Scalable Semantic Web Service Discovery for Goal-driven Service-Oriented Architectures

Michael Stollberg
PhD Thesis
University of Innsbruck
Semantic Technology Institute
March 2008
1. Introduction
   • Research Context
   • Problem Statement

2. Solution
   • Revised Goal Model for SWS
   • 2-phase Web Service Discovery
   • Semantic Discovery Caching

3. Evaluation
   • Performance Analysis (quantitative)
   • Applicability Study (qualitative)

4. Related Work
   • Foundations & Complementary Works
   • Competitive Works

5. Conclusions
   • Strengths & Limitations
   • Summary and Contributions
   • Relevant Publications
Introduction – Research Context

Web Services & SOA

• **Service-Oriented Architecture (SOA)**
  
  Use Web Services as basic building blocks
  
  – Latest paradigm for IT systems
  
  – Massive uptake in industry & research

• **Web Service**
  
  – Program (problem-solving facility) accessible over the Web
  
  – Standardized Interface, language & platform independent
  
  – Basic technologies: SOAP, WSDL, UDDI

• **Benefits**
  
  – Use the Web as infrastructure for computation
  
  – Cost reduction by reuse & combination
  
  – Facilitate interoperability (intra- & inter-organizational)

• **Sophisticated SOA technology is a grand challenge …**
Introduction – Research Context

Web Service Usage Cycle

- Repository (e.g. UDDI) points to WSDL
  - Data (XML)
  - Operations
  - Endpoint

  - Find usable Web Service

  - Categorization
  - Search facilities

  - Enables technical provision & usage of Web Services
  - Limits detection of suitable WS to manual inspection
• **Aim**: Overcome deficiencies of basic SOA technology  
  – Higher degree of automation  
  – High-quality mechanisms  
  – Facilitate semantic interoperability  

• **Approach**: automate WS Usage Cycle by  
  1. Ontology-based descriptions of Web Services and related elements  
  2. Inference-based techniques for automated discovery, composition, mediation, and execution  

• **Frameworks** (most relevant)  
  1) OWL-S  
  2) WSMO  
  3) SAWSDL
Introduction – Problem Identification

Automated Web Service Usage Cycle

Request

if successful: solved
else: try other WS

Submitted by client

Discovery

if: usable
else: try composition

Data Mediation

Process Mediation

Selection / Ranking

Behavioural Conformance

Composition

uses

Executor

uses

if: execution error

Web Service Repository

matchmaking with all WS

uses

uses

composition (executable)

information lookup for particular WS

uses
Automated Web Service Discovery

• Discovery is a central operation
  – Detect functionally usable Web Services for a client request
  – First processing operation in SWS environments
  – Heavily used component for more complex SWS techniques

• Requirements for Automated Discovery Engines
  1. High Retrieval Accuracy (precision & recall)
  2. High Computational Performance (time & scalability)

• State of the Art
  – Aspects & architectural allocation of discovery in SWS systems
  – Semantic matchmaking for several languages
  – WS categorization for enhancing computational performance
- **Aim**: scalable runtime discovery with high retrieval accuracy
- **Approach**: extensions & refinements for WSMO framework
The Goal-driven Approach

• **A Goal**
  – Describes objective that a client wants to achieve
  – Problem-oriented, abstracting from technical details

• **Goal-driven Web Service Usage**
  – Client describes request as a Goal (instantiation)
  – System automatically detects & executes the necessary Web Services for solving the goal

• **Benefits**
  1. **Client Support**: problem-oriented WS usage (abstraction layer)
  2. **Flexibility**: dynamic detection of actual Web Services

• **Promoted by WSMO, based on previous works**
1. Differentiation Goal Templates – Goal Instances

**Goal Template G**
- Generic & reusable objective description stored in the system
- Described by:
  - domain knowledge (*ontology*)
  - objective (*capability*)
  - conditions (*non-functional properties*)
  - decomposition (*orchestration*)

**Goal Instance GI(G,ß)**
- Describes a concrete client objective
- Instantiates a goal template G with concrete inputs (input binding ß)

2. Automated Web Service Consumption via Client Interfaces

- Client Interface
  - Compatible behavior (*choreography*) for consuming a Web Service to solve a goal
  - Concrete inputs (input binding ß) used for actual invocation
- Allocated in a *mediator* that connects a Goal Template with a usable WS
- Replaces „requested Interface“ in WSMO Goals
Solution – Goal Model

Goal Model Overview

Goal Template
- domain knowledge: ontology
- objective: capability
- constraints: condition
- decomposition: orchestration

Mediator
- source: goal template
- target: web service
- behavior: choreography

Web Service
- domain knowledge: ontology
- functionality: capability
- conditions: non-funct. property
- interface: choreography
- composition: orchestration

