HAL
open science

A Framework for Discovering and Automatically Composing Services

M.-C Fauvet, I.-B Caicedo-Castro, P Na-Lumpoon, Ahmed Lbath

To cite this version:

M.-C Fauvet, I.-B Caicedo-Castro, P Na-Lumpoon, Ahmed Lbath. A Framework for Discovering and Automatically Composing Services. ICSOC - demo session, 2015, GOA, India. hal-01888343

HAL Id: hal-01888343

https://hal.science/hal-01888343v1

Submitted on 5 Oct 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Framework for Discovering and Automatically Composing Services

M.-C. Fauvet[1] I.-B. Caicedo-Castro[1,2,3], P. Na-Lumpoon[1], and A. Lbath[1]

[1] Univ. Grenoble Alpes, LIG (MRIM), F-38000 Grenoble, France
[2] University of Córdoba, Colombia
[3] National University of Colombia
Marie-Christine.Fauvet@imag.fr

Abstract. The study described in this paper is a part of a broader project on E-Tourism that aims at providing mobile users with context-aware personalised services. This paper focuses on issues raised by service discovery and automated composition. Given mobile users' requirement (e.g., buying airplane tickets, booking a hotel room, renting a car, etc.) expressed in a free-text query, the framework we describe in the paper deals with discovery and composition services to be executed in order to fulfil users' needs. First, behind the service discovery module is the idea to match the users' queries with a set of documents in a corpus such that each document contains a service description. To achieve the design of this module we have proposed a family of IR models. Second the module which is responsible with on-the-fly composition of the operations of the retrieved services is based on AI planning and produces an executable business process model whose each task correspond to a call to a service operation. The whole framework has been partially evaluated: we give in the paper experimental evaluations that show that our discovery model, based on query expansion via a co-occurrence thesaurus, outperforms the effectiveness of all models we have reviewed.

Keywords: Web Service Discovery, Information Retrieval, Mobile Computing, AI Planning, Service Automated Composition, BPM;

1 Introduction and background

Recently distributed computing systems based on context awareness have been proposed in several domains such as healthcare, logistics and tourism. The study described in this paper is conducted as a part of a broader project whose main goal is to design and implement a software system which provides mobile users with services according to their needs [15, 14]. Figure 1 sketches the overall architecture of this project. The role of each module and the flow of information are detailed further in the text.

Here we consider the following definitions for context, profile and service concepts. A *context* includes spatial, temporal, physical, and environmental properties that could be collected by sensors embedded on the devices used to submit

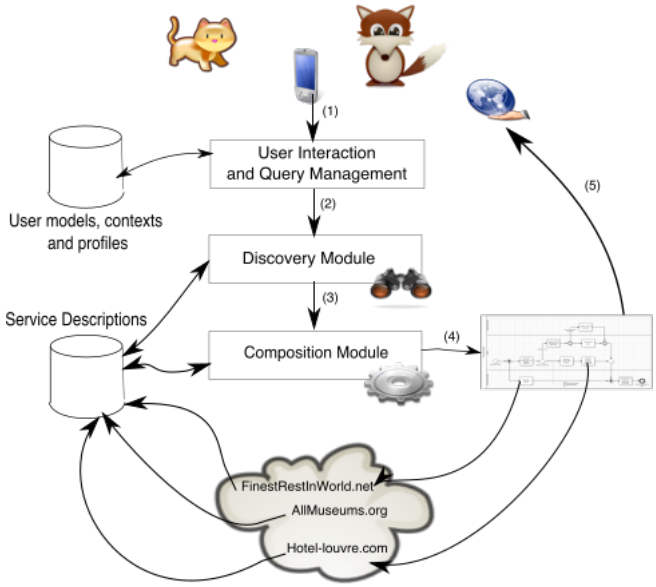

Fig. 1. A context-aware discovering system for mobile users

the queries. Such properties are for example: GPS location, timestamp, external temperature, screen size, etc. A *profile* captures users' personal details, preferences and centers of interest. For instance, a profile could contain information about users' age, citizenship, gender, occupation, favorite recreational activities, etc.

We adopt the definition of service given by the W3C [25]: *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL*[4]*). Other systems interact with the web service in a manner prescribed by its description using SOAP*[5] *messages, typically conveyed using HTTP with an XML serialisation in conjunction with other Web-related standards.*

A problem generally associated with the Service Description is the one of their publication and discovery. A proposed standard to tackle this problem is UDDI[6]. UDDI defines a programming interface to publish service descriptions in dedicated repositories, submit keyword-based queries, and navigate through the service descriptions obtained through these queries. UDDI was specifically designed for WSDL and SOAP. It was often recommended that people use it at runtime to dynamically look up services. Nevertheless, the main drawback

[4] Web Service Description Language, http://www.w3.org/TR/wsdl
[5] Simple Object Access Protocol
[6] Universal Description Discovery and Integration, http://uddi.xml.org/

is that people easily end up with a service not wanted, whether it's because the functionality is slightly different, the service is slower, or it lacks redundant backups. For these reasons UDDI was given up by the community.

This issue of service discovery has been dealt with information retrieval models, considering a corpus composed by WSDL documents. A WSDL document contains a syntactically-based description including service name, operations name and signature, and some descriptions in natural language that are commentaries written by programmers.

// add some glue with the rest

The system whose architecture is sketched in Figure 1, is accessible by registered users through a web browser installed on their mobile device (see Figure 1, the data flow (1)). This system is built on the top of four components:

To be actualise from your dissertations, but skip the part dedicated to privacy

1. User interaction and query management module aims at managing user connections and getting queries submitted by users and sent using their mobile device.
2. The module User management System is responsible for managing users' context and profile with respect of their privacy.
3. Discovery system: given the user's query, her profile and context, what are the services, discovered among a repository of services, that once composed can potentially meet the user's needs expressed by queries?
   This paper focuses only on the issues raised by this question.
4. Eventually the module Composition and orchestration system is in charge of automatically orchestrate and execute services returned by the discovery phase.

The rest of this paper is outlined as follows: we motivate and illustrate this work in Section 2, then Section 3 presents related work and highlights the lacks of existing main approaches. While Section 4 details the approach, and discusses the models implemented in the proposed framework and also compares them with the related work. Finally, Section **??** concludes the paper and sketches some further work.

2 Motivating example

Alice is an American tourist visiting Paris in France. She has forgotten to book a room. Thus, she picks up her smartphone and accesses the system above mentioned (see Figure 1), and submits the query: I want to book a room for 3 nights from tonight. Moreover, the system captures Alice's context information, namely: coordinates = "48.2167° N, 2.3332° E" and date = "1/06/2014". Besides, the system has the following information about Alice's profile: name = "Alice", citizenship="USA", travelPurpose="tourism", gender="female".

Thereafter, S^2niffer searches services for booking a room. It gives to the Composition and Orchestration System (COS) a ranked list of candidate services for booking rooms in hotels. The service which has the highest rank in this list

contains the following operation: BookARoom, this operation receives as parameters the name of the hotel, the number of nights and persons who shall stay in the room, the date and time to check-in, and the user's name and telephone number. As a result, the operation returns a confirmation whether the room has been booked or not.

COS executes this operation, and Alice successfully book a room in *Novotel hotel*. The Alice's context information is used by the COS to execute services.

