SQUIREL: Semantic Querying Interlinked OWL-S traveling Process Models

Afaf Merazi
EEDIS Laboratory, Computer Science Department. Dijillali LIABES University, Sidi Bel Abbes, Algeria
Email: afaf.merazi@univ-sba.dz

Mimoun Malki
High School of Computer Science of Sidi Bel Abbes (ESI - Sidi Bel Abbes) Algeria
Email: malki@univ-sba.dz

Abstract—With the advent of new forms of information and communication technologies, the consumer needs to combine and customize different travel components as a complete travel package, namely: Dynamic Packaging Technology. Nevertheless, disparate tourist offers and services make it difficult for consumer to use them effectively. Therefore, our paper presents an intelligent querying framework of OWL-S travel services, called SQUIREL composition engine. It uses Semantic Web Services (SWSs) technologies combined with the useful of Linked e-tourism Data concept to fulfill the preferences and constraints of the e-tourist any time. This purpose supports SWSs pre-selection through the valuation of the rewritten SPARQL consumer query at runtime that manages dynamic service dependencies extracted from Linked e-tourism Data and returns the SWSs endpoint. Then, SQUIREL catches this endpoint and makes the necessary optimizations to refine it to its relevant atomic processes needed to be composed using matrix computation. However, the experimental results indicate that this method owns both lower computation cost and higher success ratio of fine-grained discovery-based atomic processes composition.


I. INTRODUCTION

Nowadays, e-tourism supports more innovative and sophisticated tasks encouraging the consumer to seek for more personalized and customized tourism offers with a characterized need of efficiency and sense of control anytime and anywhere, called Dynamic Packaging (DP) [1]. Cardoso[2,3] is the pioneer for developing a valid DP product involved in the SEED (SEmantic E-tourism Dynamic packaging) project and described as a multilayer framework that integrates Semantic Web Services as semantic mediators ready to be composed anytime, Bilbao [4] presents also a semantic e-business platform that composes the appropriate processes given up the consumer’s desires and restrictions. Unfortunately combining and booking disparate components is still a time consuming and a challenging task, due to the continuous overloaded travel’s information1 and booking platforms 2. In order to provide an intelligent and proactive access to relevant high quality online travel information and services, the DP can particularly benefit from: (1) Linked e-Tourism Data3[5,6,7] that integrates business offers across different data sources and (2) Semantic Web services (SWSs) [8] technologies to book multiple travel components as a complete travel package. Therefore, this paper describes a tailored framework for consumers that group the appropriate travel atomic processes (indivisible operation defined in OWL-S process model) they are interested in, Called SQUIREL. Our proposal employs the notion of Linked e-tourism Data field that achieves the efficiency of the automatic OWL-S [8] discovery-based composition by supplying the necessary schema-based alignment [9] to rewrite the consumer query for pre-selecting the appropriate travel process models. However, the pre-selection result needs to be refined to its relevant sub-set atomic processes in order to find the final and optimal composition solution. Therefore, applying a local optimization technique is going to generate a set of sub-workflows represented as a sub-matrices needed to be merged in new one on which we apply a matrix computation technique that returns the new composite service. The proposed approach conducts fast service composition and proved to be very effective and efficient in determining the optimal atomic processes composition plan.

The rest of the paper is organized as follows: background is presented in Section 2; Section 3 describes a motivating travel scenario showing the meaningful of fine-grained discovery-based service composition, an architecture framework is detailed in Section 4. Section 5 presents the prototype implementation and the evaluation of our approach. Finally, related work is discussed in Section 6 followed by concluding remarks and future work.

1 Consumer can get information on routes, timetables, seat availabilities, book rental cars and restaurants etc.
2 Such as online travel portals, like Expedia.com, Travelocity.com, Orbitz.com and Kayak.com.
3 Available at: http://datahub.io/dataset?tags=tourism.
II. BACKGROUND

The Semantic Web [10] is defined as a web of data, comprised of resources machine readable and connected with links where a resource (the subject (s)) is linked to another one (the object (o)) through an arc labeled with a third one (the predicate (p)), using Resource Description Framework (RDF) [11] language that represents metadata as an RDF data as an RDF Graph (a set of RDF triples). An RDF triple is formalized as a tuple \( s, p, o \in U \cup B \cup L \times (U \cup B) \times (U \cup B \cup L) \) where \( U \) denotes the set of URIs, \( B \) the set of blanks and \( L \) the set of literals, are three pair wise disjoint infinite sets but their union forms the RDF terminology \( T \). The ontology (Web Ontology Language (OWL)) [12] defines a common vocabulary for representing knowledge about a domain and stores several concepts organized on different hierarchic levels and a set of relationships (object/data-type properties) that necessarily hold among those vocabulary terms and instances.