Client Interface
- behavior: choreography

Goal Instance
- template: goal template
- inputs: input binding

refinement / extension of WSMO Goal Model
Solution – Discovery

2-phase Web Service Discovery

• **Aim:** *high retrieval accuracy by matchmaking of ontology-based functional descriptions of Goals and Web Services*

• **Two phases:**
  1. **Design Time Discovery**
     detect suitable Web Services $W$ for Goal Template $G$
  2. **Runtime Discovery**
     detect actual Web Service(s) $W$ for Goal Instance $GI(G,\beta)$

• **Realization** follows standard approach
  - Formal Functional Descriptions (preconditions / effects)
  - First-Order Logic (FOL) as Specification Language
  - Matchmaking by Automated Theorem Proving (VAMPIRE)
  - Conceptual model: abstract sequences of states $T = (s_{start}, s_{end})$ with
    - $\{T\}_W$ = set of all abstract executions Web Service $W$
    - $\{T\}_G$ = set of all abstract solutions of Goal Template $G$
Solution – Discovery

3-layered Abstraction Model

- **Level 1**: Abstract State Space Model (Keller et al. 2006)
- **Level 2**: abstraction relevant for discovery task
- **Level 3**: reasoning on formal functional descriptions

1. Observable executions of WS / solutions for goals (sequences of states)
2. Abstract executions / solutions (relation start- & end-state)
3. Formal functional descriptions (logical models)
Solution – Discovery

**Functional Descriptions**

- **Formal description of provided / requested functionality**
- **Elements of Functional Description \( \mathcal{D} \)**
  1. **Signature** \( \Sigma \): FOL extended with dynamic symbols
  2. **Domain Ontology** \( \Omega \): consistent terminology & background knowledge
  3. **Input Variables** \( \text{IN} = (i_1, \ldots, i_n) \) (input binding \( \beta \) = variable assignment)
  4. **Precondition**: FOL-formula \( \varnothing_{\text{pre}} \) with free variables \( V_{\text{free}} \supseteq \text{IN} \)
  5. **Effect**: formula \( \varnothing_{\text{eff}} \) with same free variables \( V_{\text{free}} \supseteq \text{IN} \)
  6. **Output Variables**: constrained in \( \varnothing_{\text{eff}} \) by predicate \( \text{out}(o_1, \ldots, o_m) \)

- **Representation as single FOL formula**

\[
\mathcal{D} = (\Sigma, \Omega, \text{IN}, \varnothing^D) \text{ with FOL formula } \varnothing^D := [\varnothing_{\text{pre}}]_{\text{replace}} \Rightarrow \varnothing_{\text{eff}}
\]

where \([\varnothing]_{\text{replace}}\) replaces all dynamic symbols with their pre-variant

\( \Rightarrow \) allows reasoning with classical model-theoretic semantics
### Solution – Discovery

#### Example Functional Descriptions

<table>
<thead>
<tr>
<th>Goal Template $G$</th>
<th>Web Service $W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ship a package of any weight in Europe”</td>
<td>“shipment in Germany, max. 50 kg”</td>
</tr>
<tr>
<td>$\Omega$: location &amp; shipment ontology</td>
<td>$\Omega$: location &amp; shipment ontology</td>
</tr>
<tr>
<td>$IN$: {?s, ?r, ?p, ?w}</td>
<td>$IN$: {?s, ?r, ?p, ?w}</td>
</tr>
<tr>
<td>$\phi^{pre}$: $person(?s) \land in(?s, europe)$</td>
<td>$\phi^{pre}$: $person(?s) \land in(?s, germany)$</td>
</tr>
<tr>
<td>$\land person(?r) \land in(?r, europe)$</td>
<td>$\land person(?r) \land in(?r, germany)$</td>
</tr>
<tr>
<td>$\land package(?p) \land weight(?p, ?w)$</td>
<td>$\land package(?p) \land weight(?p, ?w)$</td>
</tr>
<tr>
<td>$\land maxWeight(?w, heavy)$.</td>
<td>$\land maxWeight(?w, 50kgClass)$.</td>
</tr>
<tr>
<td>$\phi^{eff}$: $\forall ?o, ?price. out(?o) \Leftrightarrow ($</td>
<td>$\phi^{eff}$: $\forall ?o, ?price. out(?o) \Leftrightarrow ($</td>
</tr>
<tr>
<td>$\land shipmentOrder(?o, ?p)$</td>
<td>$\land shipmentOrder(?o, ?p)$</td>
</tr>
<tr>
<td>$\land sender(?p, ?s) \land receiver(?p, ?r)$</td>
<td>$\land sender(?p, ?s) \land receiver(?p, ?r)$</td>
</tr>
<tr>
<td>$\land costs(?o, ?price)$ ).</td>
<td>$\land costs(?o, ?price)$ ).</td>
</tr>
</tbody>
</table>
Solution – Discovery

