A Scalable Relational Database Approach for WebService Matchmaking

Deepak Chenthati
Teradata(R&D)India Pvt Ltd
Hyderabad, India
chvcdeepak@gmail.com

Hrushikesha Mohanty
Department of CIS
University of Hyderabad
Hyderabad, India
hmcs_ hcu@yahoo.com

Avula Damodaram
Department of CSE
JNTU Hyderabad
Hyderabad, India
damodarama@gmail.com

***Abstract* —Web services have a potential to enhance Business to Business collaboration by exposing compatible services to a service of higher dimension. Composition of web services essentially requires matching of their conversation protocols in addition to service requirements. For modeling and matching of conversations, the usability of formal structure like Finite State Machine (FSM) is well studied. However we are in opinion that for storing FSMs in graphs and retrieving them is not scalable for high cost in terms of memory and time. As in near future, as millions of web services will be available this scalability problem is a challenge to research communities. In order to cope up with this problem this paper proposes a relational model to store FSM models and retrieve for service composition.**

Keywords - webservice, RDBMS, matchmaking

I. INTRODUCTION

A confluence of Internet and computing technologies have given rise to a new computing paradigm that host several services for use by remote users over World Wide Web(WWW). These services use SOA: Services Oriented Architecture that not only lists the services offered on web but also provides a mechanism for their interactions. Interactions among services are necessary for making of a service of higher granularity by making use of services of lower granularity. This phenomena of service composition on synthesizing services largely depends compatibility of component services and this leads to B2B: Business to Business collaboration. As in business domain collaboration is largely welcome for greater benefits to collaborating business houses as well as users, match making of services is a problem of prime interest.

Service match making studies the composability of component services and one major aspect of studying this includes matching of conversational protocols of these services. Ideally, a conversation is feasible if for a send (of a message) by a service there is a receive at another service. Then we say both the services are in compatible for the conversation. The study of this conversation compatibility in literature is termed as service matchmaking. There has been considerable volume of research on this problem. The various techniques include ontology based [2][3][4], Fuzzy Logic based[5], Rough Set based[6] and Model based matchmaking approaches.

Model based approach is found interesting for its mathematical elegance as well as understanding. Models tried for analysis, orchestration and choreography of web services include EPC [14][15], Pertinets[16][17], and FSM[7][12]. On reviewing recent works on different modeling approaches we observe that FSM is being studied actively for modeling web services. Particularly for choreography to ensure successful communication among collaboration services. In [1] communication protocol of a web service has been extracted as a FSM and each edge is annotated with a message and its sender and receiver names. In order to ensure the correctness of a protocol between two web services, there must be well-defined FSM obtained due to interconnection of two FSMs that model communication of individual web services. A search of a compatible service for a given service essentially looks for a such well-defined FSMs from the searching and searched web services. Well-defined means the correctness in sequence of message transactions between two services i.e. for each send of a message there has to be a receive and the matching of sequence of sends with the sequence of receive of messages. An implementation of the scheme needs storage of FSMs in UDDI a repository of web services.

Because of wide spread usages of Internet there has been increasing trends in offering services, business applications on web giving rise to numerous web services. Maintaining storage of FSMs as graphs is tedious and difficult to scale up. Considering rise in number of web services available on web, managing storage at UDDI and their retrieval have become a challenge that needs an urgent attention. This paper takes up the issue and proposes a relational model to store FSMs and querying a system that matches to a given service and leads to a composed service.

II. SURVEY

FSM [7][12] has been used to model and match conversation protocols. However, we are in an opinion that storing, matchmaking and retrieving FSMs using graph structure is not scalable for representing conversation protocol for thousands of web services that will soon be available on web. In order to cope up with the challenge we propose relational model to store FSMs and a query system to retrieve services that are matching to a given service.

