Ranking-Based Suggestion Algorithms for Semantic Web Service Composition

Rui Wang, Sumedha Ganjoo, John A. Miller and Eileen T. Kraemer

Presented by: John A. Miller
July 5, 2010
Outline

- Introduction & Motivation
  - Need for Increased Automation in Composition
  - Various Approaches & Existing Solutions
  - Strengths and Weaknesses of Solutions
- Our Incremental Approach
  - Service Suggestions
  - Data Mediation
  - Technological Infrastructure
    - Semantic Annotations & Similarity Measures
- Example Scenario
- Architecture & Implementation
- Evaluation
- Conclusions & Future Work
Basic Questions

- Increasing automation in service composition
  - Why increase automation?
  - What techniques have been tried?
  - How effective are those techniques?
  - Are there new approaches?
Need to Increase Automation

- Create a process by service composition
  - Select services
  - Determine data flow dependences
  - Determine control flow
  - Feed outputs into inputs

- Goal:
  - Automate as much as possible
  - Make it practical for general users

Process composed of Web services
Existing Solutions - select services

- GUI Approach
  - ActiveBPEL, NetBeans, Oracle BPEL, Taverna
Existing Solutions—feed output to input

***GUI Approach***

- ActiveBPEL, NetBeans, Oracle BPEL, Taverna
Strengths & Weaknesses

- When composing services:
  - Use GUI designers or hand code the process
    - Hand coding is hard, requires thorough technical knowledge
    - Current designers use GUI approach, it is helpful with visualization but it is time-consuming and error prone.
Strengths & weaknesses

- When composing services:
  - Users have to find suitable services
    - Use GUI designers, users still have to select suitable services
    - Current discovery approach is not aimed for service composition, so it does not consider the requirements of the data dependency or control flow of the process.
    - Automatic approach will select services without user intervention, errors may come from missing or inaccurate service annotations.
Strengths & Weaknesses

When composing services:

- Map outputs to inputs between services
  - Some designers (i.e. NetBeans) provide a GUI tool to visually drag and drop to map outputs to inputs. However, users still have to figure out the correct connections and even write some XPATH or XSLT expressions.
  - Semantic approaches
    - WSMO, OWL-S, SWSO
      - Require experts to create several subontologies for each web service, which is really hard for most people
      - Ontology mapping is still a challenge in semantic domain.
- WSDL-S/SAWSDL (bottom-up, top-down)
  » Have to meet required semantic level
    » Bottom-up approach: annotate all bottom nodes of message schema
    » Top-down approach: require XSLT or other mapping files for the whole schema of every message
Our Incremental Approach

Data Mediation Score

\[ S_{dm}(WS_x) = \text{AVG}(S_{dm}(\{l_0, O_1, O_2\}, I_x), S_{dm}(\{l_0, O_1, O_2, O_x\}, I_3)) \]

Functionality Score

\[ S_{fn}(WS_x) = \text{CS}(F_x, F'_x) \]

Formal Service Specification Score

\[ S_{pe}(WS_x) = \text{AVG}(S_{pe}(st_2, pre_x), S_{pe}(st_x, pre_3)) \]
Service Suggestions

Consider the outputs of all previous services and global inputs

As shown in picture below:

- Assume $WS_1 \rightarrow WS_2$ is already given and
- $WS_x$ is to be added to the process ($WS_1 \rightarrow WS_2 \rightarrow WS_x$)
- Data mediation algorithm will consider not only output of $WS_2$ ($O_2$) but also output of $WS_1$ ($O_1$), and global initial inputs ($I_1$)
- Therefore, the possible inputs can be fed into $WS_x$ will be $O_2$, $O_1$, $I_1$
Data Mediation

- Utilize any level SAWSDL/WSDL-S annotation
  - No annotation: based on syntax
  - Any combination of:
    - ModelReferences
      - Used for data mediation score
      - Used for functionality score
    - Lifting/LoweringSchemaMappings
      - Used for data mediation score
      - Lifting: transform XML data to ontology instances
      - Lowering: transform ontology instances to XML data
    - Preconditions & effects
      - Used for formal service specification score
Service Suggestions