Meaning of a Match

Basic condition under which a Web Service is suitable to solve a goal with respect to the provided and requested functionality

1. Goal Template Level
   - $D_W$ describes all abstract executions $\{T\}_W$ of Web Service $W$
   - $D_G$ describes all abstract solutions $\{T\}_G$ of Goal Template $G$

   \[
   \text{match}(G,W): \{T\}_G \cap \{T\}_W \neq \emptyset
   \]

2. Goal Instance Level
   - Goal Instance $GI(G,\beta)$ instantiates Goal Template $G$ with input binding $\beta$
   - Validation: $GI(G,\beta) \models G$ if $[D_G]_\beta$ is satisfiable; then $\{T\}_{GI(G,\beta)} \subseteq \{T\}_G$
   - $[D_W]_\beta$ describes all abstract executions $\{T\}_{W(\beta)}$ of $W$ when invoked with $\beta$

   \[
   \text{match}(GI(G,\beta),W): \{T\}_{GI(G,\beta)} \cap \{T\}_{W(\beta)} \neq \emptyset
   \]
Solution – Discovery
Matchmaking – Goal Template Level

4 Degrees where \( \text{match}(G,W) \)

**exact**

\[ \Omega^* \models \forall \beta. \phi^G \iff \phi^W \]

**plugin**

\[ \Omega^* \models \forall \beta. \phi^G \Rightarrow \phi^W \]

**subsume**

\[ \Omega^* \models \forall \beta. \phi^G \leftarrow \phi^W \]

**intersect**

\[ \exists \beta. \bigwedge \Omega^* \land \phi^G \land \phi^W \text{ is satisfiable} \]

**disjoint**

\[ \exists \beta. \bigwedge \Omega^* \land \phi^G \land \phi^W \text{ is unsatisfiable} \]

formal relations:

(i) plugin & subsume \( \iff \) exact

(ii) plugin \( \Rightarrow \) intersect

(iii) subsume \( \Rightarrow \) intersect

(iv) \( \neg \) intersect \( \iff \) disjoint
Solution – Discovery  

Matchmaking – Goal Instance Level

• Using the results of design time discovery

\( W \) is suitable for solving a Goal Instance \( GI(G, \beta) \vdash G \) if and only if:

1. exact \( (\mathcal{D}_G, \mathcal{D}_W) \)  
2. plugin \( (\mathcal{D}_G, \mathcal{D}_W) \)  
3. subsume \( (\mathcal{D}_G, \mathcal{D}_W) \) and \( \Omega^* \land [\phi_{\mathcal{D}_W}]_\beta \) is satisfiable, or
4. intersect \( (\mathcal{D}_G, \mathcal{D}_W) \) and \( \Omega^* \land [\phi_{\mathcal{D}_G}]_\beta \land [\phi_{\mathcal{D}_W}]_\beta \) is satisfiable.

• This means:

1. only \( W \) with \text{match}(G,W) \) need to be considered
2. under the \text{exact} or the \text{plugin} degree, no matchmaking is needed at runtime because then \( \{T\}_{GI(G,\beta)} \subseteq \{T\}_G \subseteq \{T\}_W \)
3. the additional matchmaking required under the \text{subsume} degree is simpler than under the \text{intersect} degree

\( \Rightarrow \) \text{minimal matchmaking effort for runtime discovery}
Solution – SDC
Semantic Discovery Caching (SDC)

• **Aim**: enhance computational performance of discovery task
  – Warrant operational reliability of SWS environments
  – Optimize runtime discovery as time critical operation
  – performance parameters: **efficiency, scalability, stability**

• **Approach**:
  1) **Build & maintain SDC Graph that**
     - Organizes Goal Templates in subsumption hierarchy
     - Captures minimal knowledge on design time discovery results
  
  2) **Exploit SDC Graph to optimize Runtime Discovery**
     - Maximal reduction of search space
     - Minimizing number of necessary matchmaking operations
Solution – SDC

The SDC Graph

• Elements & Structure:
  – Goal Graph (inner layer): subsumption hierarchy of Goal Templates
  – Discovery Cache (outer layer): captures design time discovery results

• Directed Acyclic Graph (DAG)
• Structure is unambiguous & constant
• Automated update whenever G or W is added / removed / modified (creation & maintenance algorithms)
Solution – SDC

Goal Graph

• Functional specialization hierarchy of Goal Templates

• Basis: *functional similarity of Goal Templates*
  – $G_1, G_2$ are similar if they have at least 1 common solution
  – Described by similarity degrees (analog to matching degrees)

• Definition & Properties
  – **Only occurring** similarity degree is $\text{subsume}(G_1, G_2)$
  – Defined by directed arcs $(G_1, G_2)$
  – Only 1 possible position for each Goal Template
  – No redundant arcs in goal graph