Another work [11] deals with the problem we address here. The authors view composability of services in terms of matching their input and output parameters. Two services match when one's output parameters match with input parameters of the other. While matching, they have used semantic similarity based in WordNet. Service input output parameters are stored in databases. Being soft in semantic matching, we may get several services matching with a given service but at different degrees. The matching result is represented in a directed graph. Where an edge connects two matching services and the label of the edge notes the degree of matching. Then the service composition problem is seen as shortest path finding problem. Other than that QoS is also considered in deciding weight of an edge. Shortest path finding algorithm is modified to operate on relational repository of service data.

Our work is distinct from [11] by taking message transactions into consideration. We argue, for work-ability of a composed service matching of their message communications is important. In-spite of input output matching, a composition will fail to match in case of their mismatch in communication. While matching we consider the sequence as well as importance of messages (defining types like compulsory and optional). Alike [11] we have used relational database to store communication details in order to make the technique scalable. Message communications of each service are recorded in databases and matching of two services is performed by joining of two tables. The join operation is relaxed to provide results of kinds of matching Full Match, Partial Match and No Match. We have also discussed possible implementation of the proposed technique. We view our technique along with the technique proposed in [11] will give precisely workable service composition. In [13] authors proposed a method in which, web services composition are computed in advance and stored in tables. For web services composition searches, they look up the pre-computed tables rather than actual web services.

Rest of the paper is organized as follows. Section 2 illustrates FSM based approach for modeling service communication and matching of compatible services for service composition. Section 3 describes RDBMS schema to store services and its communication protocol. Also this section presents a new matching algorithm and explains it with an example. Section 4 describes the way RDBMS is embedded into the currently available jUDDI architecture and how it is used for matching of services to compose a larger service. Section 5 concludes the paper.

III. SERVICE COMPOSITION AND MATCHMAKING

A service has business logic driven by business communication. In [7] we have shown how both can be modeled by FSM. Here, we are only interested in modeling business communications by FSM as shown in Figure 1. Each node indicates a state and an edge for communication. A label on edge follows a format sender#receiver#message details. Figure 1 specifies two services name Book store and Shipping and the chain of events follow from ordering of book to shipping it are as follows:

Let us consider the example of Ordering a Book process in an electronic book store. The chain of events will be as follows:

1) Customer(c) Requests for a Book in e-Book store (bs).

2) Book store sends availability confirmation to customer if book is available else it says Unavailable to customer.

3) Customer places an order for the book.

4) Book Store responds with make payment message.

5) Customer makes payment.

6) Book Store sends the payment confirmation.

7) bs interacts with shipping service provider (s) with a request message.

8) The shipper replies to bs with availability information if shipping is available to that particular location.

9) Then bs puts a message to Deliver the book.

10) On receiving Delivered message bs chooses any oneof the modes of payment (Credit Card) CCPaymentand (Debit Card) DCPayment.

11) Shipper sends a payment confirmation message onreceiving the payment.

12) Book Store sends Book Delivered message to customer.

The aim of this paper is to identify service partners whose message exchange sequence matches that of the requesting service. From the figure we see that the Book Store service communicates with consumer and shipper. Here shipper is the service collaborator with whom Book Store service collaborates. The states that are darkened involve communication with that of shipper. Now the aim is to find the shipper service whose message flow matches with that of the Book Store service.

Figure 1 shows the AFSM representation of Ordering a Book process. Here bs#c#Avail Conf, c#bs#Order Book, bs# c#MakePayment, bs#c#PayConf and c#bs#BookDelivered and c#bs#Payment are mandatory messages bs#s #CCPayment and bs#s#DCPayment are optional messages.