SPARQL [13, 14] is the most popular RDF query language, defined on top of basic graph patterns as RDF graphs with variables (from a set \( V \), a set of RDF triple patterns, where each triple pattern is a tuple \( s, p, o \in (T \cup V) \times (U \cup V) \times (T \cup V) \). Combining RDF triple patterns is possible using SPARQL operators such as AND- Grouping, Union-patterns, Nesting, Optional parts and Filtering Query Modifiers...A SELECT SPARQL query is expressed using a form resembling the SQL SELECT query. Answering to an RDF query is seen as a graph homomorphism from SPARQL graph pattern into an RDF Graph and defined as a partial mapping function from \( V \) to RDF terms \( \in T \); \( \mu: V \rightarrow T \). In order to reduce the search space over RDF graphs, Alkhateeb [15] define an extension of SPARQL query language with the use of a set of regular expression patterns over predicates characterizing the traversed paths of arbitrary length in a query called Path SPARQL (PSPARQL). For instance, the following query returns all cities reachable from the city Sydney by a sequence of trains and planes only:

\[
\text{SELECT ?city WHERE \{ \text{dbpedia-owl:Sydney} \text{(tio:train | tio:bus)}+ ?city }}
\]

Semantic Web Services (SWSs) are software components that are incorporated and reused into distinct distributed applications without the concern of how the service was implemented. We focus on the most promising SWSs language, the OWL-S (Web Ontology Language for Web Services) [8], a kind of OWL ontology specification that formalizes semantically service capabilities by providing: (1) the service profile that presents the Inputs, Outputs, Preconditions and Effects (IOPEs) and service category, (2) the grounding provides the needed details about transport protocols, (3) the process model (PM) presents the behavior of the service as a process, either atomic or composite which receives and sends a single message or retains/changes state through a sequence of messages, where:

- An atomic process (AP) is a directly invocable and executed entity that describes: (a) IOs parameters ex-pressed as a subclass of the parameter class in OWL-S and (b) PEs modeled as logical formulas or expressions.
- The Composite processes provides a concrete and dynamic semantic description of the logical execution order of the finite set of sub-processes that are connected to each other using OWL-S control constructs: Sequence, Split, Split+ Join, Unordered, Choice, If-Then-Else, Iterate and Repeat-Until/Repeat- While. Their IOPEs are described with concepts formally defined by means of ontologies.

III. RESEARCH MOTIVATING EXAMPLES

In order to better illustrate our approach, we consider a travel scenario where a consumer wants to book a hotel room where the conference event takes place and want to check the weather condition of this city’ event.

Given four OWL-S process models that are depicted in Figure 1, where each OWL-S process model describes the internal behavior of its travel atomic processes in accordance with their semantic functionalities cross different travel domain ontologies: The first service manages an event registration with payment process based four atomic processes; The second one allows the consumer to book a star rating accommodation based two Atomic processes; The third one allows to choose or to book a room hotel relying on three atomic processes. Finally, the last one checks the weather condition of a city name or of geographical coordinates. However, these four services are closely dependent on the others due to the fact that the outputs of EventReservationService produces the inputs of the three remaining, since all process models need to work together on strategy so as to satisfy the requirement of consumer.

\[
\text{\textbf{Fig. 1. OWL-S Process Models Examples}}
\]

IV. SQUIREL FRAMEWORK

We propose a SPARQL-driven approach (L0) (see figure 2), a uniform way for: (1) searching information needed from Linked e-Tourism Data [16] (through the second Layer (L1)), and/or (2) composing automatically...
different OWL-S travel services (through the third layer (L2)).