At 4 PM, she wants to book a table at the finest restaurant in the city, and the direction to get there. Once again, she uses the same system and submits the query: I want to book a table for 2 people at the finest restaurant in the city, and the direction to the restaurant. At this time, Alice's profile has not been changed, and the system captures the following Alice's context information: coordinates = "48.8567° N, 2.3508° E" and date = "1/06/2014".

This query has two requirements, then the User Interaction and Query Management (UIQM) module splits this query in two subqueries. Therefore, the first query submitted to S^2niffer is I want to book a table at the finest restaurant in the city. The second query submitted to S^2niffer is the direction to the restaurant. S^2niffer shall send to the COS two ranked lists of candidate services, which correspond with each subquery. From the list of candidate services that may fulfil the first subquery, the one which has the highest rank contains the following operations:

- FindFinestRestaurant: This operation receives as a parameter the name of the city where the user is looking for the finest restaurant. The operation returns the name and the address of the restaurant.
- BookRestaurant: This operation receives as parameters the restaurant name, the number of persons, and the user's name and telephone number. As a result, the operation returns a confirmation whether the table has been booked or not.

In the another ranked list of candidate services that fulfils the second subquery, the one which has the highest rank contains the following operations:

- FromCoordinatesToCity: Given the geographical coordinates, this operation returns the name of the city where is allocated the coordinates of certain point of interest.
- CoordinatesFromAddress: Given an address, this operation returns its geographical coordinates.
- GetDirection: This operation provides instructions on how to reach a destination. This operation receives two parameters, the coordinates of the starting point, and the coordinates of the destination.

COS takes the operations of both services and compose them. The execution of the resulting composite service fulfils both Alice's needs (i.e., booking a table in the finest restaurant of the city, and knowing the direction to go there).

On that night, while she is enjoying a delightful dinner in *Le Meurice* restaurant, Alice is wondering about the weather in the next day. She needs this information to decide whether she will go to *Louvre museum* or *Euro Disney*. One

more time, she uses the system and submits the following query: I want to buy a ticket for Euro Disney tomorrow if the weather forecast is sunny, otherwise, buy a ticket for Louvre museum. At this time, Alice's profile is still the same, however, her new context information is as follows: coordinates = "48.8651° N, 2.3280° E" and date = "1/06/2014".

Similar to the previous query, this one contains three requirements, therefore the UIQM module splits the query in three subqueries. The first subquery is to buy a ticket for Euro Disney tomorrow. The second subquery is the weather forecast is sunny. The last subquery is buy a ticket for Louvre museum. All three subqueries are sent to S^2niffer, thereby it sends three lists of services to the COS. From the list of candidate services that may fulfil the first subquery, the one which has the highest rank contains the following operation: BuyTickets4EuroDisney, this operation receives as parameters the name of the customer, the number of required tickets, information of a credit card, etc. As a result, the operation returns a confirmation whether the transaction has been successfully finished or not.

In the another ranked list of candidate services that may fulfil the second subquery, the one which has the highest rank contains the following operation: GetWeatherForecast, this operation returns the weather for a given city of a certain country, and for a given date.

In the ranked list of candidate services that may fulfil the last subquery, the one which has the highest rank contains the following operation: BuyTickets4LouvreMuseum, this operation receives similar operation as the one to by tickets for *Euro Disney*, besides, the result of this operation is the same.

In the same fashion as before, the COS composes all operations of previous services. The execution of the resulting composite service fulfils Alice's requirements regarding her condition.

With the above mentioned system, users are able to consume services accessible on the Internet, from their mobile devices. Besides, service providers do not need to produce front-end applications, which serve as interfaces to access their services. This thesis addresses the problem of searching services for fulfilling specific users' requirements.

3 Related Work

The section covers related work related to two different domain: (1) AI planning and service composition, and (2) Information Retrieval based service discovery.

3.1 AI Planning and Service Composition

From Pathathai ICSOC'14 paper and dissertation

The text below is the one from ICSOC'14 paper...

OWL-S is an OWL based ontology for describing Semantic Web Services. It will enable users and software agents to automatically discover, invoke, compose and monitor web resources offering services under specified constraints [12]. To

facilitate the OWL-S capacities mentioned, OWL-S organizes the service structure into three parts which are service profile, process model and service grounding. The service profile part is used to describe what the service does, which includes the information such as the service name and description, quality of service, publisher and contact information. The process model part describes the types of service process and the elements (a set of inputs, outputs, preconditions, effects of the service execution) inside each process. There are three types of service process which are atomic, composite and simple processes. Atomic process is process that runs completely independently of any other process. Composite process is process that requires multiple actions from other process, in which directed by one of control constructs such as sequence, iterate, choice and if-then-else. While the simple processes provide an abstraction mechanism to provide multiple views of the same process [12]. Finally, the service grounding specifies the interaction information with the service such as communication protocols, message formats and port number. We choose OWL-S as service description for our web service composition framework. Since OWL-S also works with operations and each operation has properties inputs, output, preconditions and effects optionally in which similar to operator in AI planning domain. And the reason why we select OWL-S over other semantic web services like SAWSDL and WSDL-S is that OWL-S can support control contracts in the composite process.

Because most of the time user requirement can be complex that contains control construct statements. Additional, OWL-S supports other OWL ontology used as referred data type and is dominated to many researched works. Thus many OWL-S supporting tools are available.

SAWSDL stands for Semantic Annotations for WSDL and XML schema in which is a W3C Recommendation defines it as mechanisms using which semantic annotations can be added to WSDL components [7]. SAWSDL standard solves problems regarding to data heterogeneity in the web services description language (WSDL 2.0) [5]. Two WSDL services can have similar descriptions while meaning totally different things, or they can have very different descriptions yet similar meaning. Therefore, SAWDL provides mechanisms by which concepts from the semantic models that are defined either within or outside the WSDL document can be referenced from within WSDL components as annotations. These semantics when expressed in formal languages can help disambiguate the description of Web services during automatic discovery and composition of the Web services [7]. In additional, several project and application utilizing SAWDL for Semantic Web Services automation [26]. We choose SAWSDL as service description for our web service composition framework. We can apply SAWSDL into AI planning domain since SAWSDL can recognize inputs, outputs, preconditions and effects of service operation in which is similar to that operator in AI planning domain. And with modelReference attribute in SAWSDL, we can model stateful service by adding input rule and output rule constraints related to persisting state among operations.

3.2 Related Work

Several automatic web service composition frameworks toward AI have beeb reviewed. We have analyzed the similarity and difference of techniques among them in this section. One of classical previous works on automatic web service composition is **SHOP2** [23]. The objective of SHOP2 is to synthesize a plan which is a sequence of primitive operators. Using HTN (Hierarchical Task Network) technique, SHOP2 Planner recursively decompose a given task until it reaches to primitive operators. The framework transforms semantic web service OWL-S [12] representing atomic and composite web services into operators in SHOP2 domain. However, SHOP2 dose not take into account of data heterogeneous and stateful services.

Two frameworks we reviewed use semantic web service SAWSDL to represent service markup in automatic service composition. **METHOR-S** [32] is a planning-based approach to solve both the process heterogeneity and data heterogeneity. The approach uses service template to model the service requirement in STRIPS which is the formal planning language, and then discover the SAWSDL services with profile that matches the defined abstract process in service template. The output of the system is an executable BPEL. However, this approach is not fully automated web service composition. Another framework is **Haley** [33] which is a hierarchical framework for logical composition of web services. Haley offers a method to exploit hierarchical decomposition of web service composition problem and then utilizes technique of decision theoretic planning on first order sentences of SAWSDL web services. This technique helps to solve problems on the uncertainty inherent in web service invocation and also provide an expected cost-based optimization. However, all services used in Haley are just stateless services.