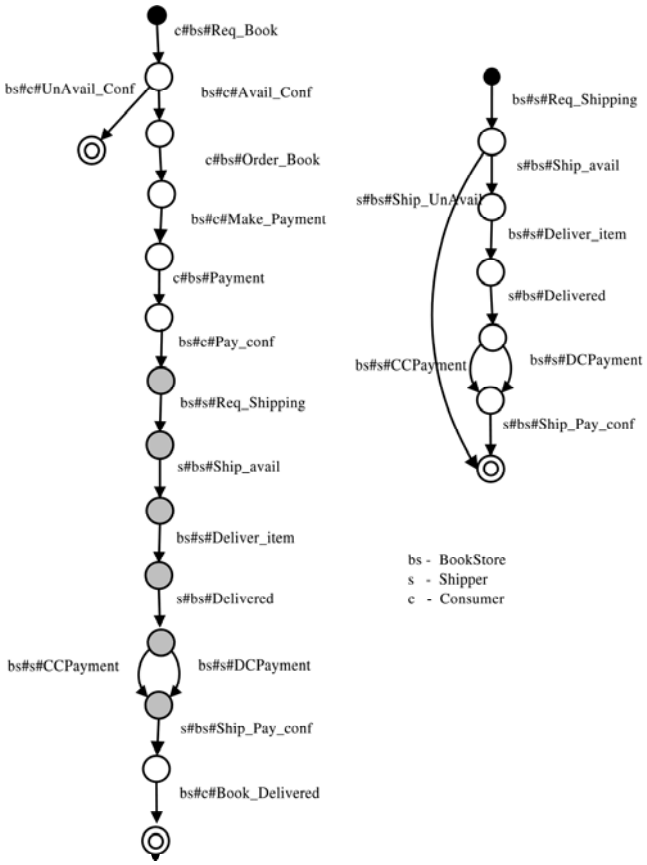

Figure 1. AFSM representation for "Book store" and "Shipping" services

A match between two FSMs is said to have occurred if and only if there exists a non-empty intersection between the two corresponding FSMs. For non-empty intersection, first we will do intersection operation on two FSMs, then we will perform emptiness test on the resulted intersected FSM. If emptiness test fails (i.e., FSM is accepting some string of messages), then it indicates that there is a path from start state to final state in that intersected FSM. So, we say that two FSMs are matched. Drawback of FSM based matchmaking is that storing, matching and retrieving FSMs using graph structure is not scalable when there are huge number of services that are available on the web. Hence, we look for relational repository for this purpose.

IV. A PROPOSED SCALABLE APPROACH

Relational databases have been successful in maintaining and accessing large repository of information. For this problem, we resort to the same technology to store FSMs for each service and to perform matching required for service composition.

A. Service Repository

In the literature, a way of storing web service details and its ontology's using Relational Database was proposed [2] [3]. But it had not taken the message flows into concern. So, we have proposed an RDBMS schema for storing not only messages but also the sequence of communications. Figure 2 shows the E-R diagram of the proposed schema. From the figure, we can observe that a single BusinessEntity can provide many BusinessServices and for every BusinessServices there is a corresponding description about its ServiceMessageFlow. We have also proposed the RequestorMessageFlow table which specifies the format a customer has to send his/her request.

BusinessEntity: A given instance of the BusinessEntity is uniquely identified by a BusinessKey. Simple textual information about the BusinessEntity is given by its name and contacts like e-mail, WebURL.

BusinessServices: A given BusinessServices entity is uniquely identified by its ServiceKey. The BusinessKey attribute uniquely identifies the BusinessEntity which is the provider of the BusinessService. Simple textual information about the BusinessServices is given by its name and short service description and its Category.

ServiceMessageFlow: A given instance of ServiceMessageFlow is uniquely identified by its ServiceKey, msg and followedByMsg. This table stores the message flow of services. Type of message, optional or mandatory, is specified by its type field. snd rcv field specifies whether the message is a sent by customer or received by customer.

