Discovering Substitutable and Composable Semantic Web Services for Web Service Composition

A.Bhuvaneswari
Assistant Professor, CSE Dept
Adhiparasakthi Engg. College
Melmaruvathur

G.R.Karpagam, PhD.
Professor, CSE Dept
PSG College of Technology
Coimbatore

ABSTRACT
Semantic Web Service Discovery is an indispensable part of Web Service Composition which is expected to (i) match the user request against multiple service advertisements and (ii) provide a set of Substitutable and Composable services by maintaining the relationship among services. This paper employs Indirect Backward Chaining Matchmaking approach to match against multiple advertisements and utilizes hash tables to maintain the relationship among services. The experimentation is done with the online product purchase scenario and the results are shown by considering (i) Input and Output only, (ii) Input, Output, Precondition and Effect and (iii) Data type of the input and output concepts. The quantitative evaluation is carried out based on the information retrieval measures recall, precision and F-measure.

General Terms
Web Service Composition, Web Service Discovery.

Keywords
Semantic Matchmaking, Indirect Backward Chaining, Substitutability, Composability, Recall, Precision

1. INTRODUCTION
The availability and accessibility of web services has facilitated users to invoke and utilize them to satisfy their requirements. The user request may be as simple as finding the product or as complex as purchasing the product. The complex user request will require more than one service which should be composed together to achieve the required task. This process of discovering and arranging a set of services in an order to satisfy the user request is called as Service Composition. Web Service Discovery (or Service Matchmaking) is an essential and integral part of service composition. Service discovery is a technique to locate the web services that match the requirements of the user request.

Conventional web service discovery is performed by manual search or through UDDI API. The standard UDDI registries permit keyword based search. The conventional discovery suffers from the following shortcomings and limitations: (i) Informal and incomplete description of service functionalities, (iii) syntactic relevance Vs intentional relevance, (iv) lack of constraint specification, (v) limited expressiveness of domain classification schemes and (vi) no support for indirect matching (finding composable services). A recent solution provided by the researchers [1,2] to overcome these limitations is enrich web services with semantics and discover them based on semantic information. The semantic matching algorithms are, in general, more complex and "intelligent" than the syntax-based.

Various semantic web service description standards are proposed namely Semantic Annotation for WSDL and XML Schema (SAWSDL), Web Service Modeling Ontology (WSMO) and Web Ontology Language for Web Services (OWL-S). OWL-S proposed by World Wide Web Consortium (W3C) has gained its popularity among researchers because of its maturity, availability of tools and expressiveness in representation language.

Semantic web service discovery allows matching a requested capability described as a set of provided inputs and required outputs with capabilities, also described as a set of required inputs and provided outputs. Inputs and outputs of capabilities are described with concepts in ontologies. The matchmaking algorithm defines four levels of matching between two ontology concepts, being respectively a provided and a required concept. These levels are:

• *exact*: if the concepts are the same or if the required concept is a direct subclass of the provided one,

• *plug in* if the provided concept subsumes the required one,

• *subsumes* if the required concept subsumes the provided one, and

• *fail* if there is no subsumption relation between the two concepts. Then, the matchmaking algorithm scores service descriptions according to the matching levels found between the concepts specified in the service request and those provided in the service advertisement.

[1] presents a set of algorithms that solve semantic matchmaking problem. These algorithms can be applied for matching the concepts between a request and a single advertisement. The relationship among services and matching against multiple service advertisements are not considered in those algorithms.

The discovery algorithm employed for web service composition should find the services that possess the (a) Substitutability and (b) Composability properties. The Substitutability property enforces that: *two services are said to be substitutable if they can perform the same required task to achieve a given goal, i.e. they are functionally equivalent by possessing relationship between their inputs and outputs.* The Composability property enforces that: *two services are composable if both the services are required to achieve a task, i.e. they are functionally dependent as the output of one service serves as the input of another service.* These two properties should be maintained in an efficient data structure which will provide faster access and support Service discovery. This paper considers hash table to efficiently maintain the relationship among the services.

The survey provided in [2] explains various algorithmic approaches to semantic web service discovery. The authors

had categorized the matchmaking approaches as shown in Figure.1.

