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

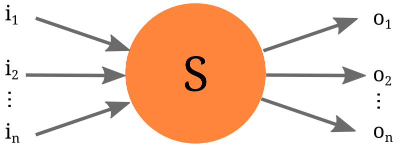
Web Services

Definition

"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)...

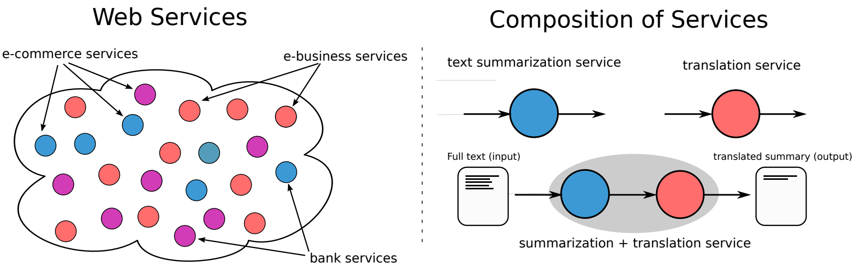
Web Services (II)

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
- ...

Composition of Web Services

A key feature of Web services is that they can be composed to create new services with new functionality by reusing the existing ones:



Problems

Web service composition is a highly complex task

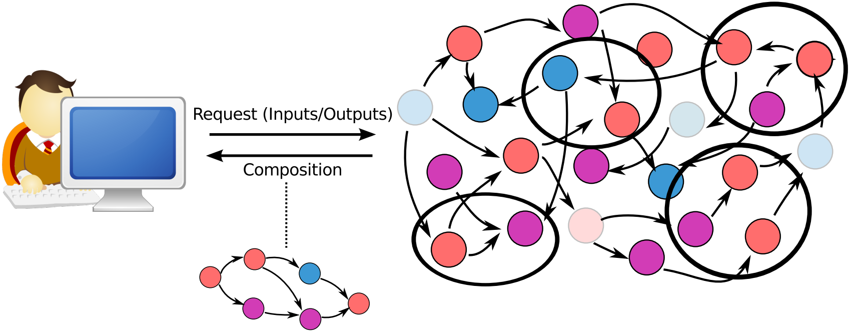
- Huge amount of Web services
- Highly dynamic nature of the Web
  - Services are constantly updated, created and destroyed
- Many possible combinations, hard to find the best one

Need for efficient **automatic composition techniques**

Automatic Composition of Web Services

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

Outline

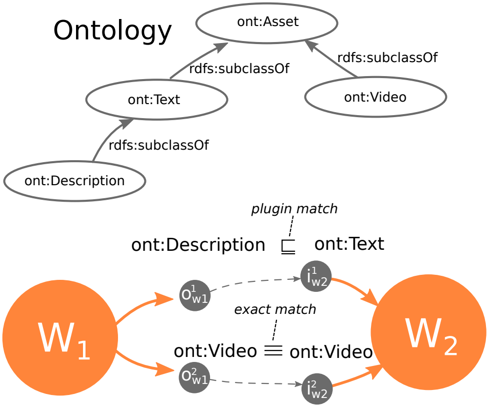
Semantic Web Services (SWS)

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:

- $In_w = \{i_1, i_2, ..., i_n\}$ is the set of required inputs
- $Out_w = \{o_1, o_2, ..., o_n\}$ is the set of generated outputs
- $In_w, Out_w \subset O$ are **concepts** from an ontology O



Semantic Matching

When can we invoke a service?