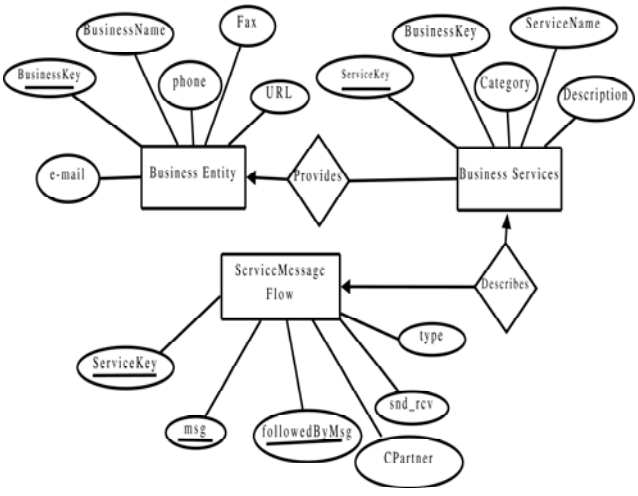

Figure 2. E-R Diagram

RequestorMessageFlow: A given instance of RequestorMessageFlow structure is uniquely identified by its msg and followedByMsg. The table has the message flow of a requested service, which can be given externally or can be generated from the ServiceMessageFlow table. Now, we can

BusinessEntity (BusinessKey, BusinessName, Phone, Fax, e-mail, URL)
Provides (BusinessKey, ServiceKey)
BusinessServices (ServiceKey, BusinessKey, ServiceName, Description, Category)
Describes (ServiceKey, msg, followedByMsg)
ServiceMessageFlow(ServiceKey,msg, followedByMsg, snd rcv, type, CPartner)
RequestorMessageFlow(msg, followedByMsg, snd rcv, type)

Figure 3. Proposed RDBMS Schema Supporting AFSM details

TABLE I. SERVICE MESSAGE FLOW

sk	msg	followedByMsg	snd_rcv	type	Cpart
1	Req_Book	NULL	receive	man	cust
1	Avail_Conf	Req_Book	send	man	cust
1	Order_Book	Avail_Conf	receive	man	cust
1	Make_Payment	Order_Book	send	man	cust
1	Payment	Make_Payment	receive	man	cust
1	Pay_Conf	Payment	send	man	cust
1	Req_Shipping	Pay_Conf	send	man	shipper
1	Ship_Avail	Req_Shipping	receive	man	shipper
1	Deliver_Item	Ship_Avail	send	man	shipper
1	Delivered	Deliver_Item	receive	man	shipper
1	CCPayment	Delivered	send	opt	shipper
1	DCPayment	Delivered	send	opt	shipper
1	Ship_Pay_Conf	DCPayment	receive	opt	shipper
1	Ship_Pay_Conf	CCPayment	receive	opt	shipper
1	Book_Delivered	Ship_Pay_Conf	send	man	cust
2	Req_Shipping	NULL	receive	man	cust
2	Ship_Avail_Req	Req_Shipping	send	man	cust
2	Deliver_Item	Ship_Avail	receive	man	cust
2	DCPayment	Deliver_Item	receive	man	cust
2	Ship_Pay_Conf	DCPayment	send	opt	cust
3	Req_Shipping	NULL	receive	man	cust
3	Ship_Avail	Req_Shipping	send	man	cust
3	Deliver_Item	Ship_Avail	receive	man	cust
3	Delivered	Deliver_Item	send	man	cust
3	DCPayment	Delivered	receive	man	cust
3	Ship_Pay_Conf	DCPayment	send	opt	cust

summarize the tables in the proposed RDBMS schema as described in the Figure3.

Considering the same example *Ordering the book* from section 2. Fig. 1 shows the FSM with annotations and Table I show the schema representation of the corresponding FSM for Book Store with ServiceKey (SK) as 1. We propose 6 column table;Where msg is the message for communication, followedByMsg is the previous message which was received, snd_rcv gives the detail if message is sent or received,there are two types of messages 'mandatory' (man) and 'optional' (op), Communication Partner (CPartner) gives the partner details with whom the service collaborates

Let us take a look at the first row. Here, *followedByMsg* is NULL because there is no preceding message to Book message. And rcv value is *receive* because BookStore(bs) is receiving that message from customer(c) type is set to mandatory. Similarly all the remaining rows in that table are inserted. Number of rows in a table is equal to the number of messages in FSM. Messages *CCPayment* and *DCPayment* are entered into table as optional because, by default if annotated FSM state has two or more messages emerging from it, those messages are treated as optional.