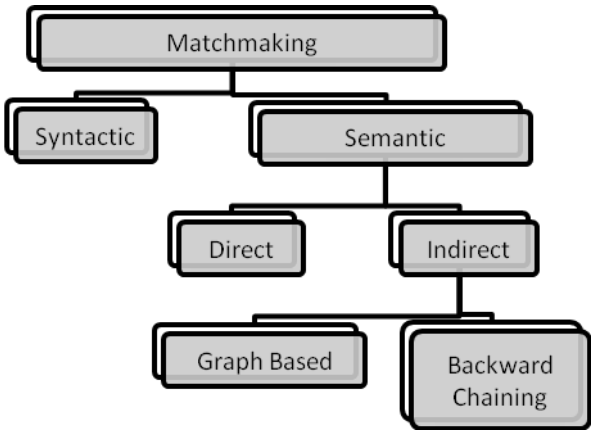

Fig. 1 Classification of Semantic Discovery

In general Matchmaking is classified into Syntactic and Semantic approaches. Semantic matchmaking is further classified into (i) Direct Matchmaking and (ii) Indirect Matchmaking.

(i) Direct Matchmaking: A service request is matched against single service advertisement

(ii) Indirect matchmaking: A service request is matched against set of services to identify the services that are more relevant to the service request.

Two types of Indirect Matchmaking approaches are suggested in [2]. One of the approaches is the Indirect Backward chaining algorithm which discovers the services starting from the query output and continues until the query inputs are reached. [2] had retrieved only one possible set of composable services as substitutable services are not considered and had specified that the algorithm can be further extended. This paper extends the algorithm in such a way that both substitutable and composable services are discovered.The paper also evidently shows the importance of precondition and effect parameters while discovering a set of composable services.

Section II provides the work related to semantic web service discovery, Section III presents a motivating scenario that requires a matchmaking algorithm, Section IV discusses the proposed algorithms, Section V shows the experimental results, and Section VI provides the contribution Section VII provides the discussion and conclusion of the paper.

2. RELATED WORK

Semantic web service discovery algorithm which identifies the similar services but without ranking facility was initially proposed by Paolucci[3]. [1] presented the concepts involved in semantic based matchmaking and provides a set of semantic matchmaking algorithms. [4] proposed an improved matching algorithm based on the grade match to compute the matching degree of two web services into a fine value denoted by a real number. However the efforts were made by these authors aid only in discovering services and do not consider the relationship among the discovered services which would help in discovering the composable services.

[2] had categorized the matchmaking algorithm as Direct, that matches a service request against a single advertisement and Indirect, that matches a service request against a set of advertisements. Indirect matchmaking algorithm leads to the discovery of set of composable services.

[5] had proposed a graph-based approach with a refined service-relationship graph generation algorithm based on Service-MessagePart Matrix. It uses semantic extended WSDL to reduce the search space but did not consider the equality semantics of Input and Output parameters to identify the services. [3] projected an approach that extracts the semantic similarities between I/O parameters of the services and represents them as a directed graph. It also recognizes and deals with cyclic dependencies among I/O parameters. [6] had suggested an approach for service composition which considered the dependency of web services using directed graph. [7] had proposed composition oriented service discovery with Service Aggregation Matchmaking algorithm which identifies the set of composable services based on I/O parameters only. [8] had proposed a framework for web service composition by checking the composability of services. The precondition and effect parameters were left out as the future work of [7] and [9] and they did not identify the substitutable services which is essential if the services are not available on the fly. [9] had suggested an approach that identifies the substitutable and composable set of services based on Input and Output parameters only. However this approach will not identify the services that are related through output and precondition parameters. [10] had proposed a method for service composition by considering hash table for maintaining the relationship of input and output parameters of services. This paper identifies the substitutable and composable services based on degree of match of IOPE parameters along with the data type match between the parameters using Indirect Backward Chaining approach.

3. MOTIVATING SCENARIO

An online product purchase is a complex scenario that requires a set of services to satisfy the user query. The scenario can be represented as a workflow of processes as shown in Figure 2.

