Implicit Service Calls in ActiveXML Through OWL-S

Salima Benbernou, Xiaojun He, and Mohand-Said Hacid

LIRIS,University Claude Bernard Lyon 1,
43 bld du 11 Novembre 1918, 69622 Villeurbanne-France
{sbenbern, x-he04, mshacid}@bat710.univ-lyon1.fr

Abstract. In this paper, we present a framework for implicit 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 discovey 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 calls evaluation. The evaluation process deals mainly with dynamic service composition. It consists in matching OWL-S descriptions contained 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.

1 Introduction

Web services can be viewed as a programming paradigm that extracts and integrates data from heterogeneous information systems by providing interface standards [7]. They can be described, published, located, invoked, and can even operate with other services to form a new, composed service over a network. 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 contains expected data (i.e., retrieval of dynamic data sources) ; (2) the dynamic composition of Web service 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 tasks into Active XML framework which is a language for Web-scale data integration by embedding calls to Web services into XML document [1]. 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 calls. By resorting to ontologies, we provide a way to specify service domain and service functionality with Active XML documents.

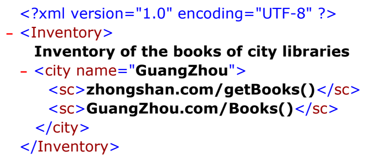
The rest of the paper is organized as follows: Section 2 presents our motivation through examples. Section 3 briefly describes Active XML. Section 4 presents our framework for incorporating implicit service calls within Active XML documents. Section 5 describes an Active XML architecture with implicit calls. We conclude in Section 6.

2 Motivating Examples

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.

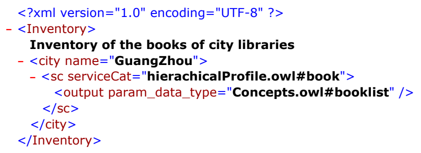
   Now we want to make an inventory of the books stored in all the local libraries of the "GuangZhou" city. By means of explicit service calls, we have to be aware of locations of all relevant services and then invoke an explicit service call. Figure 1 shows an explicit call for book inventory. A drawback 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).

   With implicit service calls, it is sufficient to be aware of the service domain (*service category*) of the data that we can offer (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 2 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. Then the user can decide which discovered services 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 dynamism 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.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <Inventory>
    Inventory of the books of city libraries
  - <city name="GuangZhou">
      <sc>zhongshan.com/getBooks()</sc>
      <sc>GuangZhou.com/Books()</sc>
    </city>
  </Inventory>
```

Fig. 1. Explicit call



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <Inventory>
    Inventory of the books of city libraries
  - <city name="GuangZhou">
    - <sc serviceCat="hierachicalProfile.owl#book">
        <output param_data_type="Concepts.owl#booklist" />
      </sc>
    </city>
  </Inventory>
```

Fig. 2. Implicit call

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 launched in the same way as in the previous example.

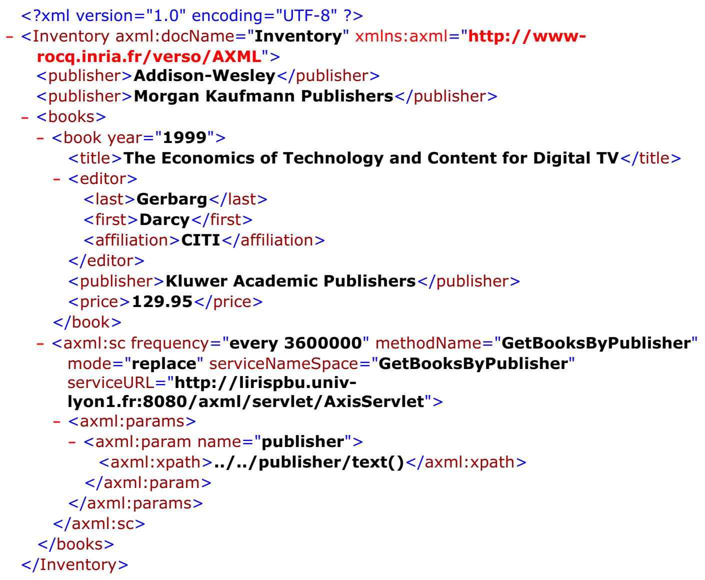
3 Background

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 [1, 13, 12].

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 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 3 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.



```
  <?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. 3. Active XML document: Inventory of books

– 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 is concrete or non concrete, tasks can be concrete or non-concrete. There are two types of evaluation for each invoked mode: (1) service call with immediate mode, where the evaluation is done first by selecting the service calls and processing the selected service call; (2) Service call with lazy mode, where the evaluation is performed by first evaluating the dependencies between calls through a dependency graph i.e. before instantiating XPath parameters, it is necessary to know which call is affected by some updates in a node, and selecting the service that can be activated according to the attribute frequency and dependency graph, and finally processing the selected service call by the algorithm for the non-concrete task.

4 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 realize an implicit service call, we have to know how to integrate the automated service discovery and composition [9] [6] in Active XML. At first glance, we describe how to add the semantic description in the service call and then how to obtain the query based on these descriptions that are used for the

```
<?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. 4. Active XML document with implicit service call

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.

4.1 Implicit Calls and OWL-S Queries

Figure 4 shows the syntax of an implicit service call which is different from the explicit service call in two respects.

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.`

4.2 Data Model for Implicit Service Calls

An implicit service call can be represented 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.