B. Service Matching Algorithm and Analysis

By looking at the example given in section 3, we have observed that a single services message flow spans over multiple rows in a relational table. So, it is not possible for any type of query in RDBMS to perform match operation between two such multiple row spanned message flow sequences. So, we have proposed an algorithm for matchmaking of web services based on above defined tables. This algorithm is also implemented as a stored procedure in MySQL. After matching process is completed, a service falls into one of the three categories namely *Exact match*, *Partial match* and *No match*.

Algorithm IV.1

Input: *servicemessageflow table, ServiceKey(skey) and req_category*
Output: *servicename and type of match*
Algorithm

```
1: match()
2: {
3: Build_RequestMessageFlow Table
4: cnt ← number of tuples in bussinessservices
   where category = req_category;
5: while cnt>0 do
6:    sk ← bussinessservices.servicekey;
      {For each service with unique sk}
7:    create temporary table t1 where service message-
      flow.ServiceKey= sk;
8:    create another temporary table t2 where msg and
      followedByMsg of t1 and RequesterMessageFlow are
      equal which is full outer join table.
9:    full, partial ← 0;
10:   ct ← numberoftuplesint2;
```

```
11:   m1,fm1,sr1,ty1,m2,fm2,sr2,ty2← t2(msg1,
      followedbymsg1,snd_rcv1, type1,msg2,
      followedbymsg2,snd_rcv2, type2);
12:   while ct>0 do
13:     if ((sr1='send' and sr2='receive') or (sr1='receive'
        and sr2='send')) then
14:       if (m1==m2 and( fm1==fm2 or (fm1=NULL or
          fm2=NULL)) then
15:         if (ty1='man' or ty1='op') and (ty2='man' or
            ty2='op') then
16:           full ← 1;
17:         end if
18:       end if
19:     else if (fm1=fm2) then
20:       if (m1 != NULL and m2 == NULL) then
21:         if (ty1='man' and ty2='opt') then
22:           partial ← partial +1
23:         else if (ty1='opt' and ty2='opt') then
24:           full ← 1
25:         end if
26:       end if
27:       if (m1 == NULL and m2 != NULL) then
28:         if (ty1='opt' and ty2='man') then
29:           partial ← partial +1
              full ← 0
30:         else if (ty1='opt' and ty2='opt') then
31:           full ← 1
32:         end if
33:       end if
34:       if (m1!=m2) and m1!= NULL and m2 != NULL)
          then
35:         if (ty1='man' or ty1='opt') and
            (ty2='man or ty2='opt') then
36:           partial ← partial +1
              full ← 0
37:         end if
38:       end if
39:     end if
40:     if ((m1==m2) and (fm1!=fm2)) then
41:       partial ← partial +1
42:       full ← 0
43:     end if
44:     if (m1=NULL and fm1=NULL) and (m2!=NULL
        and fm2!=NULL) and (ty2='man') then
45:       full ← 0
46:     end if
47:     if (m1!=NULL and fm1!=NULL) and (m2=NULL
        and fm2=NULL) and (ty2='opt') then
48:       full ← 0
49:     end if
50:     ct ← ct -1
51:     increment one tuple in t2;
52:   end while
53:   if ( full=1 and partial =0) then
54:     Print ServiceName, "Exact Match";
55:   else if ( full=1 and partial>0) then
56:     Print ServiceName, "Partial Match"
57:   else
58:     Print ServiceName, "No Match";
59:   end if;
60:   cnt ← cnt-1;
61:   increment one tuple in BussinessServices;
62: end while
63: }
```

In case of *exact match*, all the messages of both service provider and requestor message flow sequence matches totally (or) some optional messages of requestor message flow sequence may not be present in message flow sequence of service provider. In case of *Partial match*, some messages of service provider message flow sequence don't have corresponding messages in requestor message flow sequence. In case of *no match*, no messages of service provider and requestor match (or) some mandatory messages of requestor message flow sequence don't have corresponding messages in service provider message flow sequence.