[2] proposes a planning under uncertainty framework for the automated composition for web services, which can handle stateful web services using industrial standard like WS-BPEL. Thus, this approach support ability to model and solve planning problems for asynchronous domains. The goal is extracted from the requirement and then all the relevant abstract WS-BPEL are translated into state transition system to automatically be composed into composited process. Even though this approach can handle stateful services, but the issues of process heterogeneity and data heterogeneity are not discussed.

Two of previous works that are similar to our approach are [28] and [1]. Both of them work on automatic generation of an abstract composition based on the formalism of fluent calculus. [28] transforms request service and relevant web services from OWL-S format into fluent calculus domain and use FLUX constraint logic programming method to generate a plan. The plan output of the system is the sequence of fluent calculus operator. The algorithm to transform OWL-S web service into fluent calculus is proposed. However, the algorithm dose not cover transformation of stateful service into fluent calculus. While [1] proposes the new Web Service Specification Language (WSSL) based on fluent calculus for facilitating automatic web service discovery and composition process. The framework is implemented using FLUX-based planning, supporting compositions with fun-

damental control constructs such as conditionals and loops. However, WSSL is not linked with other external ontology. This may cause the problem of data heterogeneous.

3.3 Information Retrieval-based Service Discovery

Needs to be actualise, and a lot synthetise

In several works, researchers have applied information retrieval concepts in order to cope the web service discovery challenge. Typically, these existing approaches rely on either:

1. Vector space model or,
2. Latent Semantic Indexing, or
3. Hybrid models based on ontologies and information retrieval.

The VSM has been applied in many approaches (see for example: [27, 20, 9, 10, 6, 29, **?**]). In these works, a set of WSDL documents composes the collection of service descriptions. Some of these approaches does not tackle the term mismatch problems [20, 10, 6, 29, **?**]. Whereas, these problems have been addressed by expanding queries and WSDL documents with synonyms of their terms [27, 9]. Synonyms are extracted from WordNet lexicon [13]. Nevertheless, the query expansion based on the injection of synonyms significantly decreases precision because a term may have synonyms with different meanings depending of the context of the term in the query.

In other approaches, researchers applied LSI to cope term mismatch problems in the context of service discovery [21]. However, factorising a matrix through SVD causes scalability issues in LSI. Therefore, other works handled this shortcoming instead of aiming to increase the effectiveness of LSI [11, 30, 31]. However, scalability issues are out the scope of our research.

LDA is another model based on latent factors in text documents. In this model the latent factors are topics, and their distribution is assumed to have Dirichlet prior. This model is applied to discover the latent topics from concepts contained in service descriptions written in OWL-S [3, 4]. According to the results obtained in this research, LDA outperforms Probabilistic LSI (PLSI) [3, 4]. Nonetheless, in the same study LDA has not been compared with other models used in prior research (e.g., LSI).

Another direction to deal with term mismatch problems is to use ontologies. Therefore, several works combine LSI and ontologies [17, 19, 18]. An ontology is used as a vocabulary to expand the query [18]. In another hybrid approach K–means algorithm is used to divide the corpus in several clusters of documents [19]. Thereafter, given a query, SVD is applied on the most similar cluster (similar to [11]). However, this technique is complemented with a semantic–based matching, which is implemented on an ontology of services, by computing the similarity between service input and output parameters. At the end of the procedure, services are ranked according two both techniques.

The drawback of such ontology-based approach is that the human intervention is necessary, as ontologies must be built with the assistance of human

experts of the domain. Therefore, the creation of ontologies is an expensive, time-consuming, tedious, and error-prone task [8, 22]. This is why we have decided not to design and build any ontology.

In the discovery system we have implemented a model based on query expansion via co-occurrence thesaurus. In Section 4.2 we describe this model and present the outcomes of an empirical evaluation conducted to compare our model with those in the state-of-the-art.

4 A multi-layer framework...

4.1 Multi-layer Architecture

We propose a multi-layer architecture which comprise of multiple layers (see in Figure 2). From the top layer, relevant services to fulfill users' needs can be ac-

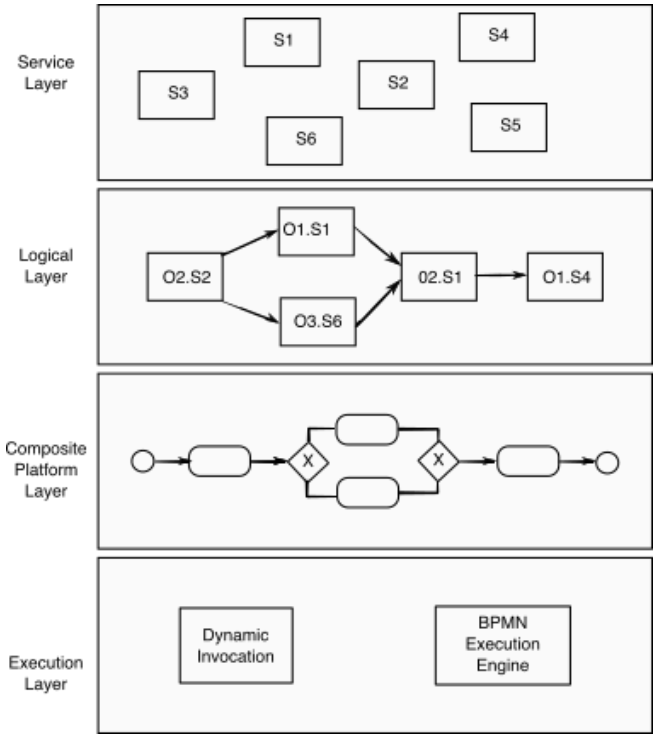

Fig. 2. Multi-layer for service composition and execution system

cessed from service providers in the *Service layer*. In this work, we consider web services, which explain operation(s) with an invocation interface to call these op-

erations. For example, restaurant service offering two operations[7] searchRestaurant:city,type $\mapsto$ restaurantName and bookTable:restaurantName,guestName $\mapsto$ confirmationBooking. The next is *Logical layer*, which contains a logical composition of operations. The operations from the service layers are assembled into a workflow against user's goals.

For example, the sequence of calls to searchRestaurant then bookTable is satisfied against confirmationBooking. However, the generated workflow is not in an executable form, this is why we perform the transformation of the workflow into an executable business process in the *Composition platform layer*. The last layer is the *Execution layer*, which handles runtime activities such as binding services, acquiring missing parameters and enacting business process engine.

