Dynamic Web Service Calls for Data Integration

Salima Benbernou and Mohand-Said Hacid
LIRIS- Université Claude Bernard Lyon1,
69622 Villeurbanne, France
{sbenbern,mshacid}@liris.cnrs.fr

***Abstract*— Web Services are considered as a dominant paradigm for constructing and composing distributed business application and enabling enterprise-wide interoperability. A peer to peer architecture provides a decentralized infrastructure in sync with the spirit of the web and that scales well to its size. In this paper, we present a framework enabling implicit i.e. dynamic service calls in data centric Peer to Peer Active XML language. Active XML is a language devoted to the management of distributed data by embedding Web service calls into XML document. The aim of implicit calls is to allow dynamic data sources discovery through dynamic services discovery and composition. Implicit service calls are based on the use of ontologies for describing the domain and functionality of services to call and an Active XML engine for the call evaluation. The evaluation process deals mainly with dynamic service composition. It consists of matching OWL-S descriptions defined in a query with service descriptions in a peer-to-peer network. Such a network is structured in such a way that peers with similar functionalities are grouped together and each peer makes itself acquainted with matching relationships between its inputs/outputs and those of other peers.**

***Index Terms*— Semantic web services, data integration, peer to peer, data discovery.**

I. Introduction

As communication infrastructure advanced, in particular the evolution of Internet technologies, needs for ubiquitous access to distributed information sources increased. Data integration has been extensively studied in the past in the context of company infrastructures. Nowadays, data analysis and integration techniques are becoming more and more prominent features of enterprise and government systems. They offer tremendous opportunities for empowering users and organizations in a variety of application domains including electronic commerce, scientific databases, enterprise information and digital government. Since the Web is becoming a main target, data integration has to deal with its large scale, and faces new problems of heterogeneity and interoperablity between loosely coupled sources.

These issues has been recently addressed in two complementary ways. First, Web services can be viewed as a programming paradigm that extracts and integrates data from heterogeneous information systems by providing interface standards [1]. They can be described, published, located, invoked, and can even operate with other services to form a new composed service over a network. Emerging standards for Web services as SOAP and WSDL simplify the interoperability problem by normalizing the way programs can be invoked over the web. When they are used to manage data on the Web, services bring new features : (1) the discovery of Web services based on their functionality leads to the discovery of data sources that contain expected data (i.e., retrieval of dynamic data sources) ; (2) the dynamic composition of Web services allows to retrieve dynamic data; (3) the invocation of Web services on demand allows retrieval of dynamic data. Our work deals with the integration of the two first issues into a single environment.

ActiveXML (for short AXML) is a language for Web-scale data integration by embedding calls to Web services into XML documents [2]. Active XML allows retrieval of dynamic data by including features in XML documents to indicate the location of the service to be called, and to control three elements: the timing of the service invocation , the lifespan of data and the extensional and intensional data exchange. A service call which explicitly makes reference to a service location is called explicit call. In order to enable dynamic data source discovery and dynamic data retrieval (i.e., when an update on data source occurs) by means of dynamic service composition in Active XML, we introduce implicit service call [3]. By resorting to ontologies, we provide a way to specify service domain and service functionality with Active XML documents. Our framework concentrates on the reasoning issue to locate automatically the data in AXML documents. The work presented focusses on:

1) dynamic data sources discovery
2) dynamic data retrieval by considering of dynamic data sources.

The rest of the paper is organized as follows: Section 2 presents our motivation through an example. Section 3 briefly describes Active XML language. Section 4 presents our framework for embedding implicit service calls within Active XML documents. Section 5 describes an Active XML architecture with implicit calls. The related work is discussed in Section 6. Finally we conclude in Section 7.

II. Motivating Example

In this section we present through an example two main motivations regarding dynamic calls in AXML data integration language. AXML allows retrieval of dynamic

This work is partially supported by the French National Research Agency (ANR) - Program "ACI Masses de Donnes:SemWeb"

data by including syntax features indicating the location of the service to call. A service call indicates explicitly the service location. In order to allow dynamic data source discovery and dynamic data retrieval by dynamic service composition in AXML, we extend AXML with functionalities to allow implicit calls by providing new features taking into account the service category and service functionality based on ontologies.