Let us take a look at the steps involved in the proposed matching algorithm. Line numbers at the end of each step refers to the part of the algorithm that is performing that particular step.

- RequestMessageFlow table shown in Table II is created from ServiceMessageFlow table which has the same ServiceKey as given input and whose Cpartner is same as req_Category.
- Temporary table is created for a service from *ServiceMessageFlow* that has matched *Category* with requestor's category. (Line 7,8)
- Another temporary table is created by doing a full outer join operation on the above obtained table and *RequestMessageFlow* table based on their *msg* and *followedByMsg*.
- For each message (row or tuple), if for each send type of message there is receive type of message with the provider service or vice-versa then if the attributes *msg*, *followedByMsg* of both (service provider and requestor) in the above full outer join table are equal and if that is the first message the followed by messages would be null this check is performed and full is set to 1. (Lines 13-15)
- For the messages whose previous messages match different conditions are checked. (Line 19)
- If both message types are equal (or) if service provider has mandatory message corresponding to optional message of requestor then set *full* to 1. (Lines 20-26)
- If there is no message in requestor for the corresponding message in service provider then for

mandatory msg of provider partial is incremented else *full* is set to 1 (Lines 20-25)

- If there is no message in provider for a corresponding message in requestor if the type of requestor is mandatory *partial* is incremented and *full* is set to 0.(Lines 27-29) It is given partial as there is scope for negotiation. If type is optional then *full* is set to 1.
- If there is mismatch in messages then the partial value is incremented and *full* is set to 0.
- Now, if full is 1 and partial is 0 then the services are **'Exactly Matched**' else if *full* is 0 and *partial* >0 then they are **'Partially Matched**' else they are **'Not Matched'**. (Lines 53-59)
- Repeat the above steps for all services that are matched with requestor category. (Lines 5-60)

V. Simulation and Analysis

Let us again consider the same example of Ordering a Book service. The following Table 1 and Table 2 shows two tables: one is ServiceMessageFlow table containing three services (Assuming that only these three services are available and all of them belongs to same category), and other one is RequestorMessageFlow. Let us analyze the matching process of service 2 message flow with requestor message flow. Full outer join table of service 2 and RequestorMessageFlow tables is shown in Table III

TABLE II. Requester Message Flow

msg	followedByMsg	snd_rcv	type
Req_Shipping	Pay_Conf	send	man
Ship_Avail	Req_Shipping	receive	man
Deliver_Item	Ship_Avail	send	man
Delivered	Deliver_Item	receive	man
DCPayment	Delivered	send	man
Ship_Pay_Conf	Payment	receive	man

Initially, full and partial are set to 0. First three messages (rows) of service 2 are exactly matched with that of requester messages. line 14-18 of algorithm handles it. From the join table shown in Table III, There is a mismatch in the fourth message of service 2 and from the join table we can observe that the followedByMsg's are different where as the messages are same. Since there is a skip in the message exchange both the services can negotiate hence lines 41-42 will set the value of full to 0 and increment the partial value. Again there is match in the fifth message hence full is set to 1 but partial value is 2 hence its a Partial Match. Few messages that are specified in the provider service may not be present at the requester's end, and both the requester and provider can negotiate and agree upon the missing message.

In this way our matching algorithm performs matching of each service in *ServiceMessageFlow* table with the *RequesterMessageFlow* table. Similarly, according to the above given algorithm, service 3 matches exactly with requester's one and service 2 has partial match with requester's one. Due to the space constraint we have not represented all the cases. As we told earlier that we have implemented the above algorithm as a stored procedure in MySQL, the way of calling it is as follows: call match();

VI. A New Framework

UDDI [8] uses WSDL [9] to describe interfaces to web services. Some organizations have already implemented UDDI depending on their requirements. One of them is jUDDI[10], which is developed by Apache group and it is open source. We have extended this jUDDI by means of adding *Service communication message flow Database* and *Matching algorithm based on RDBMS* as highlighted in Figure 3. . As we told earlier, Service communication message flow Database stores the message exchange sequences of all services that are registered in UDDI. This database is used by our newly proposed matching algorithm

A. JUDDI Extension

Now, the overall architecture of jUDDI can have the following components.