4.2 Service Discovery

In our approach we propose to automatically generate a thesaurus by computing the Terms Similarity Matrix $\mathbf{C} = \mathbf{Y}\mathbf{Y}^T$, where $\mathbf{C} \in \Re^{m \times m}$ and each component $\mathbf{C}_{ij}$ represents the similarity score between terms t_i and t_j. Thereafter, the latent factors of each column vector of this matrix are computed, by factorising it through the above mentioned methods in order to obtain $\mathbf{W} \in \Re^{r \times m}$ and $\mathbf{X} \in \Re^{r \times m}$ such that $\mathbf{C} = \mathbf{W}^T\mathbf{X}$. Let Q be a set of terms used in a query, each term $t_i \notin Q$ of the thesaurus is added to the query if $sim(\mathbf{x}_i, \mathbf{x}_j) > \theta$ (see Equation **??**), where $t_j \in Q$. The parameter θ is estimated by means of experiments. We carried out experiments using the fourth version of the OWL-S service retrieval test collection named OWLS-TC4[8] which contains the descriptions of 1083 Web services from 9 domains (i.e., education, medical care, food, travel, communication, economy, weapon, geography, and simulation). Each description is given in OWL-S 1.1. This collection includes 42 queries associated with their relevance judgment provided by several users. A pooling strategy (as used in TREC[9]) was conducted to collect the relevance judgment set which was obtained from the top-100 results of participants of the S3 contest[10] in 2008. The judgment relevance has been graded in four different levels, i.e., highly relevant (value 3), relevant (value 2), potentially relevant (value 1), and non-relevant (value 0). Therefore, during the experiments the *Normalised Discounted Cumulative Gain at 10* (NDCG@10) has been used instead of the *Mean Average Precision* (MAP) to measure the overall ranking effectiveness of each approach.

This collection is the unique one which exists in service retrieval domain which has judgment relevance. Previous versions of this collection were used for carrying out experiments in related recent works [3, 4].

Table 1 presents the results we obtained from the experiments. The three first rows show the retrieval effectiveness we obtained for existing techniques

[7] Parameters after colon refer to operation inputs while after arrow are for operation outputs.

[8] OWL-S Service Retrieval Test Collection, projects.semwebcentral.org/projects/owls-tc/

[9] Text Retrieval Conference, trec.nist.gov/

[10] Semantic Service Selection, www-ags.dfki.uni-sb.de/ klusch/s3/

(VSM, LDA and LSI). Then the three last rows show results for our model extensions: *Query Expansion via a Co-Occurrence Thesaurus* (QECOT) automatically generated through SVD is called QECOT-SVD, QECOT generated using the method MSE, QECOT-MSE, QECOT generated through NMF, QECOT-NMF.

The results we got suggest that QECOT-MSE outperforms all the models studied in the paper. Indeed, Table 2 shows that the effectiveness of QECOT-MSE is better than LDA and VSM, which are the models applied for service retrieval in [27, 20, 9, 10, 6, 29, 3, 4], and the difference is statistically significant. Despite the difference between the effectiveness of QECOT-MSE and LSI-SVD is not statistically significant, the first model outperformed the second one in more queries. Indeed, in 5 queries both models had the same effectiveness, in 24 queries QECOT-MSE outperformed LSI-SVD, and only in 13 queries LSI-SVD has better effectiveness than QECOT-MSE (see Figure 3). Figure 3 depicts a comparison of the effectiveness of both models regarding each query used in the experiments. Points below the diagonal line correspond with queries where LSI-SVD outperformed QECOT-MSE (13 points). Whereas points above the line correspond with queries where QECOT-MSE outperformed LSI-SVD (24 points). Finally, points in the diagonal line correspond with queries where both models had the same effectiveness (5 points).

4.3 Automated Composition of Service Operations

From Pathathai's dissertation...

We separate the Abstract Service Composition system into two components: Transformer and FLUX Planner as depicted in Figure 4. The Transformer component is responsible for pre-processing inputs from problem domain into the user's planning domain. While FLUX Planner component is a service operation composer, which performs assembling service operations to get the desired abstract plan.

The user's query along with her profile, context and service operations in problem domain shall be considered as initial, goal states and possible actions in FLUX query respectively. The FLUX query shall be solved by the FLUX planner. We use AI-planning techniques to implement our FLUX planner to perform automated abstract service composition.

The reason we chose FLUX over other AI systems because FLUX is implemented based on fluent calculus[11]. Modeling user's constraints and relevant operations with fluent calculus, the FLUX Planner reasons on a variety of operations such nondeterministic, conditional and concurrency to obtain a resulted abstract plan.

Transformer to fluent calculus The Transformer is the process in the Abstract Service Composition system. It intents to pre-process data from user problem domain into initial knowledge state and constraints in planning domain. The

[11] A formal mathematic expression for dynamical domain in first order logic.

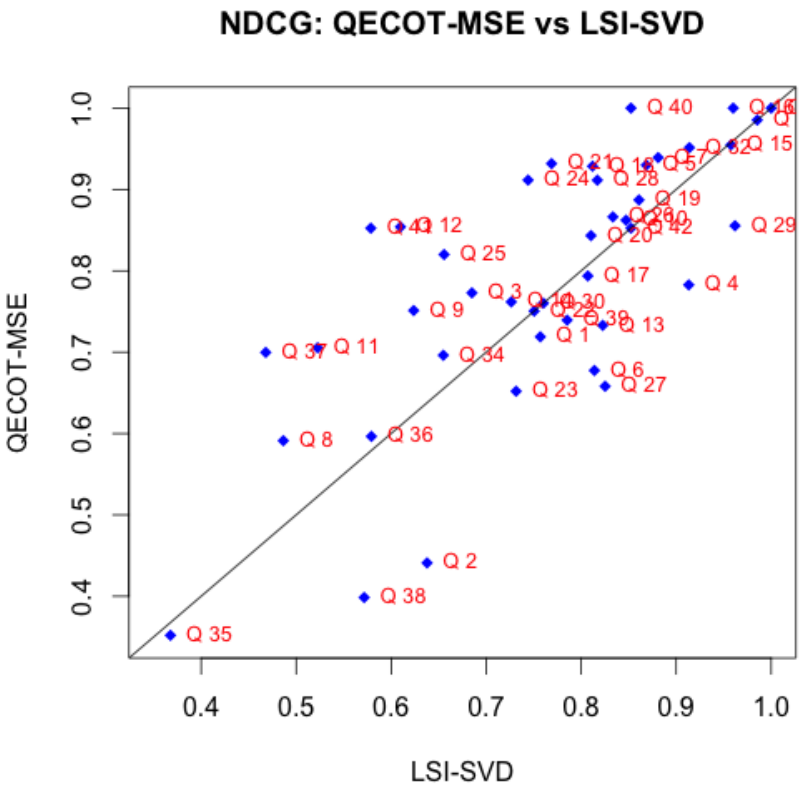

Fig. 3. Comparison between QECOT-MSE and LSI-SVD

data from user problem domain consists of parameters extracted from the user's query, the profile, the context and operations offered by service descriptions. In Figure 5, the transformer parses the parameters and the service operations and then transforms them into initial and goal states and operations in fluent calculus axioms respectively. Their converted results are combined for a FLUX query. The Flux query is then forwarded to FLUX planner. The following section presents the transformer process with the motivating example (see Section 2), which is subjected to user requirements and service operations mappings.