![Fig. 2. A proposed intelligence traveling architecture](image)

4.1. Semantic Querying Layer (L0)

While Linked e-Tourism Data offer for consumers a way to search and retrieve travel information using SPARQL semantic query language [16], Garcia [17] offer an efficiency approach enabling SWSs to be discovered and used by other ones using two different SPARQL concrete queries that improves the execution time. The first one, the Qall filter finds a reusable SWS that fulfills the whole set of related terms described by the consumer query but the second one, the Qsome filter offer more flexibility and returns a suitable composition of SWSs that contain at least one of the terms referred by the consumer query and so, can be composed to fulfill the requirements. The Input/output of a process model (PM), is a message received/delivered, corresponding to the <process:hasInput> /<process:hasOutput> properties in OWL-S codes. The formalization of OWL-S process model functionalities (I/O parameters) in RDF triple using Turtle-based syntax is in the form:

<PM, pr:hasInput,I1>...<PM,pr:hasOutput,Oj>. So, for answering complex consumer request, single PM is rarely used and so, the construction of a new composite service by integrating and reusing the existing ones is required using the Qsome filter.

Definition1. As input of SQUIREL, a 3-tuple DPS = (T, RQ, PMs) that represent a discovery process where T is the union of several travel ontologies involved in LED, T = {T1,T2,...,Tn} a consumer SPARQL query RQ= (I0,O0) that specifies a set of ontological concepts describing the provided inputs I0 and requested outputs O0(I0, O0 ⊆ T0 ⊆ T) and PMs a set of travel SWSs belonging to NAICS Travel services category (5615).

Definition2. A PM is defined as a 4-tuple (AP, E, IPM, OPM) where the atomic processes (APs) are the set of indivisible operations, IPM is a set of inputs parameters required to invoke PM, OPM is the set of outputs parameters returned by PM after the execution of PM (IPM, OPM ⊆ TPm ⊆ T) And E contains the control flow relation between processes. Both of RQ and PMs are described at a semantic level using well-defined ontological concepts defined in T.

Given the example defined in section 3, we formulate RQ using the Qsome filter by employing UNION or FILTER SPARQL operator that delivers acc:Hotel or tio:Event ... and provides s:eventReservation or s:LodgingReservation, as follows:

```
SELECT ?s WHERE {
  ?s pr:hasInput ?I1.
  ?s pr:hasOutput ?Oj.
  FILTER (((?I1=acc:Hotel) || (?I1=tio:Event) ||...) && ((?Oj=s:eventReservation) ||(?Oj=s:LodgingReservation ...)))
}
```

4.2. Interlinked E-Tourism Data Layer (L1)

Linked data [18,19] is defined as a vast, distributed data space that use many different vocabularies in different data formats4 build on a simple set of standards5 where the entities are interlinked for creating a vast collection of data graph6 that spans data sources and enables the discovery of new data sources. However, some approaches deals with linked data cloud in e-tourism domain, there may be mentioned: TourMISLOD [5] and OpeNER [6,7] (Open Polarity Enhanced Name Entity Recognition). Our framework uses an existing Linked e-tourism Data [7] allowing us to find the binary semantic relationships between two concepts (a computed pair-wise similarity): (1) equivalent concept degree (Exact Match (≡), a symmetric predicate) or (2) sub-concept degree (Plugin Match(⊙) defined as below:

Definition3. [Generalized Concept alignment relationship] Given two concepts A and B, defined in two ontologies O1 and O2 such as A ∈ O1 and B ∈ O2, We say that $A \triangleright B$ if $(A = B) \lor (A \prec B)$ where:

1. $A = B$, A is semantically equivalent to B, and
2. $A \prec B$, A is a sub-Concept of a B iff:
   a. $(A \equiv C) \land (C \supseteq B)$ where $(C \in O_2)$
   b. $(A \supseteq D) \land (D = B)$ where $(D \in O_1)$

There are several approaches [9, 21] that addresses the interlinking discover process that takes two datasets as input and produces automatically a collection of alignment between concepts of the two datasets as output across ontologies called Schema-based alignment using instance alignment techniques, the most widely used is SILK [20] that handle large volumes of data and obtain good results with high precision. There may be mentioned some concept mappings related to the example that are useful for the discovery process:

4 So there are many different ways to represent the same information
5 RDF, URIs, HTTP
6 Linked Datasets (i.e., with Dereferenceable URIs) available as RDF Dumps http://www.w3.org/wiki/DataSetRDFDumps

Copyright © 2015 MECS I.J. Information Technology and Computer Science, 2015, 12, 30-39
4.3. Interlinked Transactional Services (L2)

This layer proposes a flexible service composition that invokes services on-demand and at runtime due to the changing environment.