Publish Service: Service Provider who is interested in providing service will publish their service through this module. They publishes their business information in *Service Database* and service communication message flow details in *Service Communication Message flow Database* as shown in Figure 3

Query Service: Service Requestor who is interested in consuming service sends his/her request through this module. This request is processed by *Matching Algorithm based on RDBMS* component. After the matched services are returned, requestor will contact that particular service provider by getting information from Service Database.

Matching Algorithm based on RDBMS: This component receives request from requestor and performs matching with each service in *Service Database* by means of taking message flow of corresponding service from *Service Communication Message flow Database*.

Service Database: It contains the database schema required for storing information related to business service like Name, email, URL, etc Service Communication

Message flow Database: It contains the database schema required for storing the communication message flow sequence of each service.

Three functionality's are primarily provided by jUDDI namely a) Publish Service b) Query Service and c) Compose service. Service Providers who want to provide their service publishes their details into the Service Database and their message flow sequences into Service Communication

-message flow Database. The organization which requires a service places a query by specifying the type of service it wants. This query is handled by matching algorithm component. The organization which requires service composition sends its required message exchange sequence. This is handled by matching algorithm component and returns the resulted set of matched services by means of interacting with both Service Database and Service communication message flow Database components.

TABLE III. FULL OUTER JOIN TABLE FOR SERVICE 2

Skey	msg	followedByMsg	snd_rcv	type	msg	followedByMsg	snd_rcv	type
2	Req_Shipping	NULL	receive	man	Req_Shipping	Pay_Conf	send	man
2	Ship_Avail	Req_Shipping	send	man	Ship_Avail	Req_Shipping	receive	man
2	Deliver_Item	Ship_Avail	receive	man	Deliver_Item	Ship_Avail	send	man
2	Delivered	Deliver_Item	send	man	Delivered	Deliver_Item	receive	man
2	DCPayment	Delivered	receive	man	DCPayment	Delivered	send	man
2	Ship_Pay_Conf	DCPayment	send	man	Ship_Pay_Conf	DCPayment	receive	opt
2	NULL	NULL	NULL	NULL	CCPayment	Delivered	send	opt
2	NULL	NULL	NULL	NULL	Ship_Pay_Conf	DCPayment	receive	opt

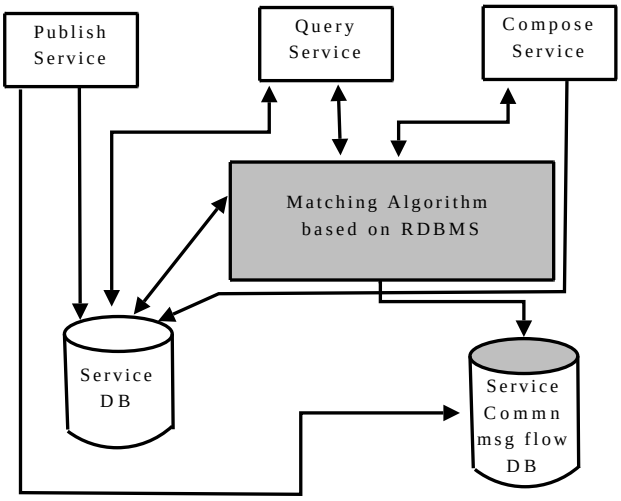

Figure 4. Extended jUDDI Architecture

VII. CONCLUSION

This paper has proposed a scalable approach for matchmaking of web services for composition of services of higher granularity from given atomic services. FSM models for communication for each service is extracted and stored in a relational database. This provides a mechanism that makes the proposed approach scalable to store a large number of web services and retrieve a service as per the requirement of matching to a given service. The proposed idea is implemented in jUDDI open source software. The software is extended with the proposed algorithm and tested to establish the practicality of the proposed technique. This work also proposes usages of established technology in developing web services and making this available on Internet to potential users.