User requirements mapping This subsection shows what are the user's requirements and the parameters of the user's goals and her profile and context, are transformed to initial and goal states. Consider user requirements mapping in Table 3 as an example:

The parameters classified into query, context and profile categories are shown in user problem domain column. We use the convention for each parameter as name of parameter followed by its value beside parenthesis. The value itself

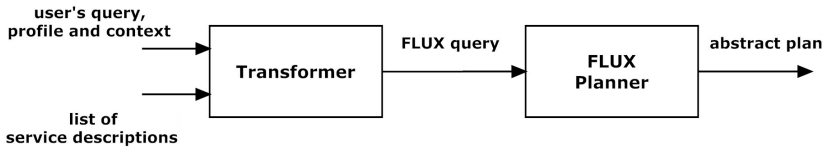

Fig. 4. Abstract Service Composition process

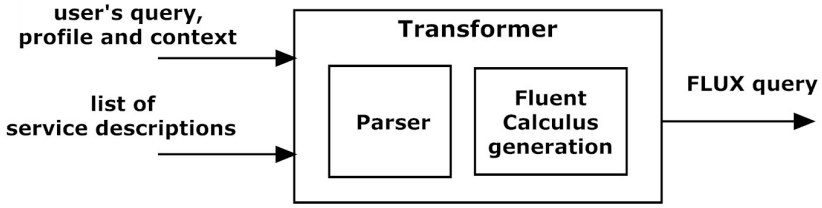

Fig. 5. Transformer process

has two options: known value or goal value. The known values are extracted from user queries, disclosed context and profile information. For example, NumberOfGuest(2), Time(8pm), Date(01/06/2014). The goal values are the unknown values of things the users want to possess or achieve. For instance, RestaurantName(goal), BookingReservation(goal), Direction(goal).

To map parameters from problem domain to fluents in planning domain, we shall follow these two rules: 1) Parameters having the goal values are grouped into a list of goals fluent. For instance, goals([RestaurantName, BookingReservation, Direction]). 2) Individual parameter having known values is converted to initial fluent. For example, initial(NumberOfGuest).

Service operations mapping Besides user requirements mapping, the transformer converts service operations into input fluent operations. Each operation has its naming conventions of *op_inputs* and *op_outputs* clauses. For each service operation, we have only one clause *op_inputs* and one clause *op_outputs*. Generally, *op_inputs* and *op_outputs* refer a set of operation input and a set of operation outputs. Consider service operation mapping in Table 4 as an example:

It's a worth noting that we separate input and output operations into different naming conventions. It enables the Planner to synthesize the control links among operations in the abstract plan. More details about the Planner are given in the next section.

FLUX Planner Planner is a process happening after the transformer process in the Abstract Service Composition system. It intents to synthesize a plan from abstract operations to fulfill the user's goals. The outcome of this resulting plan should satisfy all expected goals and initial constraints and also support linked control constructs such as sequence, condition and parallel.

To achieve this operation, we propose FLUX planner, which is a constraint programming based on fluent calculus see its architecture depicted in Figure 6. The FLUX planner consists of three components: (1) FLUX library, (2) FLUX query and abstract plan and (3) service composition agent.

The FLUX library contains a set of constraint handling rules and a constraint solver for finite domain of our FLUX planner [24]. The FLUX query contains encodings of the domain axioms including condition constraint, initial and goal knowledge state and a list of service operations, while the abstract plan contains encoding of the plan axioms including initial and goal nodes, flow of composable operations and data transformation. The service composition agent performs agent's actions including action precondition axiom and update axioms, under the behavior of planning composition.

FLUX query and Abstract plan We develop planning model for the service composition agent. The model is formulated using fluents for the FLUX query and the abstract plan structures. We use the special sorts `OPERATION, PARAMETER, VALUE` and `INDEX` along with the fluents. One instance of FLUX query necessarily consists of following basic fluents:

```
Initial(X): PARAMETER --> FLUENT
Goal(Xs): PARAMETER --> FLUENT
Op_inputs(X,Ys): OPERATION x SET OF PARAMETER --> FLUENT
Op_outputs(X,Ys): OPERATION x SET OF PARAMETER --> FLUENT
```

`Initial(X)` is a fluent for initial parameter `X` (i.e., *initial*(RestaurantType) and *initial*(Coordinates)). `Goal(Xs)` is a fluent for a set of goal paramaters `Xs` (i.e., *goals([RestaurantName, BookingReservation, Direction])*). `Op_inputs(X,Ys)` is a fluent for a set of input parameters `Ys` of a single operation `X` (i.e., *op_inputs(FromCoordinatesToCity, [Coordinates])*) and `Op_outputs(X,Ys)` is a fluent for a set of output parameters `Ys` of a single operation `X` (i.e., *op_outputs(FindFinestRestaurant, [RestaurantName, RestaurantAddress])*). Besides these basic fluents, we have fluents for representing user constraints in the FLUX query. For example:

```
Cond(C,if(guard(O1,RO1,V1),O3),else(guard(O2,RO2,V2),O4)):
INDEX x PARAMETER x OPERATOR x VALUE x PARAMETER x
PARAMETER x OPERATION x VALUE x PARAMETER --> FLUENT
```

`Cond(C,if(guard(O1,RO1,V1),O3), else(guard(O2,RO2,V2),O4))` is a fluent for condition guard request. The index `C` request contains two alternative paths if the first guard condition indicating parameter value O1 with relational operation RO1 equals to value V1 is *true* then parameter O3 exists; else if the second guard condition saying parameter value O2 with relational operation RO2 equals to V2 is *true* then parameter O4 exists. For instance, *cond(c1, if(guard(weather,==,sunny),ticketD),else(guard(weather,==,rainy),ticketM))*. It's worth noting that `DataTransform(X,X1)` fluent might be hold if there is transformation of data passing between parameter `X`, where `X` is a subset of operation outputs and parameter `X1`, where `X1` is a subset of another operation inputs.

Whereas, the partial fluents for abstract plan structures are listed below:

```
Add_initail(X): OPERATION --> FLUENT
Add_goal(X): OPERATION --> FLUENT
Flow(X,Y): OPERATION x OPERATION --> FLUENT
Flow(X, if(guard(O1,RO1,V1),Y1),else(guard(O2,RO2,V2),Y2)):
OPERATION x PARAMETER x OPERAND x VALUE x OPERATION x
PARAMETER x OPERAND x VALUE x OPERATION --> FLUENT
```

To construct the sequencing and paralleling plans, `Add_initail(X)`, `Add_goal(X)` and `Flow(X,Y)` fluents are required, where `Add_initail(X)` is a fluent for an initial node of operation `X` (i.e., *add_initail(FromCoordinatesToCity)*), `Add_goal(X)` is a fluent for a goal node of operation `X` (i.e., *add_goal(GetDirection)*) and `Flow(X,Y)` is a fluent for a flow node with a pair of head operation `X` and tail operation `Y` (i.e., *flow(FromCoordinatesToCity,FindFinestRestaurant)*).

If fluent `Op_inputs(X,Ys)` is hold, this means all inputs Ys of operation X are matched to either initial parameters or output parameters of other operations. Thereafter, the agent adds the fluents either `Add_initial(X)` or `Flow(X,Y)` into the abstract plan.

To merge condition operations into the abstract plan, the service composition agent uses `Flow(X, if(guard(O1,RO1,V1),Y1), else(guard(O2, RO2,V2),Y2))` fluent. The `Flow(X,if(guard(O1,RO1,V1), Y1), else( guard(O2,RO2,V2), Y2))` is fluent for a flow with condition of two alternative outgoing paths. The first one is a path from operation `X` to operation `Y1` if the guard condition that parameter value `O1` with relational operation `RO1` equals to `V1` is true. The second path is from operation `X` to operation `Y2` if the guard condition that parameter value `O2` with relational operation `RO2` equals to `V2` is true.