• Creation: *handling of possible similarity degrees*
  1. $\text{exact}(G_1, G_2)$ only 1 is kept (the other one is redundant)
  2. $\text{subsume}(G_1, G_2)$ store arc $(G_1, G_2)$
  3. $\text{plugin}(G_1, G_2)$ store arc $(G_2, G_1)$
  4. $\text{intersect}(G_1, G_2)$ add intersection goal template $G^I(G_1, G_2)$
  5. $\text{disjoint}(G_1, G_2)$ keep both as disconnected nodes
Solution – SDC

Intersection Goal Templates

- **Virtual** Goal Template added to the SDC Graph for every occurrence of the similarity degree $\text{intersect}(G_1, G_2)$
- $G^I(G_1, G_2)$ describes common solutions of $G_1, G_2$
  - Defined as logical conjunction of functional descriptions of $G_1, G_2$
  - $G^I(G_1, G_2)$ becomes child of both $G_1$ and $G_2$ in Goal Graph

- **Benefits:**
  - Avoids potential cycles in SDC graph
  - Ensures proper subsumption hierarchy
Solution – SDC

Example Goal Graph – Shipment Scenario

- Goal Graph Management:
  - Adjusted after each insertion / deletion / modification
  - Resulting structure is regardless of insertion order
**Solution – SDC**

**Discovery Cache**

- **Captures usability of Web Services** (from design time discovery)
  - Defined by *directed arcs* \((G,W)\) annotated with exact usability degree
  - Only *minimal set of arcs* that is necessary to deduce the exact usability degree of every \(W\) for each \(G\)
- **Logical basis**: *inference rules* for WS usability among similar goals

---

**Inference Rules for \(\text{subsume}(G_i, G_j)\)**

1. \(\text{exact}(G_i, W) \Rightarrow \text{plugin}(G_j, W).\)
2. \(\text{plugin}(G_i, W) \Rightarrow \text{plugin}(G_j, W).\)
3. \(\text{subsume}(G_i, W) \Rightarrow \text{exact}(G_j, W) \lor \text{plugin}(G_j, W) \lor \text{subsume}(G_j, W) \lor \text{intersect}(G_j, W) \lor \text{disjoint}(G_j, W).\)
4. \(\text{intersect}(G_i, W) \Rightarrow \text{plugin}(G_j, W) \lor \text{intersect}(G_j, W) \lor \text{disjoint}(G_j, W).\)
5. \(\text{disjoint}(G_i, W) \Rightarrow \text{disjoint}(G_j, W).\)
Solution – SDC

SDC Graph Creation Algorithm

1. **START**
   
   *insert $G_{new}$*

2. Calculate **similarity** ($G_{root}, G_{new}$)

3. **YES** (repeat for next root node)

   - **exact?**
     - **NO**
     - **plugin?**
       - **NO**
       - **subsume?**
         - **NO**
         - **intersect?**
           - **NO**
           - **other $G_{root}$?**
             - **NO**
             - **add $G_{new}$** (disconnected)

   - **YES**
     - **root node**

   - **YES**
     - **child node**

   - **YES**
     - **i-arc**

4. **YES**

5. **discovery** (root node)

6. **discovery** (child node)

7. **END**

   *updated SDC graph*
Solution – SDC

SDC Graph Maintenance

• Ensure that SDC Graph exposes its structure & properties at all times in evolving environments

• **Algorithms** for basic evolution support
  
  **(1) Goal Graph Maintenance**
  
  • goal template removal & update
  • manual maintenance operations

  **(2) WS Change Management**
  
  • Web service insertion, removal, update
  • integrated with WS repositories

=> SDC Graph is properly maintained whenever a Goal Template or a Web Service is added, removed, or modified
Solution – SDC

SDC Graph Properties

- **Directed Acyclic Graph (DAG)**
  - Multiple root / parent nodes possible (e.g. disjoint Goal Templates)
  - No cycles can appear (Intersection Goal Templates)

- **Structure is unambiguous & constant**
  - Only 1 possible position for each Goal Template
  - Only 1 exact usability degree of each $W$ for every $G$
  - Maintained in every situation (automated management)

- **Inferential Completeness & Minimality**
  - Captures all knowledge relevant for optimizing the discovery task
  - Only minimal set of necessary arcs is defined

- **Computational Complexity**:
  \[ O( \text{diam}(GG) \times \text{breadth}(GG)) + |W(G)| ) \]
  for both management & search operations
  - **Minimal**: $O(\log n)$ if SDC Graph is a binary search tree
  - **Maximal**: $O(n)$ if all $G$ are disjoint with distinct sets of usable $W$
Optimized Runtime Discovery

• **Exploit SDC Graph to enhance computational performance by**
  – Maximal reduction of search space
  – Minimizing number of necessary matchmaking operations