1) **Dynamic data sources discovery.** Let us consider a scenario where we want to make an inventory of books stored in city libraries. We assume that each library has an Active XML peer with a service offering its own book inventory. The implementation and the outputs of the services can be different.
Now we want to make an inventory of the books stored in all the local libraries of the "GuangZhou" city. By means of explicit service calls, we have to be aware of locations of all relevant services and then invoke an explicit service call. Figure 1(a) shows an explicit call for book inventory. A drawback with this method is that it is not resilient to changes. If Web services locations change, then we have to manually encode the changes (by modifying service calls).
On the contrary, with implicit service calls, it is sufficient to be aware of the service domain (*service category*) of the offered data (inputs of service) and of the data that we expect to be returned (outputs of service). In our example, the required service belongs to the *Book* domain, it has no data offered but a list of books is expected as output. Figure 1(b) shows an implicit call. When it is decided to activate this implicit service call drawn up by using these descriptions, the evaluation of the required service location is launched and terminates after a period of time. First, the query is propagated to its owner peer's master that shall decide the candidate master peer which contains the registry of services belonging to the same domain as the required service. In our example, the candidate master has a registry of services of the domain book. When the query is received, it compares the query with the service in its registry. The comparison is semantic-based founded on semantic description of outputs and inputs. If some services are found, the master sends their locations to the invoker. If not, which means that no existing service matches the query and a process of dynamic composition may be launched. The search terminates after a period of time. Then, the user can decide which discovered services (s)he would like to invoke later. Finally, the chosen services are invoked. As a result, we obtain the book inventories of the cities in spite of the dynamic aspect of the data sources. The other motivation of implicit service calls is that we can invoke the relevant service without any knowledge regarding its location.
2) **Dynamic data retrieval.** We want to build up a personal Portuguese-Chinese dictionary. With explicit service call, we need to be aware of the Portuguese-Chinese dictionary service location and invoke the service. In the case a Portuguese-Chinese service does not exist, while two other dictionary services – Portuguese-English and English-Chinese– exist and are locatable, we will not expect an answer to the explicit call. However, with an implicit service call by composing services through input and output descriptions, the call will return an answer. Instead of describing how to obtain the data, an implicit service call describes the domain, the inputs and outputs of a required service based on ontologies (here we use OWL-S). In our example, the required services in the *translator* domain have a Portuguese word as input and a Chinese word as output. The evaluation process of the implicit call is performed in the same way as in the previous example.

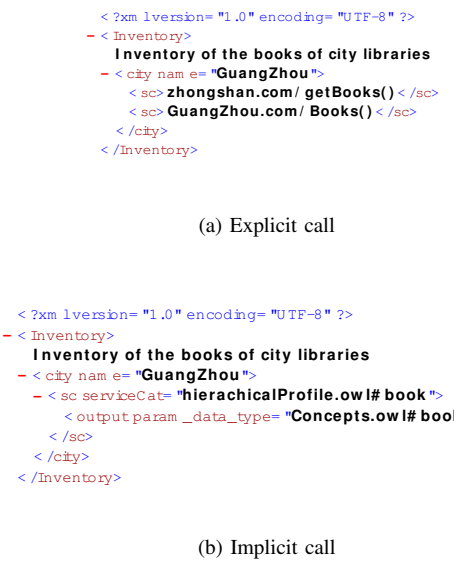

Fig. 1. Type of service calls

III. BACKGROUND

Active XML is a declarative language for distributed information management and an infrastructure to support the language in a peer-to-peer framework. It has two fundamental components: Active XML documents and Active XML service [2], [4], [5].

ActiveXml document. Active XML documents are based on the simple idea of embedding calls to Web services within XML documents. An XML syntax is defined to denote service calls and the elements conforming to this syntax are allowed to appear anywhere in an Active XML document. The presence of these elements makes the document intensional, since these calls represent some data that are not given explicitly, but intensionally, by providing means to acquire the corresponding data when