4.4 Execution of the Resulting Composition

From Pathathai's dissertation...

The Figure 7 illustrates the architecture of Composite Platform (or business process) Generation. The process starts from the BPMN transformer that converts the abstract plan, which is consisted of a sequence of operation fluents, to BPMN model in Prolog language. Next, the BPMN model is analyzed and verified in the BPMN Validation. The valid BPMN model as a result will be implemented and executed in the experiment phase later on.

Abstract plan to BPMN semantics A BPMN is a standard notation maintained by OMG[12] for modeling business processes. Its goal is to provide a notation of business specification that is understandable by all business stake holders (i.e. business analysts, software developer and business people), mainly at the level of domain analysis and high-level systems design [16]. The BPMN is widely-used in the early stages of systems life cycle. According to the OMG, 72 implementations of the BPMN are reported for known businesses [16]. Moreover, open

[12] http://www.omg.org/

sourced software companies (i.e., Activiti[13], BonitaSoft[14] and Yaoqiang BPMN Edior[15]) dramatically compete among each others to offer varied solutions to edit and run BPMN models.

BPMN is comprised of an abstract of workflow components. However, this dissertation focuses on a control-flow perspective of BPMN. Therefore, the subset of the notation that handles the order of activities are allowed to occur. It does not handle its non-functional features (i.e., artifacts and association) and organizational modeling features (i.e., lanes and pools).

Figure 8 shows an overview of a set of graphical BPMN elements related to the proposes of our work. For the event elements, only **start event** and **end event** are taken into consideration. **Service tasks** are main knowledge of processing elements; each task is a finite process with a set of inputs and a set of outputs. Two gateways: split and join control a workflow. **Split gateways** present when branching of the workflow takes place; two disjoint subtypes of splits are **AND-split gateway** and **XOR-split gateway**. AND-split allows a single flow to be split into two or more branches which can execute tasks concurrently while XOR-split allows a flow to be split into two or more flows when the incoming flow is enabled, the gateway is passed to one of the outgoing flows based on a specified condition that can select one of the outgoing flows. **Join gateways** happen when two or more paths meet; two further disjoint subtypes of merge modes are considered: **AND-join gateway** and **AND-split gateway**. AND-join allows two or more parallel flows to be joined into a single subsequent flow when all input flows have been enabled while AND-split allows a single flow to be split into two or more branches which can execute tasks concurrently. Lastly, a **sequence flow** is used to link two entities of event, activity or gateway in a process diagram and specify a control flow relation.

For BPMN transformation purpose, we map from logical analysis of BPMN component to their logical models, properties and representation in Prolog. The following table 5 lists BPMN elements along with their mapped semantic fluents into consideration:

The BPMN elements are mapped into the BPMN semantics according to the element type. For example, Start event is mapped to node(start) and so on. However, Service task and Sequence flow BPMN element require more informations for semantic mapping. A semantic of Service task needs informations of task name, inputs and outputs for defining task(Name, Inputs, Outputs), where Name, Inputs and Outputs are variable names. The same principle applies to Sequence flow that it needs a workflow information of head operation linking to tail operation for flow(HOperation,TOperation), where HOperation and TOperation are operation names.

BPMN Transformer The BPMN transformer is a process for mapping the abstract plan consisting of a sequence of the fluents into semantic BPMN model in the declaration of a formal language. The reason why we transform the abtract

[13] http://activiti.org/
[14] http://www.bonitasoft.com/
[15] http://bpmn.sourceforge.net/

plan into the formal language is that a graphical notation of BPMN elements binds information of data passing, data transformation and routing condition from BPMN workflow specification. This shall be hard to fix defects when they occur in BPMN model.

The following Table 6 shows mapping rules between particular fluents occurred in the abstract plan and semantic BPMN we defined in Table 5. We have classified the fluents into four groups: *initial group, flow group, goal group* and *data group*.

The *initial group* contains add_initial(Op) fluents, where Op is a single operation. Mapping these initial fluents into BPMN could create two possible situations. The first situation happens when the transformer detects only one initial fluent in the abstract plan. The transformer creates node(start) and flow(start,Op), linking between start event and operation Op, into BPMN model. While the second situation occurs when there are more than one initial fluents in the plan. This means operations derived from initial fluents can start a process at the same time. A converted BPMN has one AND-split gateway to combine these initial operations. For example, add_initial(Op1), add_initial(Op2) is mapped to BPMN model, which is consisted of node(start), gateway(andS), flow(start,andS), flow(andS,Op1), flow(andS,Op2).

The *flow group* contains flow(_,_), flow(_,if(guard(_,_,_,_),_),else(guard(_,_,_,_),_)) and flow(if([_,_],_)) fluents. These flow fluents can be mapped into a sequence, parallel and condition flows in BPMN model. For the *sequence flow*, the transformer does not change flow(_,_) fluent. For instance, flow(Op1,Op2), where Op1 and Op2 are instants of Task Op1 and Task Op2, stays remain in BPMN since it describes a control flow from task object to another task object in the same way as a flow dose in BPMN. While creating *parallel flow* in BPMN, the transformer checks all tasks in the head position of flow fluent. i.e., OP1 in flow(Op1,_) fluents whether Op1 exists in any head position of other flow(Op1,_) fluents. If these flow fluents exist, the transformer convert them to one AND-split gateway, relevant control flows and relevant tasks. For example, flow(Op2,Op3), flow(Op2,Op4) is mapped to gatway(andS), flow(Op2,andS), flow(andS,Op3), flow(andS,Op4). This checking parallel rule also is applied for AND-join gateway that the transformer monitors the tail position of flow fluent. i.e., Op1 in flow(_,Op1) fluent with others flow(_,Op1) fluents instead. For the *condition flow*, the transformer checks the abstract plan for flow(_,if(guard(_,_,_,_),_), else(guard(_,_,_,_),_)) fluent and flow(if([_,_],_)). If the former fluent is detected, the transformer creates one XOR-split gateway, one control flow and two control flows with condition into BPMN model. For example, flow(Op1,if(guard(O1,RO1,V1),Op2),else(guard(O2,RO2,V2),Op3) is mapped to gatway(xorS), flow(Op1,xorS), flow(xorS,Op2,guard(O1,RO1,V1)), flow(xorS,Op3,guard(O2,RO2,V2)). While the flow with XOR-join gateway is created if the latter fluent is captured. For instance, flow(if[Op2,Op3],Op4) is mapped to gatway(xorJ), flow(Op2,xorJ), flow(xorJ,Op3), flow(xorJ,Op4).

The *goal group* contains add_goal(Op) fluents, where Op is a single operation. We follow BPMN specification that a workflow in BPMN model may have more than one end events. Therefore, the transformer creates goal nodes up to number

of distinct add_goal(_) fluents found. For example, add_goal(Op4), add_(Op5) is mapped to node(end1), node(end2), flow(Op4,end1), flow(Op5,end2).

Lastly, the *data group* contains data passing and data transforming fluents. The data passing describes both the task Op creation and its data passing of inputs and outputs. To do so, the transformer searches for service operations used in the abstract plan. For each operation Op, the transformer combines op_inputs(Op,[inputs]) and op_outputs(Op,[outputs]) fluents and map them to task(Op,[inputs],[outputs]) fluent, referring to Task Op containing a set of its inputs and its output. While dataTransform(X,X1) fluent, capturing a change from a data form X into another form X1, in the abstract plan stays remain in the semantic BPMN model.