- Semantic suggestion for service composition
  - Semi-automatic approach
    - Allow human to pick one of the suggestions, this should reduce errors coming from missing or inaccurate annotations
  - Ranking available Web services to suggest Web services to the user, which is based on:
    - Data mediation
    - Formal service specification
    - Functionality of services

All weights are initially set by experiences, later will be trained using machine learning algorithms.
Service Suggestions

🔹 Semantic suggestions for service composition

- \[ S = w_{dm} * S_{dm} + w_{fn} * S_{fn} + w_{pe} * S_{pe} \]
  - where \( w_{dm} = w_{fn} = w_{pe} = 1/3 \)
  - \( S_{fn} \) : Score based on functionality
    - Compare the user specified functionality \( F_x \) with candidate service’s functionality \( F_x \)
  - \( S_{dm} \) : Score based on data mediation
    - Scores calculated during data mediation
  - \( S_{pe} \) : Score based on formal service specification:
    - Pre-conditions \( (P_x) \), effects \( (E_x) \) (requires WSDL-S)
    - whether current state \( st \) will entail precondition of the candidate service
    \[ S_{pe} = \begin{cases} 1 & \text{if } st \neq \text{pre}(OP_x) \\ 0 & \text{otherwise} \end{cases} \]
Data Mediation: Score $S_{dm}$ Calculation

- Ranking score of suggested services $WS_x$ (used for suggestions)
  - Based on how well the paths of the input message of candidate service are matched by the paths of output message of previous services and global initial inputs

$$S_{dm} = \sum_{i=1}^{t} w_i \times S_i$$

- Example path: record.teacher.name
- $S_i$: score of a path of the input of candidate service
- $t$: number of paths of the input of candidate service
- $w_i = 1/t$ (by default)
Data Mediation: Score $S_i$ Calculation

- Ranking score for a path $P_0$ within the input message $I_x$ of a candidate operation $WS_x$

```plaintext
function PATH-RANK (\{P_1, P_2, \ldots, P_n\}, P_0)
begin
    // \{P_1, P_2, \ldots, P_n\} is a set of existing paths to be compared to $P_0$
    for i in 1..n do
        S_i = COMPARE-2-PATHS (P_i, P_0)
    end
    k = arg-max\{S_1, S_2, \ldots, S_n\}
    return < S_k, P_k >
    // $S_k$ is the matching score between $P_k$ and $P_0$,
    // $P_k$ is the best matching path to $P_0$
end

function COMPARE-2-PATHS (P_i, P_0)
begin
    \{A_1, A_2, \ldots A_j \ldots, A_m\} = semantic annotations of all nodes on $P_0$
    \{w_1, w_2, \ldots, w_m\} = weights of all nodes on $P_0$
    \{A'_1, A'_2, \ldots A'_j \ldots, A'_z\} = semantic annotations of all nodes on $P_i$
    $L = \min\{m!z\}$
    return $\sum_{j=1}^{L} w_j \times CS(A'_j, A_j)$
    // $CS(A'_j, A_j)$ is the ontological concept similarity score of $A_j$ and $A'_j$
end
```

- For the weights $w_1, w_2, \ldots, w_m$ in COMPARE-2-PATHS()
  - We use geometric series decreasing from the leaf node to the root node.
  - $w_1 + w_2 + \ldots, + w_m = 1$
Data Mediation: Score $S_i$ Calculation

- Ranking based bi-directional data mediation
  - Ranking score for a path
    - Path in a message:
      - the root node (top node) to a bottom node
    - CS(): Compare semantic similarity between the annotations of the nodes that exist on the two paths in the two messages
    - Topological based: different weight for each node, geometric series decreasing from the leaf node to the root node

All weights are initially set by experiences, later will be trained using machine learning algorithms
Output Message