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Inventory axml:docName="Inventory" xmlns:axml="http://www-
  rocq.inria.fr/verso/AXML">
   <publisher>Addison-Wesley</publisher>
   <publisher>Morgan Kaufmann Publishers</publisher>
  - <books>
   - <book year="1999">
      <title>The Economics of Technology and Content for Digital TV</title>
    - <editor>
       <last>Gerbarg</last>
       <first>Darcy</first>
       <affiliation>CITI</affiliation>
      </editor>
      <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
    </book>
   - <axml:sc frequency="every 3600000" methodName="GetBooksByPublisher"
     mode="replace" serviceNameSpace="GetBooksByPublisher"
     serviceURL="http://lirispbu.univ-
     lyon1.fr:8080/axml/servlet/AxisServlet">
    - <axml:params>
     - <axml:param name="publisher">
        <axml:xpath>../../publisher/text()</axml:xpath>
       </axml:param>
      </axml:params>
    </axml:sc>
  </books>
</Inventory>
```

Fig. 2. Active XML document: Inventory of books

necessary. Active XML documents may also be seen as dynamic since the same service called at different times may give different answers if, for example, the external data source changed. So an active XML document is capable of reflecting world changes, which means that it has different semantics at different times. Figure 2 is an example of an Active XML document that represents databases of books. This document contains some extensional information such as records of the publishers and one record of a book *The Economics of Technology and Content for Digital TV*, and at the same time some intensional information: a service call to get the books published by the *publisher* described by Xpath.

Service call elements in ActiveXML. The Service Call *(sc)* element is defined in the special namespace mentioned above and has a set of attributes and children XML elements defining:

- The Web service to call which is defined by serviceURL, serviceNameSpace, methodName, and useWSDLDefinition.
- The attributes that provide information on how and when to invoke the service call.
- The attributes that influence the behaviors imposed on the results.
- Parameters that are accepted by the Web service.
- frequency states when the Web service should be instantiated and the validity of the returned results.

Frequency attribute has two modes: (1) *immediate* mode, means that service calls have to be activated as soon as they expire (2) *Lazy* mode, means that a service call will be activated only when its result is useful to the evaluation of a query or when the instantiation of a service Call parameter, defined through an XPath expression, is necessary. The presence of lazy calls may cause dependencies among call activations.

According to the expression of parameters, we distinguish two kinds of service calls: (1) *a concrete*service call is one whose parameters do not include XPath expressions, (2) *a non-concrete* service call is one whose parameters do include at least one XPath expression.

Service call evaluation. The notion of *task* is introduced to track the evaluation of each particular service call. Since the service call can be concrete or non concrete, tasks can be concrete or non-concrete. There are two types of evaluation for each invoked mode:

- Service call with immediate mode.A service call has to be activated as soon as it expires. The evaluation is done, first, by selecting the service calls, thereby processing the selected service call;
- Service call with lazy mode, a service will be activated only when its result is useful to the evaluation of a query or when the instantiation of a Service call parameter, defined through a XPath expression, is necessary.
  The presence of lazy calls may cause dependencies among call activations. Before instantiating its XPath parameters, we may need to activate some lazy calls, that may affect the result of the instantiation. The dependencies are described by means a graph called *dependency graph* . The evaluation process can be performed according to the query, first evaluating the dependencies between calls through a dependency graph. In this case, it is necessary to know which call is affected by some updates in a node. Second, selecting the service that can be activated according to the attribute frequency and dependency graph. Finally processing the selected service.

The active XML peer. The system which stores and manages the AXML documents is called Active XML peer. It takes charge of the automatic activation of the service calls and the AXML documents' updating. These peers work in a decentralized peer-to-peer environment. An ActiveXML peer has essentially three roles:

- A repository for AXML documents
- A client. An AXML peer can invoke the corresponding service calls to make use of the intensional data that can be found in AXML documents in the repository.
- A server. An AXML peer may also provide Web services in the form of queries over the documents in its repository for other peers.

The internal architecture of an AXML peer (a part is depicted in Fig 5 includes the following components: (1) *the repository*: it stores the activeXML documents, (2) *the evaluator*: it is responsible for the activation of the service calls embedded inside AXML document and for the update, (3) *the Xquery processor*: it deals with the service requests by evaluating the corresponding queries. Peers communicate with each other only by means of Web service invocations, through their SOAP *warpper* modules. They can exchange XML data with any Web service client/provider and exchange ActiveXML data with ActiveXML peer.

IV. Implicit calls in ActiveXML documents

As we have seen previously, the service call defined in Active XML is *explicit* since the service to call is indicated explicitly in the element *axml:sc* by a set of attributes that specify "*the service to call*". It requires a user to be aware of its exact location. However, we expect to call a relevant service by its description (service query), i.e. implicit service call, instead of its identification (location). In order to perform an implicit service call, we have to know how to integrate the automated service discovery and composition [6] [7] in Active XML. First, we describe how to add semantic descriptions in the service call and then how to obtain the query based on these descriptions that are used for the service discovery and composition. Then, we describe how to answer a query by peer-to-peer composition in a network. Finally, we present how to evaluate implicit service calls.

A. Implicit calls and OWL-S queries

An automated discovery or composition of services requires the semantics of its functionality, especially supported by the use of ontologies, in the description of the service. Consequently, one way to make the implicit service call possible in AXML is to annotate the description of service call by ontologies. The main issue here is to determine the way to annotate. Which attributes of a service need to be described for the search of appropriate services? Description of inputs are necessary since they are needed to invoke the service. Otherwise in the AXML context, the description of the output is also required. In some sense, an AXML document can be seen as a (partially) materialized view that integrates plain XML data and dynamic data obtained from service calls. This means when a service is embedded in a document AXML, we are aware of the data expected to be returned by the service call. So the description of the output plays an important role in the discovery of relevant services. An existing or composite service matches a service query when their respective output and input match. In the second thought, the description of the service domain (service category) is important for the reduction of search space. Then we describe the service query by its *domain*, *outputs* and *inputs*.

Query representation We suggest to represent the query in the form of the service profile description by using OWL-S. The benefit of this representation is that service discovery can be done by performing a matching between service profiles. Furthermore, the precondition and effect descriptions in OWL-S make possible the extension of the composition capability. Here, we focus on the collection of inputs and outputs for composition.

How to annotate the service call? To match the service queries described in AXML with the offered service described by means of OWL-S, we need to annotate the service call with globally shared concepts. Figure 3 shows the structure of an implicit service call which is different from the explicit service call in two respects:

```
<?xml version="1.0" encoding="UTF-8"?>
<Inventory axml:docName="Inventory"
 xmlns:axml="http://www-rocq.inria.fr/verso/AXML">
  <publisher>Addison-Wesley</publisher>
  <publisher>Morgan Kaufmann Publishers</publisher>
  <books>
    <book year="1999">
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
        <last>Gerbarg</last><first>Darcy</first>
        <affiliation>CITI</affiliation>
      </editor>
        <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>
    <axml:sc serviceCat=
     "http://lirispbu.univ-lyon1.fr/services/hierarchicalProfile.owl#Book"
     frequency="every 3600000" mode="replace" >
       <axml:params>
         <axml:param name="publisher"
          param_type=
          "http://www.daml.org/services/owl-s/1.1/Process.owl#Input"
          param_data_type=
          "http://lirispbu.univ-lyon1.fr/services/Concepts.owl#publisher">
            <axml:xpath> ../../publisher/text()</axml:xpath>
         </axml:param>
         <axml:param name="booklist"
          param_type=
          "http://www.daml.org/services/owl-s/1.1/Process.owl#Output"
          param_data_type=
          "http://lirispbu.univ-lyon1.fr/services/Concepts.owl#booklist">
            <axml:value />
         </axml:param>
       </axml:params>
    </axml:sc>
  </books>
</Inventory>
```

Fig. 3. Active XML document with implicit service call

1) The implicit service call does not specify the attributes (*serviceURL*, *serviceNameSpace*, *methodName*, *signature*, and *useWSDLDefinition*) that identify the service to be called, but a new attribute *serviceCat* allows to specify the domain of a service. In our example, the domain of the query is `http://lirispbu.univ-lyon1.fr/services/hierarchicalProfil.owl#Book`;
2) It adds two attributes *param_type* and *param_data_type* to the *param* element. *Param_type* specifies the type (Inputs, Outputs) of a parameter. *Param_data_type* describes the class the values of the parameter through a concept belong to. In our example, we want to call a service that provides a list of books based on the publisher's name. The implicit service call is defined as having two parameters:
   (1) *publisher* being the input of the service whose value is of type `http://lirispbu.univ-lyon1.fr/services/Concepts.owl#publisher`;
   (2)*booklist* being the output of the service whose value is of type `http://lirispbu.univ-lyon1.fr/services/Concepts.owl#booklist`.

B. Data model for implicit service calls

In this section we introduce the data model and the semantics of AXML documents and services.

Intuitively, an AXML instance consists of a number of peers, each one containing some AXML documents that are being run. AXML are unordered trees. The evaluation of these documents generates calls between these peers and possibly results in new documents being evaluated at each peer.

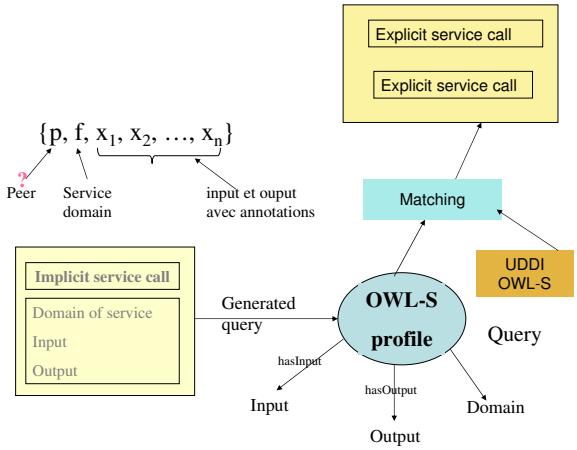

Fig. 4. Query with implicit service call

Instance. An instance $\mathcal{I}$ consists of a number of peers $p_1, ..., p_n$. The content of a peer p_i is defined by a triple $(\mathcal{R}, \mathcal{F}, \mathcal{W})$, where $\mathcal{R}_i$ the peer's repository, is a set of persistent AXML documents, $\mathcal{F}$, the peer's services, is a set of AXML service definitions, and $\mathcal{W}$, the peer working area, is a set of AXML temporary documents. All these sets are assumed to be finite.
Each document d in the working area $\mathcal{W}$ of a peer p_i represents the computation of some service call in p_i.

Documents. AXML is modeled by a labeled tree with nodes representing the document elements/attributes and the edge represent the component of relationships among documents. Some of tree leaves are special implicit service call nodes and is defined by a tuple $< p, f, x_1, ..., x_n >$,

- p : the peer that contains the expected service. It has to be evaluated by Active XML. Initially, it has NULL as default value since we do not know which service will be invoked.
- f : the domain of the expected service.
- $x_1, ..., x_n$: the inputs and outputs annotated by concepts of the expected service.

Service query. Based on the description of the implicit call, a query represented as an OWL-S profile description [8], [9] is generated for service discovery and composition. It is depicted in Fig4. The benefit of this representation is that the service discovery can be accomplished by performing matching between service profiles.

C. Peer-to-peer composition for query answering

Once the query is formalized with OWL-S profile, the discovery and composition tasks can take place.

1) **The choice of the peer-to-peer composition**
There are two computing types for service discovery and composition: centralized computing [10]–[13] and distributed computing [14]–[16].

In the first, a centralized registry exists; every Web service coming on line advertises its existence and eventually its functionalities and thereafter, every service requester has to contact the registry to discover a particular service or to compose services and gather information about them. Whereas such a structure is effective since, it guarantees the discovery of services it has registered, it suffers from problems such as performance bottlenecks, single points of failure, and timely synchronization between the providers and registries (i.e. by updating the changes of service availability and capabilities) [14].