Types of match [Paolucci 2002][1]:

- **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** ($\perp$): no match between concepts

Service invokability:

- Given $C_1, C_2 \subseteq O$, we define $\otimes : O \times O \to O$ such that $C_1 \otimes C_2 = \{c_2 \in C_2 | match(c_1, c_2), c_1 \in C_1\}$
- $match(c_1, c_2)$ is true $\iff c_1 \equiv c_2 \vee c_1 \sqsubseteq c_2$
- $w = \{In_w, Out_w\}$ is invokable with a set of concepts $C \subseteq O \iff C \otimes In_w = In_w$

Semantic Composition

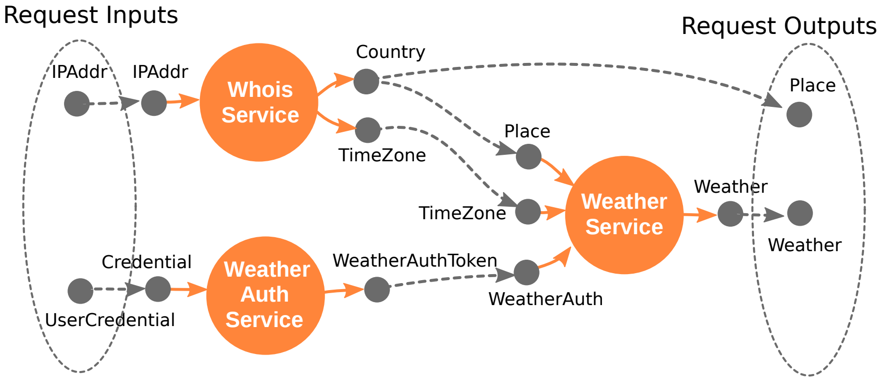
How can we model a valid composition for a request?

Given a composition request $r = \{In_r, Out_r\}$, a composite service $w_c = \{In_{w_c}, Out_{w_c}, P = \{W, \prec\}\}$ satisfies r if:

- $In_r \otimes In_{w_c} = In_{w_c}$ (invokable with the available inputs)
- $Out_{w_c} \otimes Out_r = Out_r$ (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

Composition Example

The partial order of the services in the composition can be seen as a **directed graph**.

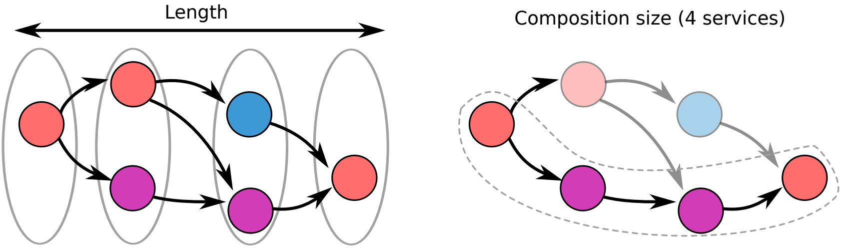


- There are many topological orderings of the services **(many ways of invoking the composition: sequence, parallel...)**

Optimizing length & services

How to generate good compositions?

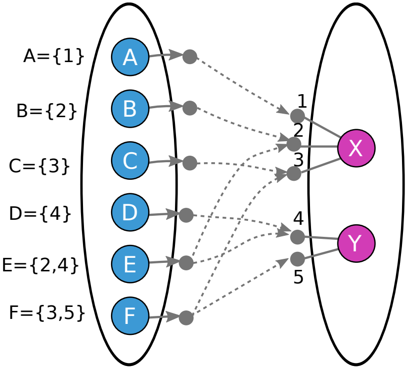
- Minimize length ⇒ maximize parallel execution
- Minimize num. of services ⇒ more interpretable & reliable solutions



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



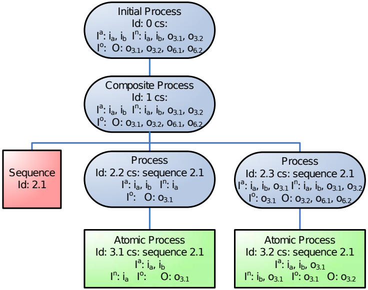
- Every instance of the SCP can be trivially represented as an instance of the Service Minimization Problem