5 Conclusion and further work

References

1. G. Baryannis and D. Plexousakis. Fluent calculus-based semantic web service composition and verification using WSSL. In *Service-Oriented Computing - ICSOC 2013 Workshops - CCSA, CSB, PASCEB, SWESE, WESOA, and PhD Symposium, Berlin, Germany, December 2-5, 2013. Revised Selected Papers*, pages 256–270, 2013.
2. P. Bertoli, M. Pistore, and P. Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence*, 174(3-4):316–361, 2010.
3. G. Cassar, P. Barnaghi, and K. Moessner. A probabilistic latent factor approach to service ranking. In *Proc. of the International Conference on Intelligent Computer Communication and Processing*, pages 103–109, Aug 2011.
4. G. Cassar, P. Barnaghi, and K. Moessner. Probabilistic matchmaking methods for automated service discovery. *IEEE Transactions on Services Computing*, 7(4):1–1, May 2013.
5. R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web services description language (wsdl) version 2.0 part 1: Core language. W3C recommendation, W3C, June 2007. Web 12 Jan. 2015, http://www.w3.org/TR/2007/REC-wsdl20-20070626/.
6. M. Crasso, A. Zunino, and M. Campo. Easy web service discovery: A query-by-example approach. *Science of Computer Programming*, 71(2):144–164, Apr. 2008.
7. J. Farrell and H. Lausen. Semantic annotations for WSDL and XML schema. W3C recommendation, W3C, #aug# 2007. http://www.w3.org/TR/2007/REC-sawsdl-20070828/.
8. A. Gomez-Perez, O. Corcho-Garcia, and M. Fernandez-Lopez. *Ontological Engineering*. Springer-Verlag New York, Inc., 2003.
9. N. Kokash, W.-J. van den Heuvel, and V. D'Andrea. Leveraging Web Services Discovery with Customizable Hybrid Matching. In *Proc. of the 4th International Conference on Service Oriented Computing*, pages 522–528, 2006.
10. K.-H. Lee, M.-Y. Lee, Y.-Y. Hwang, and K.-C. Lee. A framework for xml web services retrieval with ranking. In *Proc. of the International Conference on Multimedia and Ubiquitous Engineering, 2007.*, pages 773–778. IEEE Computer Society, 2007.

11. J. Ma, Y. Zhang, and J. He. Web Services Discovery Based on Latent Semantic Approach. In *Proc. of the International Conference on Web Services*, pages 740–747. IEEE Computer Society, 2008.
12. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. Owl-s: Semantic markup for web services, 2004.
13. G. A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, Nov. 1995.
14. P. Na-Lumpoon, M. Lei, I. Caicedo-Castro, M.-C. Fauvet, and A. Lbath. Context-Aware Service Discovering System for Nomad Users. In *7th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2013)*, 2013.
15. P. Na-Lumpoon, M. Lei, T. Kamnardsiri, A. Lbath, and M.-C. Fauvet. Illustrating some issues raised when designing context-aware personalized services for mobile users. In *Proceeding of the 6th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2012)*, 2012.
16. OMG. Business process model and notation (bpmn)) version 2.0.2. n.d. Web. 20 Dec. 2014, http://www.omg.org/spec/BPMN/2.0.2/.
17. A. V. Paliwal, N. R. Adam, and C. Bornhövd. Web Service Discovery: Adding Semantics through Service Request Expansion and Latent Semantic Indexing. In *Proc. of the International Conference on Services Computing*, pages 106—113. IEEE Computer Society, 2007.
18. A. V. Paliwal, B. Shafiq, J. Vaidya, H. Xiong, and N. R. Adam. Semantics-based automated service discovery. *IEEE Transactions of Services Computing*, 5(2):260–275, 2012.
19. S.-L. Pan and Y.-X. Zhang. Ranked Web Service Matching for Service Description Using OWL-S. In *Proc. of the International Conference on Web Information Systems and Mining*, pages 427–431, Nov 2009.
20. C. Platzer and S. Dustdar. A vector space search engine for web services. In *Proc. of the 3rd International Conference on Web Services*, pages 14–16. IEEE Computer Society, 2005.
21. A. Sajjanhar, J. Hou, and Y. Zhang. Algorithm for web services matching. In J. Yu, X. Lin, H. Lu, and Y. Zhang, editors, *Proc. of the 6th Asia-Pacific Web Conference*, volume LNCS 3007, pages 665–670. Springer Berlin Heidelberg, 2004.
22. M. Shamsfard and A. A. Barforoush. Learning ontologies from natural language texts. *International Journal of Human–Computer Studies*, 60:17–63, 2004.
23. E. Sirin, B. Parsia, D. Wu, J.-A. Hendler, and D.-S. Nau. HTN planning for web service composition using SHOP2. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):377–396, 2004.
24. M. Thielscher. Flux: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 5(4-5):533–565, 2005.
25. W3C. Web services glossary. www.w3c.org/TR/ws-gloss. Last accessed: 20th of March, 2014.
26. W3C. SAWSDL candidate recommendation implementation report, 2002. n.d. Web. 25 Dec. 2014, http://www.w3.org/2002/ws/sawsdl/CR/.
27. Y. Wang and E. Stroulia. Semantic structure matching for assessing web service similarity. In *Proc. of the 1st International Conference on Service Oriented Computing*, pages 194–207. Springer-Verlag, 2003.
28. S. M. Watt, V. Negru, T. Ida, T. Jebelean, D. Petcu, and D. Zaharie, editors. *11th International Symposium on Symbolic and Numeric Algorithms for Scientific*

Computing, SYNASC 2009, Timisoara, Romania, September 26-29, 2009. IEEE Computer Society, 2009.

29. C. Wu. WSDL Term Tokenization Methods for IR-style Web Services Discovery. *Science of Computer Programming*, 77(3):355–374, Mar. 2012.
30. C. Wu, E. Chang, and A. Aitken. An empirical approach for semantic web services discovery. In *19th Australian Conference on Software Engineering*, pages 412–421, March 2008.
31. C. Wu, V. Potdar, and E. Chang. Latent Semantic Analysis – The Dynamics of Semantics Web Services Discovery. In *Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web*, pages 346–373. Springer-Verlag, 2009.
32. Z. Wu, K. Gomadam, A. Ranabahu, A.-P. Sheth, and J.-A. Miller. Automatic composition of semantic web services using process mediation. In *9th International Conference on Enterprise Information Systems, Volume SAIC*, pages 453–462, Madeira, Portugal, 2007.
33. H. Zhao and P. Doshi. A hierarchical framework for logical composition of web services. *Service Oriented Computing and Applications*, 3(4):285–306, 2009.

Table 1. Retrieval effectiveness.