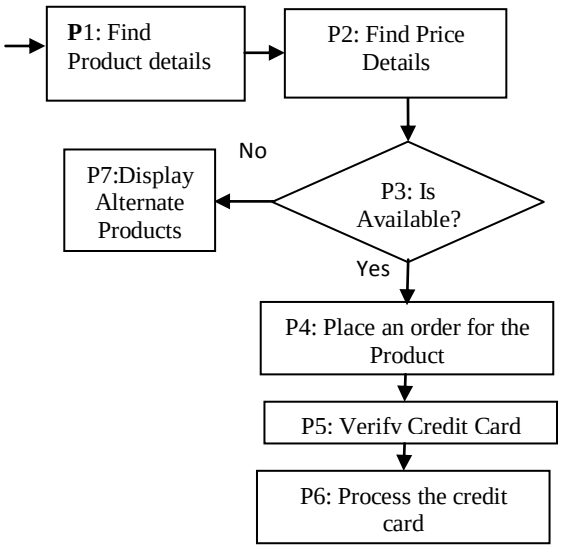

Fig.2 Workflow of the Online Product Purchase

The set of services given in Table A.1 in Appendix are considered and that are relevant to the workflow shown in Figure. 2 should be selected from them. The service parameters are extracted from the semantic description of services. The OWL-S request and semantic service

description of OrderProduct are shown in Figure A.1 and Figure A.2 of Appendix.

The user has knowledge only about the product name and credit card number the required input parameters will be Title and credit card number. Finally after purchase the user expects the bill-id and the amount debited in the credit card for verification as output. It is obvious that no single service in Table A.1 of Appendix can satisfy the user request. But a set of services when composed in an order will satisfy the query.

Clearly the scenario expects a set of composable services should be discovered by performing matchmaking against multiple advertisements which is different from the conventional web service discovery.

4. MATCHMAKING

This section provides the algorithm involved in finding the set of composable and substitutable services. Initially two hash tables HT1 and HT2 are created to maintain the composable and substitutable services, respectively. Both the tables maintain the relationship of a service with a single service or a set of services. HT1, the hash table that maintains the composable services contains an index for each web service whose input or precondition depends on the output or effect of other services. HT2, the hash table that maintains the substitutable services contains an index for each service whose input and output are same or a subset of other services. Along with the relationship of the service parameters their data type match is also considered while constructing both the tables. The construction of these hash tables is given as the algorithm in the Figure. 3.

The proposed matchmaking algorithm performs a backward chaining by initiating from the output parameters of the user query to find the set of composable services. First web services which contain q_{out} as output parameters are found and stored in W_i. Then the services which contain the input parameters of services in W_i as its output parameters using HT1. This is repeated until the query input parameters are found in the parameter list. The main algorithm is shown in Figure. 4. It proceeds as follows:

1. Initialize parameter list P_1 and P_2 with query output parameters. Initialize W_i to null which will store the set of composable services.
2. The web services whose output parameters semantically match with the query output parameters are stored in W_1. The input parameters of the services in W_1 are added to the parameter list.
3. Initialize i with 1.
4. Repeat step 4 through step 10 until the parameter list contains the query input parameters.
5. If W_i does not contain any services exit from the loop.
6. Increment the value of i by 1.
7. Initialize W_i with null and P_{i+1} with P_i.
8. For the services in W_{i-1} the composable services are found by looking up the hash table HT1 and stored in C; the substitutable services are found by looking up the hash table HT2 and stored in S.
9. If the output or effect parameters of the services in C are semantically matched with the parameter list, they are added to W_i
10. Else the services from the repository are searched such that the precondition of the service matches with the effect of the service in C. This includes the services which may be ignored as the query input is reached.
11. Finally the set of composable services are inserted into SEQ1 and substitutable services are inserted into SEQ2.

```
WS – Service registry
ConstructHT2( )
{
Foreach webservice ws in WS
        WS1 = WS – {ws}
 Foreach webservice s in WS1
 If((wsin = sin)  or (wsin ⊂ sin)) then
  If((DoM(wsin,sin) ≠ Fail) and  (DoM(wsout,sout) ≠
        Fail) or    (DoM(wspre,spre) ≠ Fail) or
        (DoM(wseff,seff) ≠ Fail)   then
    If((DoM(wsindt,sindt) ≠ Fail) and
        (DoM(wsoutdt,soutdt) ≠ Fail) then
            InsertHT2(ws,s)
    Else
        Print "Data Type Mismatch"
    End If
  End If
End If
End Foreach
}
ConstructHT1( )
{
Foreach webservice ws in WS
  WS1 = WS – {ws}
Foreach webservice s in WS1
   If((DoM(wsin,sout) ≠ Fail) and
        (DoM(wsout,sin) == Fail) or
        (DoM(wspre,sout) ≠ Fail) or
        (DoM(wspre,seff) ≠ Fail)  and
        (DoM(wsout,spre) == Fail) or
        (DoM(wseff,spre) == Fail) then
      If((DoM(wsindt,soutdt) ≠ Fail) then
        InsertHT1(ws,s)
        If(iskeyHT2(s) == true) then
          Sim = GetfromHT2(s);
          WS1 = WS1 – {Sim};
         End If
      Else
        Print "Data Type mismatch"
      End If
  End If
End Foreach
}
```

