langle \text{process} \rangle ::= \langle \text{compositeProcess} \rangle \langle \text{process} \rangle | \langle \text{atomicProcess} \rangle \langle \text{process} \rangle | \langle \text{compositeProcess} \rangle | \langle \text{atomicProcess} \rangle \)
  - \( \langle \text{compositeProcess} \rangle ::= \text{choice} \langle \text{process} \rangle | \text{sequence} \langle \text{process} \rangle | \text{split} \langle \text{process} \rangle | \text{splitJoin} \langle \text{process} \rangle | \langle \text{process} \rangle \)

Evolutionary Approach

- Initial Population
- Selection
- Replacement
- Recombination

**k-tournament** based selection of individuals

February 5, 2015
Genetic Algorithm for Automatic Composition (III)

Context-Free Grammar

- \( V = \{ \text{initialProcess, process, compositeProcess} \} \)
- \( \Sigma = \{ \text{atomicProcess, choice, sequence, split, splitJoin} \} \)
- \( S = \text{initialProcess} \)

- Rules:
  - \(< \text{initialProcess}> ::= < \text{compositeProcess}> | \text{atomicProcess}\)
  - \(< \text{process}> ::= < \text{compositeProcess}> < \text{process}> | < \text{atomicProcess}> < \text{process}> | < \text{compositeProcess}> < \text{atomicProcess}> < \text{process}>\)
  - \(< \text{compositeProcess}> ::= \text{choice} < \text{process}> < \text{process}> | \text{sequence} < \text{process}> < \text{process}> | \text{split} < \text{process}> < \text{process}> | \text{splitJoin} < \text{process}> < \text{process}>\)

Evolutionary Approach

Initial Population

Selection

Replacement

Recombination

Crossover + mutations

February 5, 2015
Genetic Algorithm for Automatic Composition (IV)

Context-Free Grammar

- \( V = \{ \text{initialProcess, process, compositeProcess} \} \)
- \( \Sigma = \{ \text{atomicProcess, choice, sequence, split, splitJoin} \} \)
- \( S = \text{initialProcess} \)
- Rules:
  - \( \text{<initialProcess>} ::= \text{<compositeProcess>} | \text{atomicProcess} \)
  - \( \text{<process>} ::= \text{<compositeProcess>} \text{<process>} | \text{atomicProcess} \text{<process>} | \text{<compositeProcess>} | \text{atomicProcess} \)
  - \( \text{<compositeProcess>} ::= \text{choice} \text{<process>} \text{<process>} \text{<process>} \text{<process>} | \text{sequence} \text{<process>} \text{<process>} \text{<process>} \text{<process>} | \text{split} \text{<process>} \text{<process>} \text{<process>} \text{<process>} \)

Fitness evaluation of each individual

\[
\text{fitness} = \omega_1 \cdot \left( \frac{\sum_{i=0}^{n} |O_{\text{obj}}|}{\text{runPath}} + \frac{|I_{\text{root}} \cap I_{\text{obj}}|}{|I_{\text{obj}}|} \right) + \omega_2 \cdot \frac{1}{\text{Length}} + \omega_3 \cdot \frac{1}{\# \text{atomicProcess}}
\]

- Inputs used
- Outputs satisfied
- Length
- Num. services

Evolutionary Approach

Initial Population

Selection

Replacement

Recombination

Evaluation

February 5, 2015
Genetic Algorithm for Automatic Composition (V)

Context-Free Grammar

- $V = \{\text{initialProcess}, \text{process}, \text{compositeProcess}\}$
- $\Sigma = \{\text{atomicProcess}, \text{choice}, \text{sequence}, \text{split}, \text{splitJoin}\}$
- $S = \text{initialProcess}$
- Rules:
  - $<\text{initialProcess}> ::= <\text{compositeProcess}> \mid \text{atomicProcess}$
  - $<\text{process}> ::= <\text{compositeProcess}> <\text{process}> \mid <\text{compositeProcess}> \mid <\text{atomicProcess}> <\text{process}> \mid <\text{atomicProcess}>$
  - $<\text{compositeProcess}> ::= \text{choice} <\text{process}> <\text{process}> \mid \text{sequence} <\text{process}> <\text{process}> \mid \text{split} <\text{process}> <\text{process}> \mid \text{splitJoin} <\text{process}> <\text{process}>$