Outline

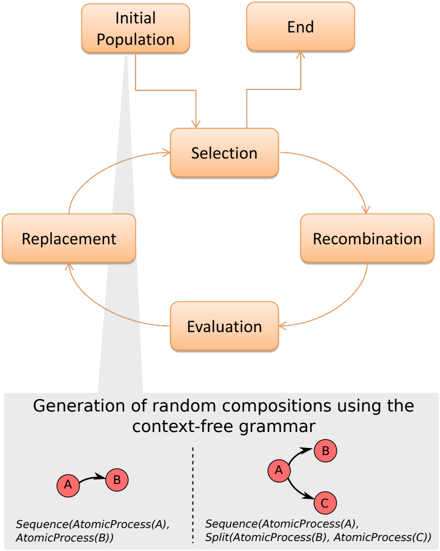
Genetic Algorithm for Automatic Composition (I)

Context-Free Grammar

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



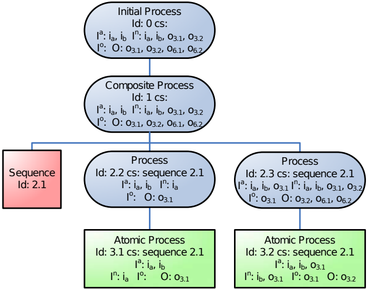
Evolutionary Approach



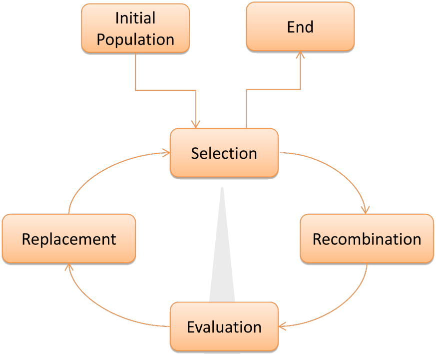
Genetic Algorithm for Automatic Composition (II)

Context-Free Grammar

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



Evolutionary Approach

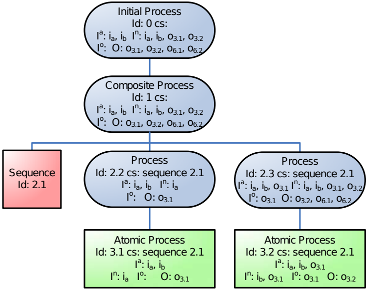


k-tournament based selection of individuals

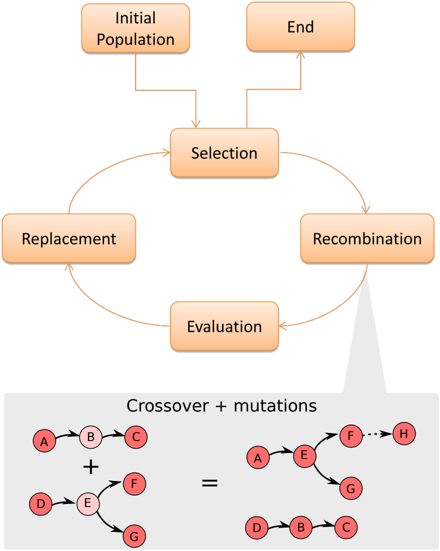
Genetic Algorithm for Automatic Composition (III)

Context-Free Grammar

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



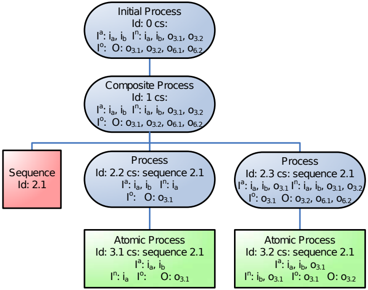
Evolutionary Approach



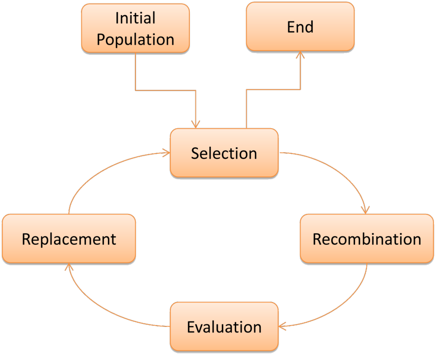
Genetic Algorithm for Automatic Composition (IV)