4.3.1. Basic Representation of Travel Services

In order to store the knowledge derived from PMs behavior, we exploit the use of graph theory [22] that benefit from the use of matrix theory due to the fast access to its nodes and specially the adjacency matrix A = (V, E) that is an N-vertex directed graph where V is a finite set of vertices and E is a set of directed edges. Element A_{ij} = 1 if and only if the edge (i, j) ∈ A. All other elements are zero. A row of A lists the nodes at the tip of the outgoing edges while a column of A lists the nodes at the tail of the incoming edges. Based on this, the adjacency PM matrix (APM) is an N-square binary Boolean matrix that represents the dependencies between the AP by analyzing the complete behavior of the service, APM = [APM]_{nn} (N denotes the number of atomic processes). The composite process determines the inter-PM dependency between APs such as if a sequence construct control edge connects two vertices from AP_{1} to AP_{2} (seq(AP_{1},AP_{2}) is in E(PM)), APM_{ij} = 1. If there is no such edge in E(PM), APM_{ij} = 0. The matrix have zeros diagonal with no self loop that implies all services are independent on itself and form an acyclic dependency on it. Therefore, parsing the nested structure of OWL-S control structures in a top-down manner allows us to rewrite logically each complex structured process (choice, if-then-else, split+join and split) [23] to a simpler form in terms of sequence control construct as below:

- Seq(S_i, S_j) → PM[S_i, S_j] = 1.
- Seq(S_i,α(S_i,S_j)) → From S_i we can go to S_j and/or S_i, we need to add two edges to APM : Seq(S_i,S_j) ∨ Seq(S_j,S_i) → PM[S_i,S_j]=1. PM[S_j,S_i]=1 where α = Split | IfThenElse .
- Seq(α(S_i, S_j), S_i)) → From S_i we can go to S_j and/or S_i, we need to add two vertices to APM : Seq(S_i, S_j) ∧ Seq(S_j, S_i) → PM[S_i, S_j] = 1 , PM[S_j, S_i] = 1 Where α = Split+Join | Choice .

Let us consider now the inter-PM dependency of EventReservationService, presented in Section 3, it contains four atomic processes and consists of three sequencing edges that connects three sub-processes SelectEvent (E1), SignInEvent (E2) and a Choice process between BankTransferPayement (E3) and creditcardpayement (E4). The service illustrates a sequence edge from E1 to E2 and from E2 to E3 or E4. In conclusion, E(PM) = { Seq(E1,E2), Seq(E2,E3), Seq(E2,E4)}. Based on the description set out above, we present the adjacency EventReservationService matrix depicted in Figure3:

<table>
<thead>
<tr>
<th></th>
<th>E_1</th>
<th>E_2</th>
<th>E_3</th>
<th>E_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E_2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E_3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>E_4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 3. Adjacency Matrix Example

4.3.2. Improving Dynamic SWS Pre-Selection Using SPARQL Rewriting

As a prerequisite for SWS composition, finding the right services to reuse and compose cross different ontologies, other-wise it is meaningless. For that, we propose a fine-grained discovery process that queries the IOs parameters of the PMs as a RDF graph using PSPARQL endpoint and none on the IOs parameters of its atomic process children that increase the search time and also the complexity. The intra-PM dependency includes the relationship between PMs according to their functionalities. Two services PM_{1} and PM_{2} are semantically interlinked if there is a semantic matching degree (Exact or Plug In matching) between the output parameters subset of PM_{1} and the input parameters subset of another one PM_{2} and defined as:

Definition 4. [Intra-PM dependency]

Let PM_{1}, PM_{2} be two description services, and let O_{PM_{1}}, I_{PM_{2}} be set of outputs of PM_{1} and set of inputs of PM_{2}. There is an intra functional dependency between PM_{1} and PM_{2}, if ∃ an output O_i ∈ O_{PM_{1}} that matches an input I_j ∈ I_{PM_{2}} with a generalized match degree if both concepts O_i > I_j .