Alternatively, distributed computing allows the registry to be converted from its centralized nature to a distributed one. In the current Active XML context, each peer in the network provides the other peers with its own data as Web services using XQuery queries raised over the Active XML documents in their repository. Hence, changes are frequent and numerous in the service availability and functionalities in an Active XML peer. Furthermore, we envision that the number of implicit service calls is enormous. As we have seen previously, centralized computing is not suitable for such a situation, while the distributed computing can resolve the availability, reliability and scalability problems in this environment.

2) **A composition network**
In order to reduce the complexity of the peer-to-peer composition, we suggest to compute it in a network, structured into two dimensions based on the one proposed in [16]. In this network, each peer can provide some Web services dealing with particular domains. The peers that provide services for the same domain are grouped together. Each peer is a member of at least one domain. Each domain has both *a master peer* and *a backup peer*. The master peer in each domain maintains two lists: (1) the list of master and backup peers of other domains and (2) the list of all peers within the master peer domain together with the services they provide as well as the input and output parameters they accept and generate respectively. The backup peers have a replica of these lists. Furthermore, each peer maintains its master, backup peer and the predecessor-successor lists for its respective services. A predecessor of a service means the outputs match the inputs of this service, while a successor of a service has the inputs matching the outputs of this service. So, discovery of peers that can participate in the composition through these predecessor-successor relationships, starts from the peer(s) providing the query's outputs, up to those accepting the inputs (provided by the query) required for the composition.

3) ***SearchService*: The peer-to-peer composition**

structure

A peer-to-peer composition service component in ActiveXML system, namely *searchService*, should be defined in order to achieve the service discovery and composition task for implicit service call in the network described previously. Its structure is based on WSPDS [15]. WSPDS (Web services peer-to-peer discovery service) is a distributed discovery service implemented as a cooperative service.

SearchService is composed of two engines: the communication engine and the local query engine. Figure 5 depicts the proposed structure for *searchService*:

a) **The communication engine**: It provides the interfaces to the Active XML *evaluator*, to the user and to the other peers. It is responsible for the following tasks:
- Receiving service queries from *evaluator*, answering the queries by local query (through the local query engine) and global query (via the other peers) based on the query phase, merging the different answers in order to allow the user to choose the services (particular or composite) to be invoked, and finally delivering to the *evaluator* the list of locations of chosen services;
- Receiving queries from the other peers in the peer-to-peer network, resolving the queries by local query engine, and sending the response to the caller as well as forwarding to the candidate peers the query whose lifetime is not yet over ($TTL > 0$). The parameter *TTL* (Time To Live) is used to restrict the dissemination of a query in the network and to control the depth of the composition. For example, we can suppose the value for TTL to be 7, and then the query can be propagated in the network with only a depth of 7.

b) **The local query engine**: It answers the query received by the communication engine to the local peer. It contains three modules: *ServiceCat*, the *Outputs* and the *Inputs* which are respectively responsible of the service domain, outputs matching, and inputs matching between the OWL-S profile description of the query and those of existing services.

Most parts of the matching algorithm will be built on the semantics of the services. The semantics themselves are described by ontologies. So the matching between the services can be reduced to the matching between the concepts that describe the services including inputs and outputs.

Algorithm 1 deals with output matching. A matching is recognized if and only if for each output of the query, there is a matching output in the advertised services. The success depends on the degree of matches. If one of the query's output is not matched to any of the advertisement's outputs, the match fails. The matching between inputs is performed the same way, but with the query-advertisement order reversed: the query's outputs are matched against the advertisement's outputs, the advertisement's inputs are matched against the query's inputs

4) **Composition algorithm used by *searchService***