Context-Free Grammar

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



Evolutionary Approach



Fitness evaluation of each individual

$$fitness = \omega_1 \cdot \left(\frac{\sum_i^{|O_{obj}|} \frac{1}{DO_i+1}}{|O_{obj}|} + \frac{|I^n_{root} \cap I_{obj}|}{|I_{obj}|} \right) + \omega_2 \cdot \frac{1}{runPath} + \omega_3 \cdot \frac{1}{\#atomicProcess}$$

Inputs used
Outputs satisfied

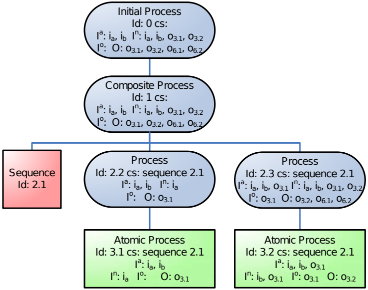
Length

Num. services

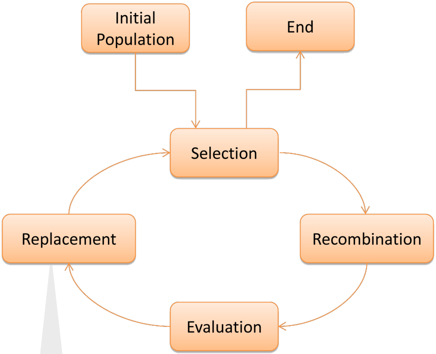
Genetic Algorithm for Automatic Composition (V)

Context-Free Grammar

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



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)

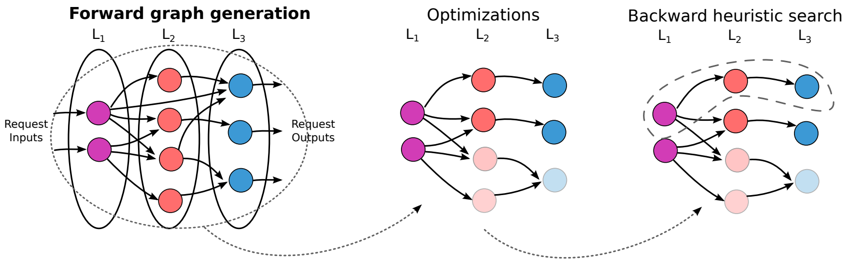
Cons

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

Contributions

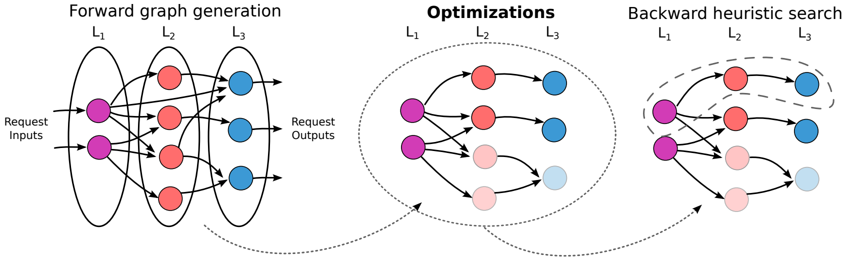
P. Rodríguez-Mier, M. Mucientes, M. Lama and M.I. Couto. Composition of web services through genetic programming. *Evolutionary Intelligence*, 3:171-186, 2010

Graph-based algorithm (I)