Therefore, managing dynamic service dependencies should take schema-based alignment (resulting from LED) into account. In order to support the intra-PM dependency, we need to rewrite the main SPARQL query R_{0} in a new one R_{Q} by using path expressions (defined in PSPARQL language) that combine generalized multiple matching patterns. The following query expresses the direct and indirect intra-PM dependency with the regular expression that searches all pairs of services connected by variable path length with a sequence of an output of service s_i that feature semantically or not to the input of another service s_i with regards to the generalized alignment relationships, depicted in figure4 (see (a),(b) and (c)):

\[ (\text{at least 1}, \text{using the repeat operator plus}) \]

\[ (\text{Where } ^{-1} \text{ is the inverse operator. For example, given the RDF triple } (s,p,o), \text{ we can deduce } (o,p,s)) \]

\[ (\text{equivalent or sub-concept using the repeat operator star}) \]
(a) The following query expresses the direct intra-PM dependency:

\[ \text{\texttt{?s1 pr:hasOutput. - pr:hasInput ?s2;}} \]

(b) The following query is an extension of the previous one that supports the indirect intra-PM dependency, using the repeat operator plus:

\[ ?s1 + ( pr:hasOutput. - pr:hasInput) ?s2; \]

(c) The following query is an extension of the previous one that supports the generalized alignment relationships:

\[ ?s1 + ( pr:hasOutput. ?i owl:equivalentClass | rdfs:subClassOf). - pr:hasInput) ?s2; \]

Fig. 4. Different inter-PM dependencies

On this basis, the new query \( Q_0' \) supports the IOs schema-based alignment as a disjunction of equivalence / subsumption relationships regardless the vocabulary of the ontology used where link = ( owl:equivalentclass | rdfs:subClassOf ) and expressed as follow:

Select distinct ?sj, ?si Where {


As a pre-selection result, an ordered set of pair-wise PM denoting the initial, final and the intermediate services with no repetition by the condition \( (?si ! = ?sj) \) that will be generated as a M-layer graph. The complexity of determining all the composition solutions using path expression belongs to the NP space.

4.3.3. Constructing the PM Composition Graph

To find the optimal APs composition plan, we need to provide only one set of simple web services (atomic processes) with high combining ability by avoiding unused and unmanageable solutions. A set of PM was obtained that can be grouped as a set of PM cluster according to their common semantic functionalities and further transformed to M-layer graph \( L \) where a PM cluster constitute an individual graph layer and one PM could be linked semantically to one or several other PM and so, can be executed in parallel or sequencing. The layer 0 consists of a set of initial PM clusters and the final layer consists of a set of final PM clusters. For each layer \( (i > 0) \), \( L_i \) consists of a set of PM clusters that depends directly from \( L_{i-1} \) and constructed as a union of the \( L_{i-1} \) set. The construction of the M-layer ends either when the final PM clusters are reached. The general expression for any \( L_i \) can be defined as follows:

\[ L_i = \{ PM_j : PM_j \notin L_j (j < i) \land O_{\text{in}} \subseteq I_{\text{in}} \} \]

In the figure 5, a PM Service is rectangle and grouped with other PM Services in cluster that are represented as rectangles with rounded corners. It shows that the EventReservationService cluster is inter-connected with three services as they are grouped in two clusters, (1) the first one contains HotelReservationService and AccomodationReservationService, both of them share a common semantic inter-PM dependency parameters such as s:startDate/ s:endDate/ s:City/ s:Country that are equivalent to dbpedia:startDate/ dbpedia:endDate/ dbpedia:City/ dbpedia:dia:Country (see section 4.2) and also the same output parameter (s:lodgingReservation), and (2) the second contains only the WeatherService.

4.3.4. Local Optimization for Atomic Processes Selection

The generated M-layer graph will be used for an efficient selection method targeting the identification of its relevant atomic process children according to the requirements specified in the consumer query. The commonly utilized approach is to optimize locally the PM clusters candidates independently on the other ones.