Evolutionary Approach

Population-based selection approach

$N$ offspring + $N$ parents merged, best $N$ selected
Genetic Algorithm for Automatic Composition (VI)

Pros

- Can handle very complex control constructions
- Many different solutions (improved over time)

Cons

- Slow convergence for large compositions
- Complex and suboptimal solutions
- Hard to adjust tradeoffs in the fitness function

Contributions

Given a request, compute the shortest dependency graph of services that produces the expected outputs.

The graph is computed incrementally in polynomial time:

- The first layer ($L_1$) contains the services that are invokable with the inputs of the request.
- The second layer ($L_2$) contains the services that are invokable with the inputs of the request plus the outputs of $L_1$.
- The generation stops when the expected outputs are achieved.
Graph-based algorithm (II)

- **Optimizations to prune irrelevant services**
  - Remove all services that do not contribute to the output goals
- **Analyze equivalence / dominance of functionality**
  - Admissible state-space pruning by combining equivalent and dominated services
There are 6 different compositions: \( \{B, C\} \times \{D, E, F\} \) but:
- B and C are functionally (I/O) equivalent
- C, D and F are also functionally (I/O) equivalent

We can merge both groups of services to end with just one composition: \( \text{Sequence}(A, \text{Split(Choice}(B,C), Choice(D,E,F), G)). \)
Graph-based algorithm (II) - Interface Dominance

- Service $E$ dominates $B$, $C$, and $D$:
  - It only needs $A$ to solve its inputs
  - It resolves all the inputs of service $G$
  - Any other combination of services is redundant, i.e., leads to a composition with more services and same functionality.
Backward heuristic A* algorithm to extract the optimal composition subgraph from the graph

The algorithm starts searching from the last layer $L_N$ until it reaches $L_1$

Heuristic based cost function $f(x) = g(x) + h(x)$ where

- $g(x) = \text{number of different services selected}$
- $h(x) = \text{distance from the current layer to } L_1$ (consistent heuristic)
Graph-based algorithm (IV) - A* search example

A* backward heuristic search

L₀ \rightarrow \ldots \rightarrow L_{N-2} \rightarrow L_{N-1} \rightarrow L_N

Initial state

Neighbor state

estimated distance

Source

Sink

February 5, 2015
Graph-based algorithm (V) - Evaluation

- Evaluation with the Web Service Challenge 2008 (8 datasets)
- From 158 to 8,000 services with semantic annotations
- Graph example of the smallest dataset (158 services):
Graph-based algorithm (VI) - Results

- Our algorithm solves all the datasets with optimal results.
- It finds a solution which is better than the winners of the challenge (42 vs 46 services).

<table>
<thead>
<tr>
<th>Test</th>
<th>Gr.s.</th>
<th>iter.</th>
<th>time (ms)</th>
<th>#serv.</th>
<th>ex. path</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC’01</td>
<td>17</td>
<td>37</td>
<td>91</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>WSC’02</td>
<td>19</td>
<td>29</td>
<td>123</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>WSC’03</td>
<td>60</td>
<td>856</td>
<td>1929</td>
<td>40</td>
<td>23</td>
</tr>
<tr>
<td>WSC’04</td>
<td>31</td>
<td>18</td>
<td>314</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>WSC’05</td>
<td>62</td>
<td>1823</td>
<td>6356</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td>WSC’06</td>
<td>95</td>
<td>13</td>
<td>777</td>
<td>42</td>
<td>7</td>
</tr>
<tr>
<td>WSC’07</td>
<td>89</td>
<td>332</td>
<td>9835</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>WSC’08</td>
<td>78</td>
<td>198</td>
<td>6398</td>
<td>30</td>
<td>20</td>
</tr>
</tbody>
</table>