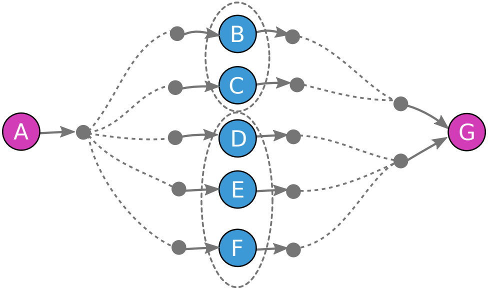
- Given a request, compute the shortest dependency graph of services that produces the expected outputs
- The graph is computed incrementally in polynomial time:
  - The first layer (L_1) contains the services that are invokable with the inputs of the request
  - The second layer (L_2) contains the services 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)



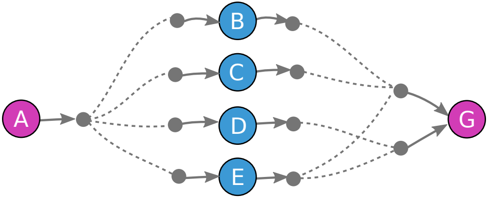
- Optimizations to prune irrelevant services
  - Remove all services that do not contribute to the output goals
- Analyze equivalence / dominance of functionality
  - Admissible state-space pruning by combining equivalent and dominated services

Graph-based algorithm (II) - Interface Equivalence



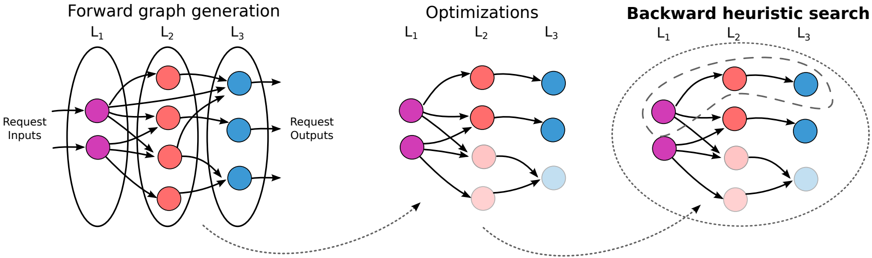
- There are 6 different compositions: $\{B, C\} \times \{D, E, F\}$ but:
  - B and C are functionally (I/O) equivalent
  - C, D and F are also functionally (I/O) equivalent
- We can merge both groups of services to end with just one composition: *Sequence(A, Split(Choice(B,C), Choice(D,E,F), G)*.

Graph-based algorithm (II) - Interface Dominance



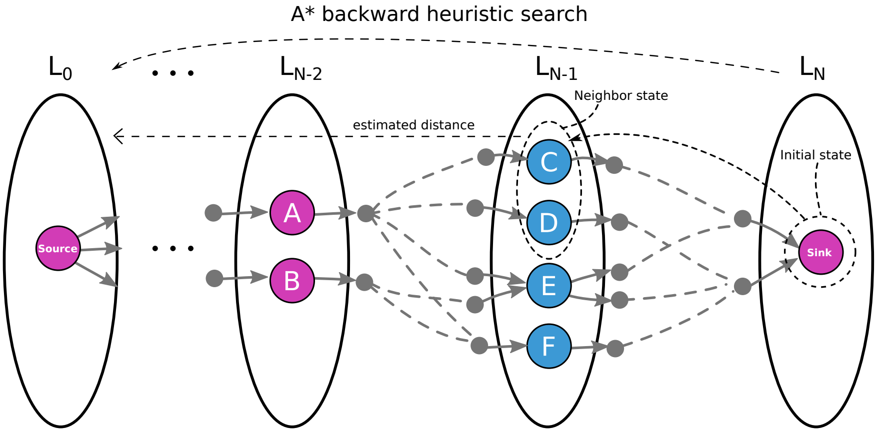
- Service E dominates B, C and D:
  - It only needs A to solve its inputs
  - It resolves all the inputs of service G
  - Any other combination of services is redundant, i.e., leads to a composition with more services and same functionality.