• **Algorithm:** “find 1 usable \( W \) for a Goal Instance \( GI(G,\beta) \)”

  1. **Look-Up: without matchmaking**
     – Look for \( W \) with \( d(G,W) = \text{exact, plugin} \) in Discovery Cache
     – If successful: return \( W \)

  2. **Refinement: stepwise traversal of SDC Graph**
     1. Replace current \( G \) with a child \( G' \) where \( \text{subsume}(G,G') \) and \( GI(G,\beta) \models G' \)
        – There can only be one such \( G' \) for every \( G \)
        – For lowest \( G' \): set of candidate \( W \) is minimal
     2. Try **Look-Up** for new \( G' \)

  3. **Inspect other \( W \): candidates that require matchmaking**
     1. inspect \( W \) with \( \text{subsume}(G,W) \)
     2. inspect \( W \) with \( \text{intersect}(G,W) \)

• **Algorithm for finding all \( W \) for \( GI(G,\beta) \):** (1) refinement, (2) look-up + other \( W \)
**Example Optimized Runtime Discovery**

- **Client Request:** "ship a 5.16 kg package from San Francisco to NYC"
- **Goal Instance \( \text{GI}(G,\beta) \):**
  - Goal Template \( G \): \( \text{GT-US2world} \)
  - Input Binding \( \beta = (?s\.loc | \text{San Francisco}, ?r\.loc | \text{NYC}, ?w | 5.16 \text{ kg}) \)
- **SDC Graph:**

<table>
<thead>
<tr>
<th>WS</th>
<th>Sender</th>
<th>Receiver</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muller</td>
<td>USA</td>
<td>AF,EU,NA,AS</td>
<td>w50lq</td>
</tr>
<tr>
<td>Runner</td>
<td>USA</td>
<td>EU,AS,SA,OC</td>
<td>heavy</td>
</tr>
<tr>
<td>Weasel</td>
<td>USA</td>
<td>USA</td>
<td>heavy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>wsMuller</th>
<th>wsRunner</th>
<th>wsWeasel</th>
</tr>
</thead>
<tbody>
<tr>
<td>gtUS2world</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
</tr>
<tr>
<td>gtNA2NAlight</td>
<td>intersect</td>
<td>X</td>
<td>intersect</td>
</tr>
<tr>
<td>gtUS2NA</td>
<td>intersect</td>
<td>X</td>
<td>subsume</td>
</tr>
<tr>
<td>iGT</td>
<td>plugin</td>
<td>X</td>
<td>intersect</td>
</tr>
</tbody>
</table>
(1) Look-Up**(GT-US2world)**: **NO**
(2) Iterative Refinement ...
Solution – SDC

Example: Step 2

(1) Instantiation Check($\text{GT-US2NA}$): YES
(2) Look-Up($\text{GT-US2NA}$): NO
(3) Iterative Refinement …
(1) Instantiation Check (Intersection GT): YES
(2) Look-Up (Intersection GT): YES
⇒ discovered wsMuller (usability degree = plugin)
[also msWeasel is usable for the goal instance]
Solution – SDC

SDC Prototype Architecture

- Implemented as WSMX discovery component
- Available at: http://www.michael-stollberg.de/phd/
Solution – SDC

Goal-based SWS Browser (in WSMT)
Evaluation

Evaluation Aspects

1. Quantitative

*Aim:* determine & quantify achievable performance increase

- Performance Tests (SWSC Shipment Scenario)
  1. SDC Graph Creation & Management
  2. Runtime Discovery: SDC vs. Naive Engine
  3. Runtime Discovery: SDC-full vs. SDC-light

2. Qualitative

*Aim:* assess general applicability

- Estimation Model for applying SDC
- Verizon SOA System
- Other Use Cases
Evaluation – Quantitative

SDC Graph Management

• **Test Set-Up**
  – 15 Goal Templates
  – 100 Web Services (max. 5 are usable)
  – 10 repetitions

• **SDC Graph Creation** (Average Times):
  – complete creation: **64.81 sec.**
  – root node insertion: **19.88 sec.** (0.2 sec. / matchmaking operation)
  – child node insertion: **4.60 sec.**

• **SDC Maintenance** (Average Times):
  – Removal of G / W: **0.034 sec.** (no matchmaking)
  – Insertion of Web Service: **2.89 sec.**

• **SDC Graph correct after every operation**
Evaluation – Quantitative

SDC Graph Shipment Scenario – Goal Graph

- gtRoot
  - gtUS2world
    - gtUS2EU
    - gtUS2AF
    - gtUS2AS
    - gtUS2SA
    - gtUS2OC
    - gtUS2NA