Main contributions

QoS-Driven Automatic Composition

- Services are associated with non-functional properties such as **response time** or **throughput**
- Extension of the previous approach to:
  - Optimize the end-to-end Quality-of-Service of the composition
  - Keep the composition simple (optimize the number of services)
- Proposed approach:
  1. Compute the service graph for a request
  2. Run an adapted version of the Dijkstra’s algorithm to obtain the best possible QoS in polynomial time (forwards)
  3. State-space search to minimize the number of services but keeping the optimal QoS (backwards)
    - Optimization: use best QoS value as a bound to prune all states that worsen the overall QoS
QoS-Driven Automatic Composition

Forward computation of the best QoS

Backward State-Space Search to minimize the services
QoS-Driven Automatic Composition - Evaluation

- We have validated the algorithm using the 5 repositories of the Web Service Challenge 2009
- We found shorter solutions in datasets 4 and 5

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Optimal QoS solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSC-2009'01</td>
<td>w1/w2</td>
</tr>
<tr>
<td>1.0/0.0</td>
<td>13</td>
</tr>
<tr>
<td>0.5/0.5</td>
<td>7</td>
</tr>
<tr>
<td>0.0/1.0</td>
<td>7</td>
</tr>
<tr>
<td>WSC-2009'02</td>
<td>1.0/0.0</td>
</tr>
<tr>
<td>0.5/0.5</td>
<td>24</td>
</tr>
<tr>
<td>0.0/1.0</td>
<td>24</td>
</tr>
<tr>
<td>WSC-2009'03</td>
<td>1.0/0.0</td>
</tr>
<tr>
<td>0.5/0.5</td>
<td>33</td>
</tr>
<tr>
<td>0.0/1.0</td>
<td>31</td>
</tr>
<tr>
<td>WSC-2009'04</td>
<td>1.0/0.0</td>
</tr>
<tr>
<td>0.5/0.5</td>
<td>73</td>
</tr>
<tr>
<td>0.0/1.0</td>
<td>72</td>
</tr>
<tr>
<td>WSC-2009'05</td>
<td>1.0/0.0</td>
</tr>
<tr>
<td>0.5/0.5</td>
<td>41</td>
</tr>
<tr>
<td>0.0/1.0</td>
<td>41</td>
</tr>
</tbody>
</table>

Main contributions

An Integrated semantic Web service discovery and composition framework was developed in collaboration with the Knowledge Media Institute, The Open University, UK

Main contributions:

- Integration with service discovery
- Reference implementation
- Performance analysis with different optimizations
Reference implementation:

- **ComposIT**: graph-based composition algorithm developed in this thesis (http://github.com/citiusus/sc/composit).
- **iServe**: service warehouse developed by the KMi, The Open University, UK. Project lead by Dr. Carlos Pedrinaci (https://github.com/kmi/iserve).

Part of this research was used in the European COMPOSE Project.
PhD Chronology

- **2010**: EVIN (Int. Journal)
- **2011**: ICWS (CORE-A) (Acceptance rate: 14%)
- **2012**: JCIS Conference
- **2013**: IJWSR (JCR Journal)
- **2014**: ICSOC (CORE-A) (Acceptance rate: 28%)
- **2015**: CISTI (Iberian conference)

- **Other activities**
  - Conferences: JCIS Conference, ICWS (CORE-A), ICSOC (CORE-A)
  - Journals: EVIN (Int. Journal), IJWSR (JCR Journal), TSC (JCR Journal)
  - Research Internship (Barrié Grant) at KMI, The Open University, UK

February 5, 2015
Thank you!

Questions? :-)

pablo.rodriguez.mier@usc.es