Input Message

\[
\text{SCORE} (P_4, P_0) = \text{CS (ID, Name)} \times W_0 + \text{CS (Student, Teacher)} \times W_8 \\
= 0.1 \times 0.67 + 0.2 \times 0.33 = 0.133
\]

\[
\text{SCORE} (P_3, P_0) = \text{CS (Name, Name)} \times W_0 + \text{CS (Student, Teacher)} \times W_8 \\
= 1 \times 0.67 + 0.2 \times 0.33 = 0.703
\]

\[
\text{SCORE} (P_2, P_0) = \text{CS (Name, Name)} \times W_0 + \text{CS (Teacher, Teacher)} \times W_8 \\
= 1 \times 0.67 + 1 \times 0.33 = 1
\]

\[
\text{SCORE} (P_1, P_0) = \text{CS (Email, Name)} \times W_0 + \text{CS (Teacher, Teacher)} \times W_8 \\
= 0.1 \times 0.67 + 1 \times 0.33 = 0.397
\]

\[
\text{SCORE}(P_0) = \text{MAX} \{\text{SCORE} (P_1, P_0), \text{SCORE} (P_2, P_0), \text{SCORE} (P_3, P_0), \text{SCORE} (P_4, P_0)\} = \text{SCORE} (P_2, P_0) = 1
\]
Similarity Measure

- Concept Similarity (CS) computes the overall similarity between two concepts
  - Syntactic_{sim} computes the syntactic similarity between the names and descriptions of the two concepts.
    - Using a string matching algorithm
  - Coverage_{sim} computes the similarity based on the relative position of the two concepts in the ontology
  - Given that concept C_1 has properties P_1 and concept C_O has properties P_O, Property_{sim} calculates an overall similarity measure between P_1 and P_O
    - The properties are matched as one-to-one mappings, using the Hungarian algorithm

\[ CS = w_1 \times \text{Syntactic}_{sim} + w_2 \times \text{Coverage}_{sim} + w_3 \times \text{Property}_{sim} \]
Formal Service Specification

- Preconditions are required to be true before an operation can be successfully invoked.
- Effects must be true after an operation completes execution after being invoked.
- WSDL-S is used:
  - Precondition and effect are added as extensible elements on an operation.
  - Prolog language described precondition and effect are annotated as values of modelReference in WSDL-S.
  - Sample precondition & effect for service operation getIds:
    - `<wssem:precondition name="getIdsPrecond" wssem:modelReference="hasBlastJobid."/>
    - `<wssem:effect name="getIdsEffect" wssem:modelReference="assertz(hasBlastHitIds(blasthitid)), assertz(isStringArray(blasthitid)), assertz(isHomolog(blasthitid))."/>`
We use Prolog to maintain the state

- State of the process is maintained as a Prolog knowledge base (KB).
  - We originally considered using RIF, however as to the best of our knowledge, there is no mature RIF engine for Java.
  - RIF is a Prolog style language and Prolog has many mature engines and Java support, so we choose Prolog for current implementation.

- At first, the initial state is used by the Prolog engine to initialize the knowledge base.
- We implement isEntail() method to query the KB whether the precondition of the candidate service is entailed by the current state, KB.
- updateState() method is implemented to update the KB, current state, with effects of an operation when the operation is added to current process.
Example Scenario

Complex task:
- protein sequence --> multiple sequence alignment (MSA)
- Use real world Web services, such as EBI Web services.
  http://www.ebi.ac.uk/Tools/webservices/
- Possible Web service composition (WSC):
  - runWUBlast → getIds → array2string → fetchBatch → runClustalw2 → poll → byte2string

Issues:
- Which Web services to use?
  - Some biologists may know they need Blast and ClustalW, but that is not enough, what is in between?
  - So many Web services available on EBI web site.
  - Even one Web service may have multiple operations.
    - EBI WUBLast services have 19 operations, which operation will be used?
- How to connect two Web service operations?
  - Which output of the previous operations can feed to the next operation?
Architecture
Implementation