Fig. 3 Hash Table Construction

The semantic match between the parameters is found by using the Degree of Match algorithm in the Figure. 5. A scheme for degree of matching proposed by [1] is used in this paper as follows:

- **Exact:** If ws_{out} *is an equivalent concept to* q_{out}, then they are labeled as *Exact* match.
- **Plug in:** If q_{out} subsumes ws_{out} or q_{out} is a superclass of ws_{out} it is a Plugin match.
- **Subsumes:** If ws_{out} subsumes q_{out}, then it is a *Subsume* match.

- **Fail:** If no subsumption relation is found between q_{out} and ws_{out}, then it is declared as failure.

```
WS represents the service registry

P1 ← qout; P2 ← P1; W1 ← Φ
Foreach webservice ws in WS do
        If ((DoM (wsout, P1) ≠ Fail) or
        (DoM (wseff, P1) ≠ Fail)) then
                W1 ← W1 ∪ ws;
                P2 ← P2 ∪ wsin;
        Else
        Print "NoService Matches";
        End if;
End Foreach;

i←1;

while qin ∉ Pi+1
  if Wi == Φ then exit;
  i++;
  Wi ← Φ; Pi+1 ← Pi;
  Foreach webservice ws in Wi-1 do
        C ← C ∪ GetfromHashTable1(ws);
        S← S ∪ GetfromHashTable2(ws);
  End Foreach;

  Foreach webservice ws in C do
    If ((DoM (wsout, Pi) ≠ Fail) or (DoM(wseff, Pi) ≠
                                    Fail)) then
        Wi ← Wi ∪ ws;
        Pi+1 ← Pi+1 ∪ wsin;
      Else If ((DoM (wsout, Pi) == Fail) or
            (DoM(wseff, Pi) == Fail)) then
      Foreach webservice w in WS and w ∉ Wi
        If (DoM (wsout,wpre) ≠ Fail) or (DoM(wseff,
                    wpre) ≠ Fail)
            Wi ← Wi ∪ w;
      End if;
    End if;
  End Foreach;
End while;
l← i;
insert(W1, W2, … Wl, S ) to sequence SEQ;
```

Fig. 4 Matchmaking Algorithm

```
DoM(wsout, qout)
{
If (wsout ≡ qout) then
        Return Exact
Else if (qout superclassOf wsout) then
        Return Plugin
Else if (qout subsumes wsout) then
        Return Plugin
Else if((wsout subsumes qout) then
        Return Subsumes
Else
        Return fail
End if
```

Fig. 5 Degree of Match Algorithm

5. EXPERIMENTAL RESULTS

The experimentation was done with the semantic web services taken from the OWL-S TC2 data set released in 2010. The experiment was conducted based on (i) only output and input parameters (ii) input, output, precondition and effect parameters (iii) data type match between parameters. The hash tables that were constructed for the services are shown in Table A.2 in appendix.

The quantitative evaluation of service discovery is based on the popular information retrieval measures recall and precision [11]. A true positive result is the one in which a service that should be selected and that is available in the result. A false positive result is the one in which a service that should not be selected but that is available in the result. A false negative result is the one in which the service that should be selected but that is not available in the result.

F-measure [11] provides the harmonic mean of recall and precision measures. It is possible to get 100% recall by just returning all documents, and therefore always arithmetic mean will be 50% by the same process. This strongly suggests that the arithmetic mean is an unsuitable measure to use. In contrast, if only 1 document in 10,000 is relevant to the query, the harmonic mean score of this strategy is 0.02%. When the values of two numbers differ greatly, the harmonic mean is closer to their minimum than to their arithmetic mean.

$$Recall = \frac{number\ of\ correctly\ discovered\ services}{number\ of\ all\ correct\ services} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$Precision = \frac{number\ of\ correctly\ discovered\ services}{number\ of\ discovered\ services} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$F - measure = 2 * \frac{Recall * Precision}{Recall + Precision}$$

When input and output parameters were considered, the services to check the availability of the product and verify the validity of the credit card were not discovered. When discovery was improved by considering Input, Output, Precondition and Effect all the required services were discovered. As specified in [11], as recall increases precision will decrease as more false positives will be included. This typical situation occurred in this quantitative evaluation also.