The algorithm 2 describes the process of discovery and composition in *searchService*. When a peer's *searchService* receives the query from its *evaluator*, it forwards the query to the master in its domain, communicates its master's response with the user and returns the list of compositions selected by a user to the *evaluator*. The master of the initiator peer determines the candidate domains for the query and then relays the request to the master peers of these domains It orders the compositions by the matching degree and returns the result to the initiator, when the master peers return the result. To respond to the query, the masters then consult their proper *Query DB* to find whether some of the existing queries match this query. If such queries exist, an answer is sent. Otherwise, they search in their *Peer DB* to determine which services in their domain provide all the expected outputs of the query and transmit the query to the hosts of these candidate services. When these host peers return the list of compositions, the master peers compute the matching degree for each composition based on the output matching degree, the input matching degree and the number of its components. Then the master peers update their own *Query DB* and return the list of compositions. To answer the query, the host determines whether the service requires inputs that can be provided by the query inputs. If they match, the host adds the service to the list of compositions. Otherwise, it relays this query to the peer providing the predecessor of this service and waits for an answer from its predecessor peer.

D. Evaluation of implicit service calls

We have seen that in the case where an Active XML document contains the service call in a lazy mode, the service call evaluation consists in three steps: (1) evaluating a dependency graph for each non-concrete service call; (2) selecting the service call that can be executed based on the *frequency* attribute and the dependency graphs; and (3) processing the selected service.

However, for the implicit call, the evaluation of the service location is necessary. Then in the evaluation process, the third step deals with the evaluation of the service location and processing of the selected service.

Algorithm 3 describes the processing of an *implicit non-concrete* task *t*. A local process *queryGenerator* that takes the parameters annotated as inputs will produce a query based on OWL-S profile description and will return

Algorithm 1 Algorithm for Outputs matching – $Match\_Outputs$

Require: Q_{out} – list of query's outputs
S_{out} – list of service's outputs
Ensure: tsMatchDegree
$outputsMatchDegree \longleftarrow 0$
for all output $Out_Q \in Q_{out}$ **do**
Finding an output $Out_S \in S_{out}$
if $\nexists Out_S \mid match\_Concepts(Out_Q.Co, Out_S.Co) \neq disjoints$ **then**
the service does not offer this output demanded, return $outputsMatchDegree \longleftarrow disjoints$
else
$outputsMatchDegree \longleftarrow outputsMatchDegree + match\_Concepts(Out_Q.Co, Out_S.Co)$
end if
end for
return $outputsMatchDegree \longleftarrow outputsMatchDegree / |Out_Q|$

Algorithm 2 sketch-Composition Algorithm – $searchService$

Require: LQD – Location of Query in OWL-S profile Description
QP – Query Phase: $toMaster, choiceMaster, choicePeer, choiceComponent$
TTL – Time To Live
Ensure: $SLLD$ – Service (composite or simple) Location List with matching Degree
if Query comes from the evaluator, i.e. QP = toMaster **then**
Transmit this query with choiceMaster phase to its master and communicate the result($SLLD$) returned with the user
else
if QP = choiceMaster **then**
Transmit this query with choicePeer phase to the masters whose services are in the same domain of the query
Fusion the results($SLLD$) received and range the services in the results($SLLD$) based on their matching degrees
else
if QP = choicePeer **then**
if $\exists query \in QueryDB$ is similar to this query **then**
Return the results of the similar query as the responds
else
Transmit the query with choiceComponent phase to the member peers that provides the services whose outputs match those of the query
Calculate the matching degree for each composition returned and add the composition returned in $SLLD$
Save the query with the results($SLLD$) obtained in its Data bases of query
end if
else
Reduce the TTL of the query
if the inputs of candidate service match those of the query **then**
Generate a composition that contains the service matched with its matching degree and add it to the local composition list $SLLD$
else
if $\exists predecessors$ for the candidate service and the TTL $<>$ 0 **then**
Transmit this query to its predecessors
Add the candidate service to the compositions in the list $SLLD$ returned by its predecessors
end if
end if
Fusion the local composition list with those returned by their predecessors
end if
end if
end if

the address of the query. A local service *searchService* takes as parameters the location of the OWL-S query, the query phase (QP), the TTL and the service name as inputs to achieve this task. Then, it returns the locations of services fulfilling the query. When the evaluation of the service location is completed, the XPath parameters that are not annotated as "output" of the service call are evaluated. Once the evaluation is done, each p_i has the

Algorithm 3 peer P, implicit non-concrete task $t(d, P_f, f, p_1, p_2, ..., p_n)$

if $P_f = NULL$ **then**
  $LQD \longleftarrow queryGenerator(P_f, f, p_1, p_2, ..., p_n)$ – Location of OWL-S Query Description
  $QP \longleftarrow toMaster$ – Query Phase
  $TTL \longleftarrow 7$
  $servieN \longleftarrow NULL$ – service Name
  $SLL \longleftarrow call\ local\ service\ searchService(LQD, QP, TTL, serviceN)$. – *Services' Location List*
else