So, for each cluster, only one service can be considered according to its selected atomic processes subset with higher degree of matching is chosen. The behavior of atomic processes selection is summarized by the pseudocode listed that returns a nonempty result due to the efficient pre-selection phase. It takes as parameters: the generated M-layer graph \( L \), the set \( I_{\text{in}} \) of the inputs to be generated (initially the consumer query outputs, \( O_{\text{in}} \) (line 2)), the set \( I_{\text{in}} \) of the available inputs (initially the consumer query inputs, \( I_{\text{in}} \) (line 1)), the set Comp list of the atomic processes selected so far initially empty (line 3)
3) and the set Sol list of the extracted sub-matrices selected so far initially empty (line 4). After initialization, the AtomSelection algorithm explores the M-graph layer, by performing a visit on each cluster (line 7) of each layer (line 5) and extracts the relevant subset of atomic processes (Service.GetAP()) of each service contained in the currently explored cluster satisfying either all inputs of atomic processes that matches Iav (line 12) by avoiding the unused atomic processes. Then, it computes the degree of matching(Sim$_{atom}$) of the relevant atomic processes for each service (line 15) and returned the most promising (higher matching value) sub-set atomic processes APs according to formula2(line 17).

AtomSelection adds the subset of atomic processes to composition list (line 22), and therefore extract the sub-matrix according to its sub-set that represent a sub-workflow (line 20). Then, it updates the available inputs Iav by adding the outputs of the atomic processes subset (line 18), and updates the required inputs by adding the inputs of the atomic processes subset and removing all the concepts that are now available in the available inputs (lines 19). Then, AtomSelection continues on the next layers. When there are no outputs to be generated, AtomSelection selection returns the set of sub-matrices selected (line 23), which satisfies the functional consumer query. Where a sub-matrix SPM of a given graph APM = (AP, E) is a sub-graph SPM = (AP’,E’) where AP’ ⊆ AP and E’ ⊆ E. Let’s have a process model that contains M atomic processes but only H atomic processes (H<M) matches the available inputs. Semantic similarity Sim$_{AP}$ verifies the compatibility between the inputs of the consumer and inputs of the corresponding atomic process contained in the Process model and avoid the unmatched atomic processes. Sim$_{AP}$ is a score computed of a set of pairs (AP$_{i}$, input$_{i}$) between the input parameters of i$^{th}$ atomic process contained in cluster k from layer j and the available inputs. It measure, to what degree the inputs in the atomic process are used in the available input. We compute Sim$_{AP}$ inspired from [24] with the following formula:

$$Sim_{AP}(AP, I) = \frac{Sim(AP^k input, I^k)}{|AP input|}$$

(1)

Sim$_{atom}$ represent the global degree matching of the selected atomic processes based (1). It corresponds to the sum of Sim$_{AP}$ of each atomic process selected according to its number that is greater than 0 and less than 1.

$$Sim_{atom}(AP, I) = \frac{\sum_{k=1}^{h} Sim_{AP}(AP, I)}{h}$$

(2)

Let us continue with the same example, AtomSelection takes as parameters the graph L, the query inputs (i.e., I$_{in}$=tio:event, foaf:person, cc:creditcard, acc:hotel) and the query outputs (i.e., I$_{out}$=f Eventreservation, lodgingreservation, weatherconditiong), while composition is an empty set. Initially, the first layer contains one service. So, the AtomSelection algorithm extracts three APs (E1, E2 and E3) of the EventReservationService either all its inputs are available and then updates the two following sets with the outputs of the selected atomic processes Inout=[tio:event, foaf:person, cc:creditcard, acc:hotel, dbpedia:startDate, dbpedia:endDate, dbpedia:city, dbpedia:country, s:weather], Inreq=[s:lodgingreservation, we:weathercondition]. Then, for the second iteration, there are two similar services in the first cluster, the need to compute the semantic degree between them is required in order to select the better solution. For the HotelReservationService, it extracts two APs (H2 and H3), then it compute the degree matching that is equivalent to 1 due to the fact that all the inputs of the atomic processes are satisfying by the available equivalent inputs. For the second service, it extracts two APs (A1 and A2) with less degree of matching due to the fact that the star-ranking input of A1 cannot be fulfilled. So, it adds the first service and for the next cluster it adds W1. As a result, it returns the AP composition list with their sub-matrices; Comp=[E1,E2,E3,H2,H3,W1].