Table 1 Recall and Precision

Parameters	Recall	Precision	F-measure
I/O	0.8	0.66	0.72
I/O/P/E	0.93	0.63	0.75
with data type	0.93	0.77	0.76

From Table 1 it is evident that as recall increases, precision decreases. When data type is included the false positives decreases which increases the precision while recall is consistent and hence a straight line is obtained. In all the cases F-measure increases. Hence the obtained recall-precision graph is as shown in Figure 6 and F-measure is as shown in Figure 7.

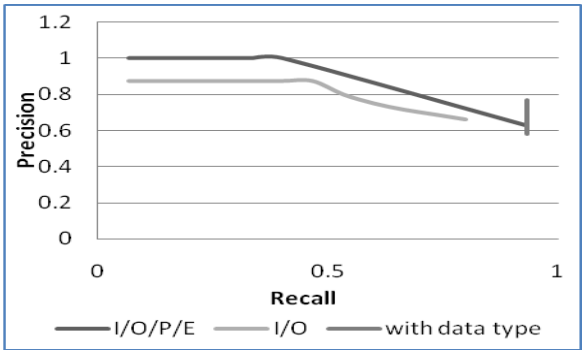

Fig. 6 Recall – Precision Graph

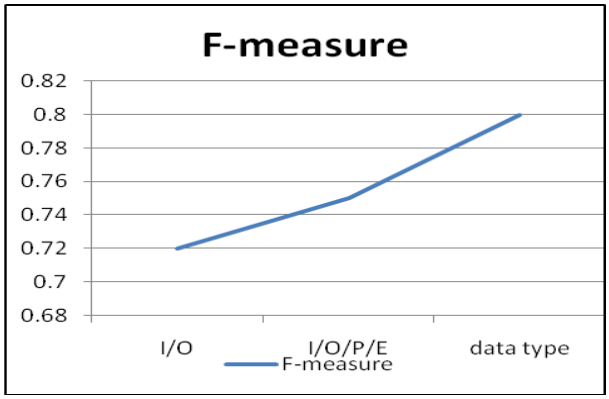

Fig. 7 F-measure

6. CONTRIBUTIONS

Following are the major contributions of this paper:

- It is identified that the web services required for web service composition should possess two properties namely Substitutability and Composability which are based on the relationship among the services.
- The relationship among the web services are identified based on their semantic profile which consists of Input, Output, Precondition and Effect parameters.
- Hash table is suggested as an efficient data structure to maintain the relationship among services.
- Indirect Backward Chaining is employed to find the composable and substitutable services based on the relationship maintained in the hash table.

7. DISUSSION AND CONCLUSION

1. Through research survey it has been identified Web Service Composition considers only composable services. However composable and substitutable services have to be considered for composition. The matchmaking (indirect backward chaining) algorithm focuses on identifying both composable and substitutable services which will be helpful while composing the services and when an alternate service is required for a selected service.
2. The usage of IOPE will increase the recall and provides the maximum possible services as output.
3. Type checking is normally considered during the verification or planning which might populate the hash table with irrelevant services. As part of the hash table construction type checking is also performed that will eliminate irrelevant services.
4. The impact of type checking has reflected in precision and recall metrics and hence the F-measure.

8. REFERENCES

[1] Umesh Bellur, Harin Vadodaria, Amit Gupta. 2008. Semantic Matchmaking Algorithms, In Witold Bednorz(Ed.), Greedy Algorithms, pp. 586, I-Tech, Vienna, Austria, InTech Publisher.

[2] Vassileios Tsetsos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades, 2007. Semantic Web Service Discovery: Methods, Algorithms and Tools, In Dr. Jorge Cardoso(Ed.), Semantic Web: Theory, Tools and Applications, Ch. XI pp. 240-280, IGI Global, IDEA Group Publishing.