Based on the description of the implicit call, a query represented as an OWL-S profile description [3, 4] is generated for the service discovery and composition. The benefit of this representation is that the service discovery can be accomplished by performing matching between service profiles.

4.3 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, 14, 4, 11] and distributed computing [8, 5, 2].

   In the former case, 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) [8].

   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 [2]. 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 [5]. WSPDS (Web services

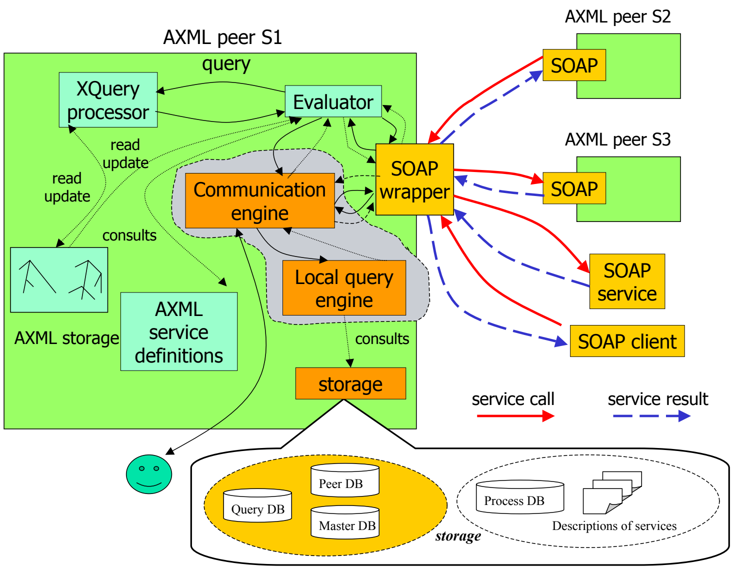


Fig. 5. Architecture for Active XML with implicit service calls

peer-to-peer discovery service) is a distributed discovery service implemented as a cooperative service.

SearchService is composed by 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.

4. **Composition algorithm used by *searchService*** The Algorithm 1 describes the process of discovery and composition in *searchService*. When a peer's *searchService* receives the query from its *evaluator*, it forwards the

Algorithm 1 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
```

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.

4.4 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 the , in the evaluation process, the third step deals with the evaluation of the evaluation of the service location and processing of the selected service.

Algorithm 2 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 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 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.

Algorithm 2 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 $searchService(LQD, PQ, TTL, serviceN)$. – 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
```

5 Architecture

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

1. ***searchService.*** It contains two components: the communication engine and the local query engine. It is in charge 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:
   (a) *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.
   (b) *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.

(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.
(c) *Query DB* maintains the query, together with its solution.

6 Conclusion

In this paper, we have presented the benefits of embedding implicit service calls in Active XML and its realization by the discovery and composition of services. The introduction of implicit service calls in Active XML leads to dynamic data sources discovery by which we can obtain the expected data without knowledge on the data location. To enable implicit service calls, we integrate some techniques in the Active XML framework: (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.

References

1. Omar Benjelloun. *Active XML: A data centric perspective on Web services.* PhD thesis, Paris XI university, 2004.
2. Boanerges Aleman-Meza Budak Arpinar, Ruoyan Zhang and Angela Maduko. Ontology-driven web services composition platform. *e-Commerce Technology, 2004. CEC 2004. Proceedings. IEEE International Conference*, July 2004.
3. The OWL Services Coalition. Owl-s: Semantic markup for web services. *http://www.w3.org/Submission/OWL-S/*, November 2004.
4. James Hendler Evren Sirin and Bijian Parsia. Semi-automatic composition of web services using semantic descriptions. *In Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003, Angers, France*, April 2003.
5. Ching-Chien Chen Farnoush Banaei-Kashani and Cyrus Shahabi. Wspds: Web services peer-to-peer discovery service. *International Symposium on Web Services and Applications(ISWS'04), Nevada*, June 2004.
6. Anupriya Ankolekar Katia Sycara, Massimo Paolucci and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics*, 1(1), September 2003.
7. Stuart Madnick Mark Hansen and Michael Siege. Data integration using web services. *MIT Sloan Working Paper*, May 2002.
8. Takuya Nishimura Massimo Paolucci, Katia Sycara and Naveen Srinivasan. Using daml-s for p2p discovery. *Proceedings of the International Conference on Web Services*, 2003.
9. Terry R. Payne Massimo Paolucci, Takahiro Kawamura and Katia P. Sycara. Semantic matching of web services capabilities. *in Proceedings of the First International Semantic Web Conference*, 2002.
10. Marie desJardins Mithun Sheshagiri and Tim Finin. A planner for composing services described in daml-s. *AAMAS Workshop on Web Services and Agent-Based Engineering*, 2003.
11. Christophe Rey Mohand-Sad Hacid, Alain Leger and Farouk Toumani. Dynamic discovery of e-service. *the proceedings of the 18th French conference on advanced databases. Paris*, 2002.

12. Omar Benjelloun Serge Abiteboul and Tova Milo. The active xml project: an overview. *ftp://ftp.inria.fr/INRIA/Projects/gemo/gemo/GemoReport-331.pdf*, 2004.
13. Omar Benjelloun Serge Abiteboul and Tova Milo. Positive active xml. *In Proc. of ACM PODS*, 2004.
14. Evren Sirin, Bijan Parsia, and James Hendler. Composition-driven filtering and selection of semantic web services. *In AAAI Spring Symposium on Semantic Web Services*, 2004.