4.3.5. Transitive Closure Composition

We present an efficient and effective algorithm that generates dynamically the logical order of the set of atomic processes PM providing from different services that will be executed at run-time. As a result of previous section, a set of sub-matrices needed to be merged in any order to form a new square matrix, the composition dependency matrix (CMD); CMD = $\bigcup_{i=1}^{n}$ SPM$_{i}$. For instance, let’s have two sub-matrices SPM$_{1}$ and SPM$_{2}$ where the vertices of each sub-matrix are defined as below V (SPM$_{1}$)=[AP$_{11}$, AP$_{12}$ ..., AP$_{1k}$] and V
The new matrix CMD generated merges SPM\(_1\) and SPM\(_2\) where: V (CMD) = {AP\(_{11}\), AP\(_{12}\), ..., AP\(_{1N}\)}, AP\(_{1k}\) exists for every vertex AP\(_k\) such that (AP\(_k\), AP\(_m\)) and (AP\(_m\), AP\(_k\)) are edges in E. Using the transitive closure matrix, we use Strassen’s algorithm \[25\] that runs in an asymptotic runtime of \(\theta (n^{\log 7})\) or \(\approx \theta (n^{2.81})\), making it asymptotically faster than traditional matrix multiplication, with run-time \(\theta(n^3)\). The result of matrix transitive closure contains a set of paths between each atomic service in the new composite process with nonzero entries on its diagonal by ranking for larger values of each column in the CMD. It generates M-layer graph where the smallest number denotes to the departure nodes, and gradually locates each AP in the work-flow until the final nodes such as the APs present in the same level are executed concurrently but APs in level \(k\) should be executed before APs in level \(m\), where \(m > k\).

As a result, a hierarchical composition graph is generated automatically.

\[
CMD = \begin{pmatrix}
SPM_1 & 0 \\
0 & SPM_2
\end{pmatrix}
\]

Fig. 6. Composition dependency Matrix for the travel scenario

In order to evaluate the performance and the accuracy of the SQUIREL composition results, we created a collection (due to the unavailability of a benchmark) of OWL-S SWSs (process models) by combining and reusing existing semantic web services developed in (1) OWLS-TC 4.0 benchmark that contains 164 travel services describing a simpler composite process and none a complex one but their semantic functionalities refer to different tourism ontologies (2) Brogi'\(^{10}\) collection which lists only 15 OWL-S SWSs (3) The geolocation Jena Geography Dataset (JGD) describes 203 services available at \[^{11}\]. In our experiments, a set of 110 semantic Web services are managed and annotated according to the OWL-S \[^{8}\] specification where some of them contain a simple composite process and others share the same functionality parameters but expressed using different ontologies. For instance, we can create a new one that allow the consumer to choose between a set of atomic processes that produces the same outputs but provided different inputs. So, we group them by using a choice control construct. Our algorithm was implemented using JavaTM JDK 1.7 and experiments were run under windows 7 PC with 64-bit operating system on a PC with an Intel Core 2 Duo E6550 at 2.33GHz and 4.00 GB of RAM. The XAMPP package was installed to create the localhost for deploying the collection of OWL-S SWSs and schema-based alignment expressed using Turtle-based syntax. In our implementation, we use CPSPARQL engine\(^{12}\) that pre-select the OWL-S SWSs. Jena and ARQ to rewrite the query and Strassen algorithms to reduce the complexity of the transitive closure. The testing was conducted on the same machine via the CPSPARQL endpoint. The quality of the solutions is based on the best selected atomic processes children for each process model contained in a specific cluster from each layer. Our algorithm runs over several experimental sets of semantic web services such as the number of services vary from \(n = 4\) to \(n = 10\) containing a set of atomic processes with different input and output parameters. For each run, semantic queries are generated from a randomly selected service.

\(^{10}\)http://www.di.unipi.it/brogi/projects/owls2pnml/owls-repository/index.html

\(^{11}\)http://iserve.kmi.open.ac.uk/datasets/

\(^{12}\) http://exmo.inrialpes.fr/software/psparql/
I/O parameter enabling SQUIREL to generate a distinct set of services for every simulation, we run our engine several times using the same set of services and calculated the average time. The figure 7 (the left one) shows the needed computation time for finding the optimal solution in nano-seconds. It shows a great performance as in all cases the best solution was found depending on the atomic processes children selected according to the number of inputs/outputs described in the query.