  $SLL \longleftarrow (P_f, f)$
end if
evaluate the XPath parameters $p_1, p_2, ..., p_m$ – the parameters annotated as *input*.
for all $p_i \in (p_1, p_2, ..., p_m)$ **do**
  let f_i be the value obtained for x_i (an AXML forest)
end for
for all $(P_{f'_1}, f'_1), ..., (P_{f'_t}, f'_t) \in LSL$ **do**
  for all $x = x_1, x_2, ..., x_m \in f_1 \times f_2 \times ... \times f_m$ **do**
    create $t_x(t.root, P_{f'_1}, f'_1, (t.root, P_{f'_2}, f'_2, (...(P_{f'_t}, f'_t, x_1, x_2, ..., x_m)...)))$
    insert t_x in W
  end for
end for
suspend until all t_x finish

value of an Active XML forest f_i. Then the implicit non-concrete service call is unrolled into explicit concrete calls. Each service candidate has to be called and takes as parameters each element in the cartesian product of the forest *f*. The processing of *t* will end when all these concrete tasks complete their execution. Similarly, the processing of a concrete call can be adapted to accomplish the processing of implicit concrete calls.

V. Architecture

In this section, we propose a new architecture for Active XML in order to take into account implicit service calls. Figure 5 depicts the internal architecture of Active XML with implicit service calls. We add two new components to the initial structure:

1) ***searchService***. It contains two components: the communication engine and the local query engine. It is in charge of the reception of the query from the *evaluator*.
2) The ***storage***.
   It maintains the components describing its own peer. Each peer in the network contains two components in the storage:
   - *Description of services* is a registry of OWL-S descriptions of the services provided by the peer. These service descriptions will be compared to the service query by the local query engine.
   - *Process DB* is a database maintaining the predecessor-successor relations dealing with the services provided by the peer. The directed graph with input/output compatibility provided by *Process DB* can reduce the computing complexity of the composition.

   The master peers and backup peers contain three additional components in the storage:
   a) *Peer DB* contains the peers providing the services of the community presented by the master. It serves as the service registry of a particular domain.
   b) *Master DB* contains the master peers and backup peers of the other domains. This database is necessary for the query propagation between different domains. When the master peer of the query owner receives the query, it sends the query to those master peers that are in its Master DB and whose domain matches the query domain.
   c) *Query DB* maintains the query, together with its solution. It allows to reuse the solution of a similar query, simple or composed, to answer other queries.

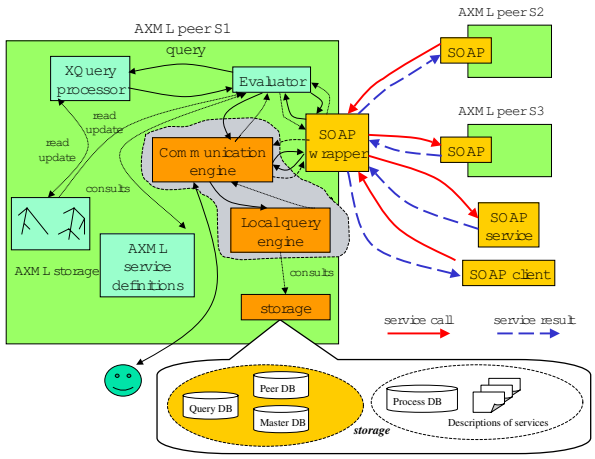

Fig. 5. Architecture for Active XML with implicit service calls

VI. Related work

There are many works related to P2P based approach for Web services discovery and composition. In this section we review the main approaches.

In [17] a Hypercube ontology-based P2P system that focuses on the discovery of Web services is presented.

In [18] the Speed-R system is described that makes use of ontologies to organise Web service discovery registries is proposed. It also addresses the scalability of the discovery process.

In [19] a peer to peer indexing system and associated P2P storage that supports large-scale, decentralized, real-time search capabilities is developed. It supports complex queries containing partial keywords and wildcards.

In [20], the authors proposed a Web services integration platform based on semantic Web and P2P. The platform can automatically find Web services based on specific domain ontology. Web services can be annotated and transformed into OWL-S with richer semantic information. Based on user's requirement, SEWSIP can perform Web service discovery, selection and composition in a semi-automatic way. The integrated Web services are transformed into BPEL for the execution of the services.

[21] presents a workflow engine that supports runtime look-up of service endpoints based on a a P2P middleware. Using a service identifier based on a DHT identifier, service proxy objects that encapsulate port information are downloaded over the structured P2P network from the host where the service is deployed. A service proxy delegates service invocations to an abstract protocol adaptor framework that uses dynamic invocation mechanisms to provide a protocol-independent execution of remote services, e.g., GIOP/IIOP or SOAP.

In [22]a self-serv project is described. The proposed framework uses a P2P based orchestration model to support the composition of multienterprise Web services.

The framework we proposed puts the interactions between peers at the core of data model, through the *use* of service calls. It allows peers to play different roles, and it does not impose strong constraints on interaction patterns between peers, since they are allowed to define and use arbitrary Web services. Moreover, AXML is not a framework for service composition, but for data integration using Web services.

VII. Conclusion