- gtNA2NAlight

- iGT

Nodes:
- sender: world
- receiver: world
- weight: heavy
- sender: USA
- receiver: world
- weight: heavy
- sender: N, America
- receiver: N, America
- weight: light
- sender: USA
- receiver: Europe
- weight: heavy
- sender: USA
- receiver: Africa
- weight: heavy
- sender: USA
- receiver: Asia
- weight: heavy
- sender: USA
- receiver: S, America
- weight: heavy
- sender: USA
- receiver: Oceania
- weight: heavy
- sender: USA
- receiver: N, America
- weight: heavy

Edges:
- sender: world
- receiver: world
- weight: heavy
- sender: USA
- receiver: world
- weight: heavy
- sender: N, America
- receiver: N, America
- weight: light
- sender: USA
- receiver: Europe
- weight: heavy
- sender: USA
- receiver: Africa
- weight: heavy
- sender: USA
- receiver: Asia
- weight: heavy
- sender: USA
- receiver: S, America
- weight: heavy
- sender: USA
- receiver: Oceania
- weight: heavy
- sender: USA
- receiver: N, America
- weight: heavy
## Evaluation – Quantitative

### SDC Graph Shipment Scenario – Discovery Cache

<table>
<thead>
<tr>
<th>Usability Degrees</th>
<th>Muller</th>
<th>Racer</th>
<th>Runner</th>
<th>Walker</th>
<th>Weasel</th>
<th>ship 1</th>
<th>ship 2</th>
<th>ship 3</th>
<th>ship 4</th>
<th>ship 5</th>
<th>ship 6</th>
<th>ship 7</th>
<th>ship 8</th>
<th>ship 9</th>
<th>ship 10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>level 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gtRoot</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
</tr>
<tr>
<td><strong>level 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gtUS2world</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
<td>subsume</td>
</tr>
<tr>
<td>gtNA2NAlight</td>
<td>intersect</td>
<td>intersect</td>
<td>intersect</td>
<td>intersect</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>level 3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gtUS2AF</td>
<td>intersect</td>
<td>intersect</td>
<td>X</td>
<td>intersect</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2AS</td>
<td>intersect</td>
<td>intersect</td>
<td>plugin</td>
<td>intersect</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2EU</td>
<td>intersect</td>
<td>intersect</td>
<td>plugin</td>
<td>intersect</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2NA</td>
<td>intersect</td>
<td>intersect</td>
<td>X</td>
<td>intersect</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2OC</td>
<td>X</td>
<td>intersect</td>
<td>plugin</td>
<td>intersect</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2SA</td>
<td>X</td>
<td>intersect</td>
<td>plugin</td>
<td>intersect</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
<tr>
<td><strong>level 4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gtUS2AFlight</td>
<td>plugin</td>
<td>plugin</td>
<td>X</td>
<td>plugin</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2ASlight</td>
<td>plugin</td>
<td>plugin</td>
<td>plugin</td>
<td>plugin</td>
<td>(optional)</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2OClight</td>
<td>X</td>
<td>plugin</td>
<td>plugin</td>
<td>plugin</td>
<td>(optional)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2SAlight</td>
<td>X</td>
<td>plugin</td>
<td>plugin</td>
<td>plugin</td>
<td>(optional)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
<tr>
<td>gtUS2EUnight</td>
<td>plugin</td>
<td>plugin</td>
<td>plugin</td>
<td>plugin</td>
<td>(optional)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
<tr>
<td>iGT</td>
<td>plugin</td>
<td>plugin</td>
<td>X</td>
<td>plugin</td>
<td>intersect</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>subsume</td>
<td>X</td>
</tr>
</tbody>
</table>

PhD Thesis - Michael Stollberg 38
Evaluation – Quantitative

Runtime Discovery: SDC vs. Naive Engine

• Aims
  – Demonstrate need for optimization
  – Quantify performance increase achievable with SDC

• Compared Engines
  a) SDC-optimized Runtime Discoverer (see above)
  b) Naive Runtime Discoverer
     • Uses same matchmaking techniques & infrastructure
     • Inspects all available Web Services in random order

• Test Set-Up
  – 15 Goal Templates
  – Web Services: 10, 50, 100, …, 1500, 2000 (max. 5 are usable)
  – 10 Goal Instances
  – 50 repetitions

• Flavors of Web Service Discovery:
  1. Single WS Discovery: terminate after detecting 1 usable W
  2. All WS Discovery: find all usable W out of available ones
## Evaluation – Quantitative