Graph-based algorithm (III)



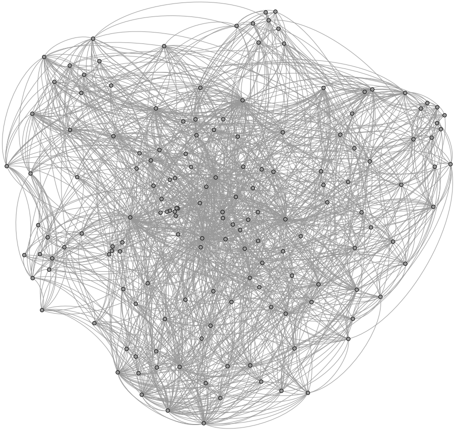
- Backward heuristic A* algorithm to extract the optimal composition subgraph from the graph
- The algorithm starts searching from the last layer L_N until it reaches L_1
- Heuristic based cost function $f(x) = g(x) + h(x)$ where
  - $g(x) =$ *number of different services selected*
  - $h(x) =$ *distance from the current layer to* L_1 (consistent heuristic)

Graph-based algorithm (IV) - A* search example



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)

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

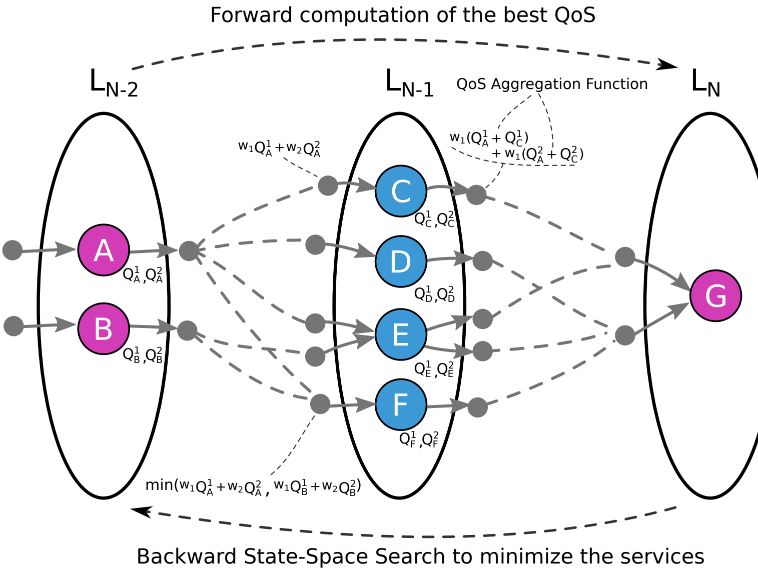
Main contributions

- P. Rodríguez-Mier, M. Mucientes and M. Lama. Web Service Composition with a Heuristic-based Search Algorithm. In IEEE ICWS, pages 81–88, Washington DC (USA), 2011 **(CORE-A 14% acceptance)**
- P. Rodríguez-Mier, M. Mucientes, J.C. Vidal, and M. Lama. An Optimal and Complete Algorithm for Automatic Web Service Composition. IJWSR, 9(2):1-20, 2012 **(JCR)**

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



QoS-Driven Automatic Composition - Evaluation

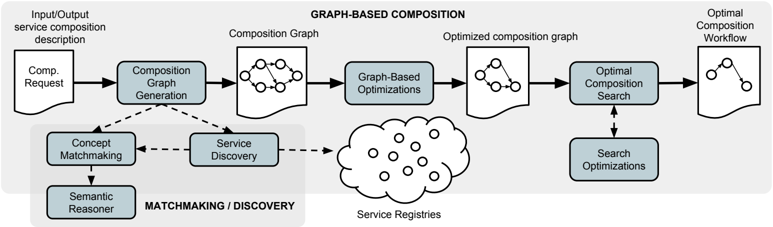
- We have validated the algorithm using the 5 repositories of the Web Service Challenge 2009
- We found shorter solutions in datasets 4 and 5