Model	NDCG@10	Gain (%)
Models applied in prior research on IR-based service discovery		
VSM (baseline)	0.5435	N/A
LDA	0.6661	22.55
LSI-SVD	0.7586	39.57
Proposed family of models for text-based service retrieval		
QECOT-NMF	0.7792	43.37
QECOT-SVD	0.7804	43.59
QECOT-MSE	**0.7897**	45.29

Table 2. Student's paired t-test on NDCG@10 to compare QECOT-MSE with other models applied in prior research on IR-based service discovery

Model QECOT-MSE	NDCG@10 **0.7897**	p-value	is statistically significant?
VSM	0.5435	3.47×10^{-7}	Yes
LSI-SVD	0.7586	0.08039	No
LDA	0.6661	7.613×10^{-5}	Yes

User problem domain		**Planning domain**
Query	RestaurantName(goal) BookingReservation(goal) Direction(goal) NumberOfGuest(2) Time(8pm)	*goals*([RestaurantName, BookingReservation, Direction]) *initial*(NumberOfGuest) *initial*(Time)
Context	Date(01/06/2014)	*initial*(Date)
Profile	Name(Alice)	*initial*(Name)

Table 3. Example of a mapping between user problem domain and planning domain

Operations in Problem domain	FindFinestRestaurant input: City outputs: RestaurantName, RestaurantAddress
Operations in Planning domain	*op_inputs*(FindFinestRestaurant, [City]) *op_outputs*(FindFinestRestaurant, [RestaurantName,RestaurantAddress])

Table 4. Example of service operations mapping between problem domain and planning domain

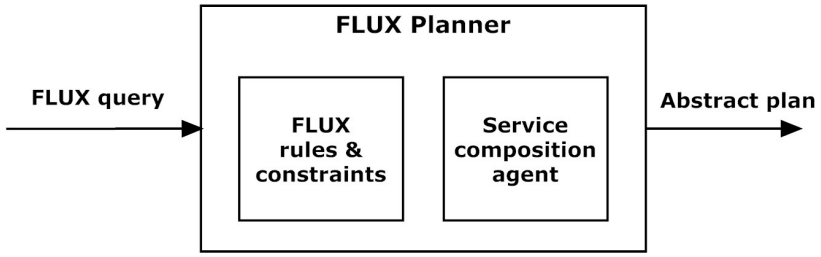

Fig. 6. FLUX planner

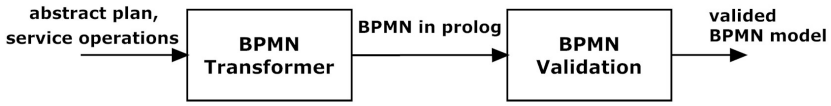

Fig. 7. BPMN generation process

BPMN elements	**BPMN semantics**
Start event	node(start)
End event	node(end)
Service task	task(Name,Inputs,Outputs)
Sequence flow	flow(HOperation,TOperation)
XOR-split gateway	gateway(xorS)
XOR-join gateway	gateway(xorJ)
AND-split gateway	gateway(andS)
, AND-join gateway	gateway(andJ)

Table 5. Mapping between BPMN elements and BPMN semantics

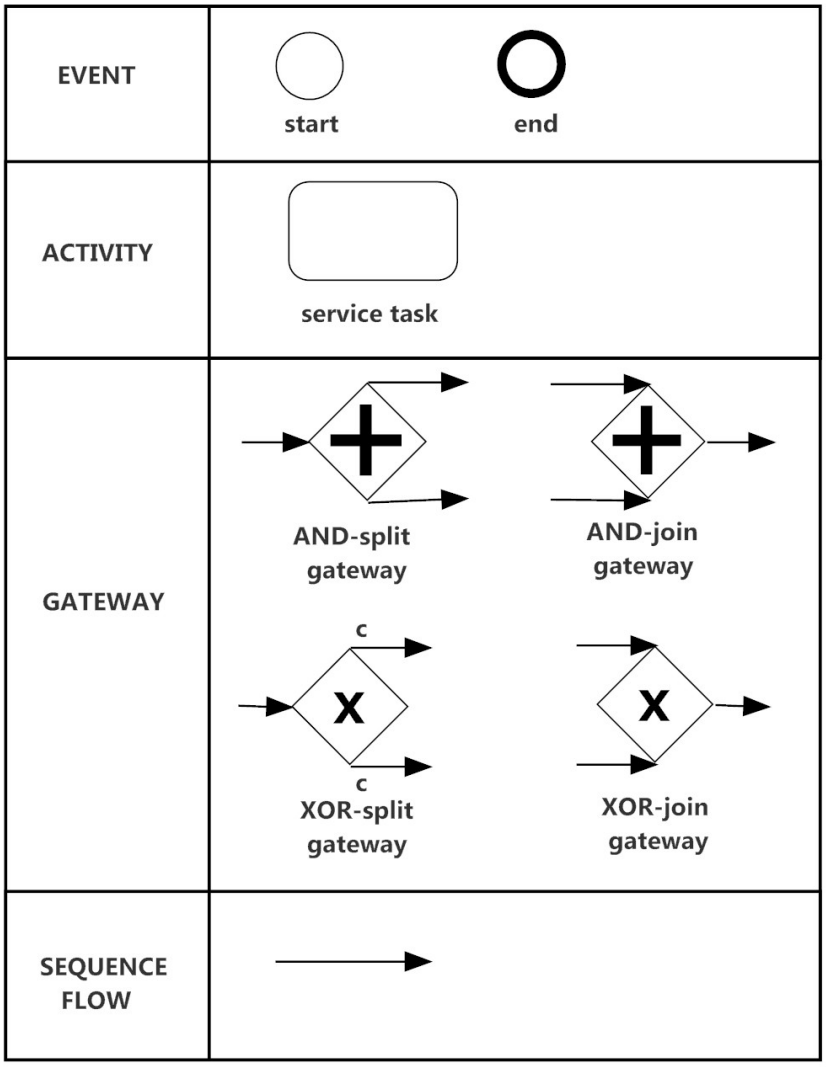

Fig. 8. BPMN notation related to the proposed of our work [16]

Group	Fluents in Abstract plan	BPMN
Initial	*initial* add _initial(Op1)	node(start) flow(start, Op1)
	initial with parallel add _initial(Op1) add _initial(Op2)	node(start) gateway(andS) flow(start,andS) flow(andS,Op1) flow(andS,Op2)
Flow	*flow with sequence* flow(Op1,Op2)	flow(Op1,Op2)
	flow with AND Split parallel flow(Op2,Op3) flow(Op2,Op4)	gateway(andS) flow(Op2,andS) flow(andS,Op3) flow(andS,Op4)
	flow with AND Join parallel flow(Op3,Op5) flow(Op4,Op5)	gateway(andJ) flow(Op3,andJ) flow(Op4,andJ) flow(andJ,Op5)
	flow with XOR Split condition flow(Op1, if(O1,S1,V1,Op2), else(O2,S2,V2,Op3)	gateway(xorS) flow(Op1, xorS) flow(xorS,Op2,guard(O1,S1,V1)) flow(xorS,Op3,guard(O2,S2,V2))
	flow with XOR Join condition flow(if([Op2,Op3], Op4)	gateway(xorJ) flow(Op2, xorJ) flow(OP3,xorJ) flow(xorJ,OP4)
End	*goal* add_goal(Op4) add_goal(Op5)	node(end1) node(end2) flow(Op4, end1) flow(Op5, end2)
Data	*data passing* op_inputs(Op,[Inputs]) op_outputs(Op,[Outputs])	task(Op, [inputs], [outputs])
	data transforming dataTransform(X,X1)	dataTransform(X,X1)

Table 6. Mapping between the abstract plan and BPMN workflow