The service calls embedded in XML documents in the ActiveXML framework are a powerful tool for data integration. Including support for various integration scenarios like mediation, data warehousing and distributing computations over the Web through the exchange of ActiveXML documents. The exchange is controlled by using schemas for the input and output parameters of Web services. Moreover, the implicit call paradigm we introduced in this paper has some benefits to enable dynamic data source discovery and dynamic data retrieval. The dynamic data is obtained without knowledge on the data location.

By means of peer-to-peer composition service in the network according to the community and predecessor-successor relations, the time to find out the candidate composition is reasonable.

In our framework, we integrated some techniques in the Active XML system:

1) OWL-S is used to draw up the query based on the annotation in the implicit service call.
2) A peer-to-peer composition service is defined to be used in a structured network.

We consider in this paper only the service functionality described by inputs and outputs. More appropriate data source will be found by adding preconditons and effect descriptions in the implicit call. We plan to investigate this issue.

Acknowledgment

We would like to thank He Xiaojun and ActiveXML group (http://activexml.net/) for their help.

References

[1] M. Hansen, S. Madnick, and M. Siege, "Data integration using web services," *MIT Sloan Working Paper*, May 2002.

[2] O. Benjelloun, "Active xml: A data centric perspective on web services," Ph.D. dissertation, Paris XI university, 2004.

[3] S.Benbernou, X.He, and M.S.Hacid, "Implicit service calls in activexml through owl-s," *Service-Oriented Computing - ICSOC 2005, Third International Conference, Netherlands, LNCS 3826 Springer 2005*, 2005.

[4] S. Abiteboul, O. Benjelloun, and T. Milo, "Positive active xml," *In Proc. of ACM PODS*, 2004.

[5] O. Benjelloun, S. Abiteboul, and T.Milo, "The active xml project: an overview," *ActiveXML website, ftp://ftp.inria.fr/INRIA/Projects/gemo/gemo/GemoReport-331.pdf*, 2004.

[6] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara, "Semantic matching of web services capabilities," *in Proceedings of the First International Semantic Web Conference*, 2002.

[7] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated discovery, interaction and composition of semantic web services," *Journal of Web Semantics*, vol. 1, no. 1, Sept. 2003.

[8] T. O. S. Coalition, "Owl-s: Semantic markup for web services," *http://www.w3.org/Submission/OWL-S/*, Nov. 2004.

[9] D. M. et al., "Bringing semantics to web services: The owl-s approach," *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, 2004.

[10] M. Sheshagiri, M. desJardins, and T. Finin, "A planner for composing services described in daml-s," *AAMAS Workshop on Web Services and Agent-Based Engineering*, 2003.

[11] E. Sirin, B. Parsia, and J. Hendler, "Composition-driven filtering and selection of semantic web services," *In AAAI Spring Symposium on Semantic Web Services*, 2004.

[12] E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic composition of web services using semantic descriptions," *In Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003, Angers, France*, Apr. 2003.

[13] B.Benatallah, MS.Hacid, A.Leger, C.Rey, and F.Toumani, "On automating web services discovery," *VLDB journal*, vol. 14, no. 1, 2005.

[14] M. Paolucci, K. Sycara, T. Nishimura, and N. Srinivasan, "Using daml-s for p2p discovery," *Proceedings of the International Conference on Web Services*, 2003.

[15] F. Banaei-Kashani, C.-C. Chen, and C. Shahabi, "Wspds: Web services peer-to-peer discovery service," *International Symposium on Web Services and Applications(ISWS'04), Nevada*, June 2004.

[16] B. Arpinar, R. Zhang, B. Aleman-Meza, and A. Maduko, "Ontology-driven web services composition platform," *e-Commerce Technology, 2004. CEC 2004. Proceedings. IEEE International Conference*, July 2004.

[17] M.Schlosser, M.Sintek, S.Deker, and W.Nejdel, "A scalable and ontology-based p2p infrastructure for semantic web services," *2nd International conference on P2P Computing (P2P'02)*, 2002.
[18] K.Sivashanmugam, K.Verma, R.Mulye, and Z.Zhong, "Speed-r: Semantic p2p environment for diverse web services registries," *http://webster.cs.uga.edu/ mulye/SemEnt/final.html*.
[19] C. Schmidt and M. Parashar, "A peer-to-peer approach to web service discovery," *WWW Journal*, vol. 7, no. 2, 2004.
[20] L. Juanzi, X. Bin, Y. Wenjun, C. Dewei, Z. Po, and W. Kehong, "Sewsip:semantic based web services integration in p2p," *IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)*, 2005.
[21] D.Dahlem, D.McKitterrick, L. WNickel, J.Dowling, B.Biskupski, and R.Meier, "Binding- and port-agnostic service composition using a p2p soa," *Technical report at the The University of Dublin Trinity College Ireland TCD-CS-2006-13*, 2006.
[22] B. Benatallah and et al., "The self-serv environment for web services composition," *Internet Computing*, vol. 7, no. 1, 2003.

Salima Benbernou is an associate professor in the computer science department at the University of Lyon 1, France. Her research interests include Web services, semantic Web and knowledge representation. Dr. Benbernou received a PhD in computer science from University of Valenciennes, France. Contact her at sbenbernou@liris.cnrs.fr.

Mohand-Said Hacid is a professor in the computer science department at the University of Lyon 1, France. His is leading the Database, Knowledge Representation and Reasoning group (http://www710.univ-lyon1.fr/ dbkrr).