Moreover, the discovery time search (in blue) is much longer than the composition process. It enables us to explain that the computation time is not only based on the number of services but also based on the number of intra-PM dependencies when necessary. As the number of semantic web services increase, the composition solution time also grows due to the large number of equivalent combinations of services (PM clusters) that can be generated in each step. The Qsome filter finds all possible solutions with acyclic dependencies but only one solution is taken into account using an optimal algorithm. So, deleting nodes from the M-layer graph is not a simple operation. Moreover, the run-time performance does not include the construction time of the new composite service. This task is left as a future work. The resulting composite service correctness is verified manually in terms of the correctness of the execution order. Additionally, the composition process (in red) shows a great performance by using Strassen algorithm multiplication that reduces the average time response. However, discovery time search can still be reduced by decomposing the relaxed query and suggest parallelism computation for successive matrix multiplication. The figure 8 (the right one) shows the number of the inter-intra atomic processes dependency versus the number of services that was necessary to generate the optimal composition solution and performed using local optimization technique that meets fine-grained the initial request with no human intervention. It shows that the maximum length of a single workflow could contain 40 atomic processes.

VI. RELATED WORK

There are several works related [26,27,28,29,30] to the automated OWL-S Service Composition using AI-planning that translates the OWL-S process models to planning domain and the query service to a planning problem where the initial state that corresponds to the required inputs and the achieved goals that reflect the desired outputs. The both entries are submitted to a specific planner who creates a plan solution that contains a set of ordered actions that is converted to a composite service executed by OWL-S API. Other approaches [31, 32, 33, 34] treat the problem of composition as a dependency graph (tree) of services and applied a meta-heuristic search algorithm in order to extract the optimal composition scenario but none of them improve the performance in front of large amounts of services due to the redundant services. For that, [35,36] adds a set of dynamic optimization techniques over the A* search algorithm. However, they treat the composition problem on simple composite processes and none on a complex process. However, very few approaches treat the problem on complex process model. We list [37] that employ an offline pre-computing semantic matching cross different ontologies to determine the dependencies within/among atomic services as well as the relationships among concept ontologies for constructing a hypergraph. A search recursive algorithm is used to analyze this hypergraph in order to discover the sets of services that are candidate to be composed given a client request. Nevertheless, this approach was tested on a small repository (ten services) and start from a predefined workflow. Another approach [38, 39] develops an algorithm that matches an I/O user query to each leaf node (atomic service) by traversing the whole processmodel through its root. If a match is found, it is added to a temporary List since it does not exist on it. If it
required other inputs, it checks the next service to find other atomic processes. Finally, it returns the ordered list of atomic processes that produce the output user expected. However, it takes much time to parse the whole process models. Our composition method does not start from a predefined workflow but from the consumer query and provide a quick access to SWSs that increase the probability of finding potential SWS using SPARQL Query language.

VII. CONCLUSION

In this paper, we have proposed an original method for finding the optimal atomic process composition solution using SPARQL-inspired relaxation approach to improve the dynamic SWSs pre-selection without considering the entire search space. After that, it uses namely an M-layer graph so as to store the set of services as cluster of services plus the schema-based alignment metrics resulting from Linked e-Tourism Data. Therefore, in order to incrementally find the better solution, it employs a local optimization on this generated graph by considering the semantic quality of the atomic processes of each service in a cluster that’s avoiding the local optimum stagnancy problem. Additionally, the proposed approach known as SQUIREL Composition Engine obtains promising results for creating dynamic packaging product but as future work, we intend to test it on more larger and complex set of process models (SWSs) by Parallelizing the computations matrix to better speed up the process of automatic composition.

REFERENCES

SQUAREL: Semantic Querying Interlinked OWL-S traveling Process Models

David Mimoun Malki: is graduated with Engineer Degree in Computer Science from National Institute of Computer Science, Algiers (Algeria) in 1983. He received the Magister degree and the PhD degree in Computer Science from DJILLALI LIABES University of Sidi Bel Abbes (Algeria) in 1992 and 2002, respectively. He was a Senior Lecturer in Computer Science Department at the same university from 1986 to 2002. Currently, he is a Professor at the High School of Computer Science of Sidi Bel Abbes (ESI- Algeria). His research interests include databases, ontology engineering, web based information systems, semantic web (services), Linked Data, Semantic Querying, Ontology Alignment.