REFERENCES

[1] Andreas Wombacher, Peter Fankhauser, Bendick Mahleko, Erich Neuhold : "Matchmaking for Business Processes based on Choreographies". In Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service, Taipei, Taiwan, pages 359-368,28-31 March (2004)

[2] Ruiqiang Guo, DongDong, Jiajin Le, Discovery for Web Services based on Relationship Model, In Proceedings of the 6th International Conference on Computer and Information Technology, Korea, pages 253-259, September (2006).

[3] Souripriya Das, Eugene Inseok Chong, George Eadon, Jagannathan Srinivasan, Supporting Ontology-based Semantic Matching in RDBMS, In Proceedings of the 30th VLDB Conference, Toronto,Canada, pages 1054-1065, August 29- September 3, 2004.

[4] Hongsuda Tangmunarunkit, Stefan Decker, Carl Kesselman, Ontologybased Resource Matching in the Grid - The Grid meets the Semantic Web, In Proceedings of International SemanticWeb Conference, Sanibel Island, Florida, USA, pages 706-721, 20-23 October 2003

[5] Kuo-Ming Chao, Muhammad Younas, Chi-Chun Lo, Tao-Hsin Tan,Fuzzy Matchmaking for Web Services, In Proceedings of the 19th International Conference on Advanced Information Networking and Applications, Tamkang University, Taipei, Taiwan, pages 721-726, 28-30March 2005.

[6] Maozhen Li, Bin Yu, Chang Hang, Yong-Hua Song Service Matchmaking with Rough Sets, In Proceedings of the 6th International Symposium on Cluster Computing and the Grid, Singapore, Vol-1, pages 123-130, 16-19 May 2006.

[7] Hrushikesha Mohanty, Deepak Chenthati, Supriya Vaddi, and R. K. Shyamsundar. 2008. HUMSAT for State Based Web Service Composition. In Proceedings of the 2008 International Conference on Information Technology (ICIT '08). IEEE Computer Society, Washington, DC, USA, Pages 273-278.

[8] UDDI Version 3.0.2 from http://uddi.org/pubs/uddi v3.htm

[9] WSDL Version 1.1 from http://www.w3.org/TR/wsdl.

[10] JUDDI from http://ws.apache.org/juddi/.

[11] Cheng Zeng, Weijie Ou, Yi Zheng, Dong Han. Efficient Web Service Composition and Intelligent Search Based on Relational Database, In International Conference on Information Science and Applications (ICISA) seoul,Pages 1-8, 21-23 April 2010.

[12] Hu jingjing, Zhao xing, Cao Yuanda, Zhou ruitao. A Service Composition Model with Characteristic of Transaction Based on Finite State Machine. International Conference on Computer and Electrical Engineering, 2008. ICCEE 2008. 20-22 Dec. 2008. Page(s): 450 - 454

[13] Scalable and efficient web services composition based on a relational database, Journal of Systems and Software(JSS- 8735), In Press, Corrected Proof, 7 June 2011.Daewook Lee, Joonho Kwon, Sangjun Lee, Seog Park, Bonghee Hong

[14] August-Wilhelm W. Scheer "Aris-Business Process Frameworks".Springer-Verlag New York, Inc., 2 edition, 1998

[15]] J. Ziemann and J. Mendling. "Epc-based modelling of bpel processes: a pragmatic transformation approach.", In Proceedings of the 7th International Conference Modern Information Technology in the Innovation Processes of the Industrial Enterprises (MITIP 2005), Genova, Italy, 2005.

[16] Hui Kang, Xiuli Yang, Sinmiao Yuan,"Modeling and Verification of Web Services Composition based on CPN,",Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on Network and Parallel Computing Workshops2007 , pages 613 – 617

[17] CPN tools. http://www.daimi.au.dk/CPNtools/