Dataset	Optimal QoS solution					
WSC-2009'01	w1/w2	#I. Serv.	#S. (LM/GM)	Rt.(LM/GM)	Th.(LM/GM)	Time (ms) (LM/GM)
	1.0/0.0	13	5/5	500/500	3000/3000	274/389
	0.5/0.5	7	5/5	760/760	15000/15000	277/291
	0.0/1.0	7	5/5	930/930	15000/15000	270/298
WSC-2009'02	1.0/0.0	25	20/20	1690/1690	3000/2000	868/1988
	0.5/0.5	24	20/20	1800/1770	6000/6000	860/3103
	0.0/1.0	24	20/20	1970/2000	6000/6000	117/7530
WSC-2009'03	1.0/0.0	11	10/10	760/760	2000/4000	1071/1545
	0.5/0.5	33	10/10	840/760	4000/4000	1069/1533
	0.0/1.0	31	18/11	1780/1110	4000/4000	1101/5249
WSC-2009'04	1.0/0.0	50	40/-	1470/-	2000/-	4399/-
	0.5/0.5	73	64/-	3540/-	4000/-	4586/-
	0.0/1.0	72	62/-	3840/-	4000/-	4506/-
WSC-2009'05	1.0/0.0	41	32/32	4070/4070	1000/1000	2646/2801
	0.5/0.5	41	32/32	4280/4200	4000/4000	2667/2680
	0.0/1.0	41	32/30	5470/4750	4000/4000	2657/10953

Main contributions

- P. Rodríguez-Mier, M. Mucientes and M. Lama. A Dynamic QoS-Aware Semantic Web Service Composition Algorithm. In Proceedings of the 10th International Conference on Service-Oriented Computing (ICSOC), pages 623-630, Shanghai (China), 2012 **(CORE-A)**

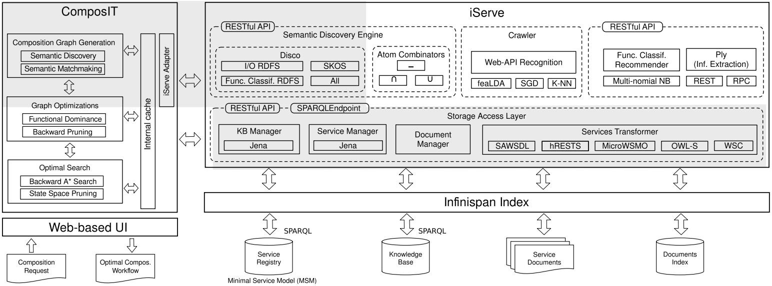
Integrated Framework & Architecture (I)



- 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

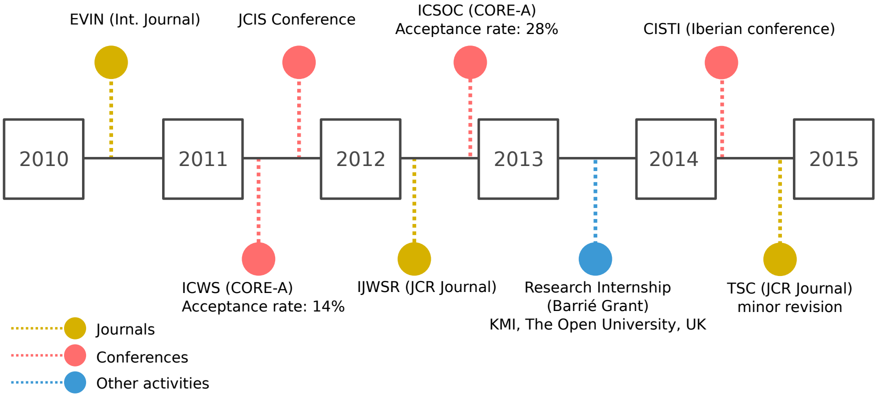
Integrated Framework & Architecture (II)

- Reference implementation:
  - ComposIT: graph-based composition algorithm developed in this thesis (`http://github.com/citiususc/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



Thank you!

Questions? :-)

pablo.rodriguez.mier@usc.es