### SDC vs. Naive Engine (single WS Discovery)

<table>
<thead>
<tr>
<th>no. of WS</th>
<th>Engine</th>
<th>Mean $\mu$</th>
<th>Median $\bar{x}$</th>
<th>Standard Deviation $\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>SDC</td>
<td>0.28</td>
<td>0.27</td>
<td>0.03 (11.74 %)</td>
</tr>
<tr>
<td></td>
<td>naive</td>
<td>0.41</td>
<td>0.39</td>
<td>0.21 (51.71 %)</td>
</tr>
<tr>
<td>100</td>
<td>SDC</td>
<td>0.29</td>
<td>0.28</td>
<td>0.03 (11.53 %)</td>
</tr>
<tr>
<td></td>
<td>naive</td>
<td>3.96</td>
<td>3.68</td>
<td>2.55 (64.48 %)</td>
</tr>
<tr>
<td>1000</td>
<td>SDC</td>
<td>0.29</td>
<td>0.29</td>
<td>0.04 (14.79 %)</td>
</tr>
<tr>
<td></td>
<td>naive</td>
<td>37.69</td>
<td>33.22</td>
<td>26.28 (69.70 %)</td>
</tr>
<tr>
<td>2000</td>
<td>SDC</td>
<td>0.31</td>
<td>0.29</td>
<td>0.05 (18.03 %)</td>
</tr>
<tr>
<td></td>
<td>naive</td>
<td>72.96</td>
<td>65.55</td>
<td>52.13 (71.45 %)</td>
</tr>
</tbody>
</table>

(values in sec.)

- **Naive Engine has inappropriate performance**
  
  $\Rightarrow$ **Optimization is needed**

- **SDC-Engine exposes sophisticated performance:**
  - sufficiently fast & faster in average (300 ms.) **efficient**
  - independent of search space size **scalable**
  - significantly less time variation (ca. 14%) **stable**
Evaluation – Quantitative

SDC vs. Naive Engine (single WS Discovery)

Time Comparison Goal Instance 10
Single Web Service Discovery (Median)

Variance Comparison for Goal Instance 10
(10 available Web Services)
Further Comparison Tests

• SDC vs. Naive (all WS discovery)
  – Naive Engine: needs to inspect all available Web Services
  – SDC-Engine: measured times similar to single-WS-discovery
  ⇒ Optimization even more important for this flavor
  ⇒ SDC can ensure sophisticated performance

• SDC-full vs. SDC-light
  – Purpose: expose effects of SDC Graph on the achievable performance improvements
  – SDC-light does not include the refinement-operation
  – Observations:
    • SDC-full ca. 2 x faster than SDC-light (in average)
    • Reason: saved matchmaking operations by search space reduction
  ⇒ SDC particularly beneficial in larger applications
Evaluation – Quantitative
SDC vs. Naive Engine (all WS Discovery)

- Naive Engine needs to always inspect all available WS
- Similar observations as above
### Evaluation – Quantitative

**SDC full vs. SDC light** (without refinement)

<table>
<thead>
<tr>
<th></th>
<th>gi1</th>
<th>gi2</th>
<th>gi3</th>
<th>gi4</th>
<th>gi5</th>
<th>gi6</th>
<th>gi7</th>
<th>gi8</th>
<th>gi9</th>
<th>gi10</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>full (single)</td>
<td>141</td>
<td>578</td>
<td>875</td>
<td>31</td>
<td>359</td>
<td>516</td>
<td>602</td>
<td>47</td>
<td>62</td>
<td>47</td>
<td>326</td>
</tr>
<tr>
<td>light (single)</td>
<td>360</td>
<td>1265</td>
<td>969</td>
<td>47</td>
<td>391</td>
<td>1047</td>
<td>954</td>
<td>63</td>
<td>78</td>
<td>62</td>
<td>524</td>
</tr>
<tr>
<td>full (all)</td>
<td>375</td>
<td>875</td>
<td>937</td>
<td>532</td>
<td>750</td>
<td>1196</td>
<td>1109</td>
<td>699</td>
<td>953</td>
<td>555</td>
<td>798</td>
</tr>
<tr>
<td>light (all)</td>
<td>546.5</td>
<td>1718</td>
<td>1313</td>
<td>641</td>
<td>672</td>
<td>1290</td>
<td>1313</td>
<td>625</td>
<td>969</td>
<td>524</td>
<td>961</td>
</tr>
</tbody>
</table>

**Median** (in msec)
Evaluation – Qualitative

Estimation Model for Employment

• **Aim**: *model for minimal effort decision support on the employment of the SDC technique in any application*

• **Criteria Catalogue**:
  – Support for problem-oriented WS usage by clients desirable
  – Automated discovery is needed
    • High retrieval accuracy is desirable
    • Central and often performed operation
  – Large numbers of Goals and Web Services expected

• **Expectable Performance Increase (in %):**

\[ PIR = R_{sim} \times \left( 100 - \left( \frac{Cost_{SDC}}{Cost_{Naive}} \times 100 \right) \right) \]

where:

\[ R_{sim} \approx \frac{|G_{sim}|}{|G_{total}|} \quad \text{(Goal Similarity Ratio)} \]

\[ Cost_{SDC} \approx \frac{diam(SDC_{GG})}{2} \times \frac{b(SDC_{GG})}{2} \]

\[ Cost_{Naive} \approx \frac{|W|}{\mu(\text{usable}(W, GI(G, \beta)))} \]
Evaluation – Qualitative