Modules Implemented:

- **Suggestion engine**
  - Invoke three modules to calculate suggestion ranking score based on data mediation, formal service specification and functionality

- **Data mediation engine**
  - Be able to run independently to map input/output between different service operations.
  - Invoke similarity measure to retrieve concept similarity score
  - Java, Jdom, Jaxen, Jena

- **Knowledge base management system (KBMS)**
  - Be able to initialize a knowledgebase, query the knowledge base for entailment of precondition and update the knowledge base.
  - SWI-Prolog, JPL

- **Similarity measure module**
  - Provide concept similarity score to data mediation engine and suggestion engine

- **Parsers**
  - Parse SAWSDL / WSDL-S and OWL files for suggestion engine and data mediation engine
Evaluation

Service suggestions

- Hypothesis:
  - Good precision of the suggested services
  - Tool’s response time is acceptable by users
  - Able to suggest data type/format converters

- Preliminary Evaluation:
  - Test with our motivating scenario: seven operations in our process.
    - Ask for suggestions for every operation to be added next. Our approach can successfully suggest the correct service.
    - Time each suggestion and calculate the average time used.
    - Successfully suggested the converter required between getIds and fetchBatch
Evaluation

- Data mediation
  - Hypothesis:
    - Our approach can map input/output between services correctly
    - Lower semantic requirements compared to other semantic approaches
    - Works for special case:
  - Evaluation:
    - Test with our motivating scenario, our approach correctly mapped input/output between services.
    - Test special case with our motivating scenario, global input (email) correctly map to the input of runWUBlast and runClustalw; global input (db) correctly map to the input of runWUBlast and fetchBatch.
    - Test special case with our motivating scenario using different data mediation approaches: top-down and bottom-up.
      - Non-semantic annotation, the other two fail to run, our approach works (but correctness ratio is lower)
      - Fully semantic annotated WSDL, top-down approach can not handle the two mappings: email and db; bottom-up approach considers them as missing value and did not find correct mappings
Conclusions

- Developed an approach for making suggestions to aid users composing services.
- Implemented an independent external Web service suggestion engine, which can hook to a WSC designer to help users composing services.
- Developed a new data mediation algorithm that extends our previous work on top-down and bottom-up data mediation.
- Implemented a data mediation model which can attach to a WSC designer to help users composing services.
- First-order logic (Prolog) is used for formal service specification. As to the best of our knowledge, we are the first one applying Prolog to WSDL-S for the formal service specification.
- Utilizing formal service specification to guide semi-automatic Web service composition.
Future Work

- Consider QoS while suggesting services
- Using a planner to generated chained services when one service is not sufficient.
- Consider suggesting complex structure: loop, branches, parallel
- When a mature RIF engine becomes available, we would like to use RIF for our formal service specification and related implementation. This would greatly help resolving the semantic heterogeneities between the terms used in the service specification
- Utilize during workflow design using Galaxy
THANK YOU!
Existing Solutions

- **GUI**
  - NetBeans BPEL designer
  - ActiveVOS designer
  - Taverna
  - Oracle BPEL designer

- **Planner** (to the best of our knowledge, in past year, new approaches extend/customize planners such as below)
  - HTN while enforcing Regulations, Markov-HTN
  - GOLOG with user preferences
  - Color Petri Net (CPN), Associate Petri Net (APN), High Level Petri Net, Elementary Petri Net
  - Dynamic Description Logic (DDL)
  - Conformant-FF
  - Factored Markov Decision Process
  - Fluent Calculus
  - Integer Linear Programming planner
  - Multi – agent planning
  - QoS + planner

- **Semantic**
  - UML model + semantic
  - Agents + semantic
  - QoS + semantic
  - Our approach: SAWSDL/WSDL-S + Logical Programming (Prolog)
New approaches

- Social trust + OWL-S
- Service dependency graph and bidirectional heuristic search
- Service suggestion approach
Various Approaches