Application to Verizon SOA System

• **Existing SOA System** (status January 2007)
  – 1,500 WS, 650 client applications, 15 million invocations per day
  – Web Services mostly for *central data administration*
  – System realized with *conventional WS technology*
  – Web-based registry system with *manual keyword-based search* based on Verizon Business Taxonomy

• **Challenges** reported by Verizon SOA Team
  – **Higher retrieval accuracy** for discovery
  – Increase **flexibility** & better **change management** support

• **Benefits of employing SDC technique**
  – **Significantly higher retrieval accuracy** by semantic matchmaking (descriptions extend the Verizon Business Taxonomy)
  – **Better flexibility** by goal-driven approach
  – Optimization needed, **substantial performance improvements** expectable with SDC (90% in comparison to naive engine)
Evaluation – Qualitative

SDC Graph for Verizon SOA System

- Subsumption hierarchy of Goal Templates (ca. 3500)
- Goal Instance defined on Level 1, then traverse down to most appropriate goal
- Algorithm: replace refinement step by indexing (no matchmaking)
Other Application Scenarios

• **SDC is beneficial for all application scenarios where**
  – The goal-driven approach is desirable
  – Automated discovery with high retrieval accuracy is required
  – Larger numbers of similar goals and Web Services are expected

• **Particularly relevant for more complex SWS technologies**
  *where automated discovery is an integrated & often performed operation*
  – Automated WS Composition with integrated discovery
  – Semantically enabled BPM (SUPER project)

• **Examples**
  – Travel Scenario
  – Business Process Management
  – Supply Chain Management
  – Several industries (Telco, Banking, Industrial Goods, …)
  – …
Related Work

Foundations & Complementary Works

• **Foundations**
  – **WSMO** and related efforts
  – **Goal-based approaches** from different AI disciplines
  – **Architectures for SWS**
  – **WSMO Discovery Working Group**

• **Complementary**
  *several WSMO-based works on goal-driven SWS techniques*
  – **Goal (Template) Identification**
  – **Goal Refinement** for handling not-but-nearly matches
  – **Client Interfaces** for goal-driven WS invocation
  – **WSMX**: choreography-driven WS execution
  – **WSMX**: „Concrete Goals“
  – **Selection & Ranking** on non-functional aspects
  – **Mediation** for handling data-level mismatches
Related Work

Competing Works

(1) SWS Discovery / Semantic Matchmaking
   - Widely explored research topic with significant results
   - My approach extends state-of-the-art
     • 2-phased model (design time – runtime distinction)
     • High retrieval accuracy (matchmaking of sufficiently rich functional descriptions)
     • Semantic matchmaking for goal instance level

(2) Optimization of Automated Discovery

   only received very little attention so far

   a) Classification (keyword-based categorization of WS)
      • Most common approach (e.g. UDDI extensions)
      • Low precision, not integrated with discovery => inadequate

   b) Indexing based on Functional Descriptions (EPFL 2005)
      • Significantly less expressive functional descriptions
      • Worst case performance: O(n) search time (not better than SDC)
      • Complete tree-traversal for each new request (SDC: discovery-by-lookup)
Conclusions

Strengths & Limitations

• **Strengths**
  + **Novel technique** exploiting full potential of goal-driven approach
  + **Significant performance increase** achievable for automated discovery while *sustaining high retrieval accuracy*
  + **Generic solution** adoptable to several SWS languages
  + **Substantial benefits** for several application scenarios

• **Limitations**
  - **FOL as specification language**
    • General undecidability & high computational complexity
    • Chosen to achieve generality & high expressiveness
    • Restrictions defined for Bernays-Schönfinkel fragment (NExpTime-Complete)
    • Trade-off expressiveness – complexity
  - **Implementation status**
    • Manual translation WSML -> TPTP
    • Technical integration of VAMPIRE is improvable
Conclusions

Summary & Contributions

• Novel integrated technique for automated WS discovery with high retrieval accuracy & high computational performance
  – Supports goal-driven approach
  – 2-phased Web Service discovery
  – Caching mechanism for performance optimization

• Demonstrated success & usefulness by extensive evaluation
  – Significant performance increase achievable
  – Substantial benefits for several application scenarios

• Main Contributions
  1. Revised conceptual model for goal-driven SWS approach
  2. Extensions for the state-of-the-art in semantic matchmaking for automated Web Service discovery
  3. Novel approach for optimizing automated Web Service discovery
     • Can ensure sophisticated performance
     • Suitable for integration into existing SWS environments
Relevant Publications (out of 36)