[3] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. 2002. Semantic Matching of Web Services Capabilities. In Proceedings of the First International Semantic Web Conference on The Semantic Web (ISWC '02), Springer-Verlag, London, UK, UK, 333-347.

[4] Hui Peng, Zhongzhi Shi, Liang Chang, and Wenjia Niu. 2008. Improving Grade Match to Value Match for Semantic Web Service Discovery. In Proceedings of the 2008 Fourth International Conference on Natural Computation - Volume 05, Vol. 5. IEEE Computer Society, Washington, DC, USA, 232-236.

[5] Shudong, Zhang. Naiming, Yao. Ye, Qin. 2010. "A Refined Service Relationships Graph Generation Algorithm for Web Service Composition", In Proceeding of Asia Conference on Informatics in Control, Automation, and Robotics, Vol. 3, pp. 196-199.

[6] Abrehet, M., Omer, Alexander Schill, 2009. "Dependency Based Automatic Service Composition using Directed Graph", Fifth International Conference on Next Generation Web Services Practices, nwesp, Prague, Czech republic, Sep 9 – 11, pp.76-81.

[7] Antonio Brogi, Sara Corfini, Razvan Popescu, Composition Oriented Service Discovery, 2005, In Proceedings of the 4th international conference on Software Composition (SC'05), Springer-Verlag, Berlin, Heidelberg, pp. 15-30.

[8] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. 2003. Composing Web services on the Semantic Web. The VLDB Journal, Vol. 12, No. 4 (November 2003), 333-351.

[9] Michael D. Ernst, Raimondas Lencevicius. 2006. "Detection of Web Service Substitutability and Composability", Int. workshop on web services modeling and Testing, pp.123-135.

[10] Joonho, Kwon. Daewook, Lee. 2012. "Non-redundant Web Services Composition based on a Two-phase algorithm", International Journal of Data and Knowledge Engineering, Vol.71, No.1, pp. 69-91.

[11] precision is the fraction of retrieved instances that are relevant, while recall is the fraction of relevant instances that are retrieved available at http://en.wikipedia.org/wiki/Precision_and_recall.

9. APPENDIX

Table A.1 Sample Set of Services

Index	Service Name	Input	Output	Precondition	Effect
s1	FindProductPrice	ProductName, ProductType	Pcode(int) Price(int)	ProductFound	
s2	FindProductType	ProductName	ProductType Expirydate		ProductTypeFound ProductTypeNotFound
s3	AddtoCart	ProductName Price	OrderId OrderAmount	ProductAvailable	ProductAdded
s4	ShowAvail	ProductName ProductType	ProductAvailable ProductNotAvailable		
s5	PlaceOrder	ProductName ProductType	OrderId, OrderAmount	ProductAvailable	ProductOrdered
s6	OrderProduct	Pcode(string) ProductType(string) Price(int)	OrderId OrderAmount	ProductAvailable	ProductOrdered
s7	PcodePrice	Pcode	Price		
s8	ChargeCC	CCNum Orderid Orderamt	debitedAmount Bill-id	CCApproved	CCCharged ProductPurchased
s9	CheckCC	CCNum CCType PIN	ValidCC	ProductAvailable ProductOrdered	CCApproved CCNotApproved
s10	DisplayAlt	ProductName TypeX	ProductName TypeY	ProductNotAvailable PtypeNotFound	
s11	ProductPrice	ProductName Price(float)	ProductType		
s12	ProductAvail	ProductName ProductType1	ProductAvailable ProductType2		
s13	CCCheck	CCNum Pin ExpiryDate			CCApproved CCNotApproved
s14	DisplayAltProduct	Product1	Product2 Manufacturer	ProductNotAvailable	
s15	ProductCodePrice	ProductName ProductType	Pcode(float) Price(float)	PTypeFound	
s16	PlaceOrder_Product	ProductType(string) Price(float)	OrderId OrderAmount	ProductAvailable	ProductOrdered
s17	OrderforProduct	ISBN(string) ProductType(string) Price(float)	OrderId OrderAmount	ProductAvailable	ProductOrdered
s18	ChargeCC1	CCNum Orderid Orderamt	debitedAmount Bill-id	CCApproved	CCCharged ProductPurchased
s19	ProductPrice	ProductName Price(int)	ProductType		
s20	CCCheck	CCNum ExpiryDate			CCapproved CCNotapproved
s21	SearchProductType	ProductName	ProductType		
s22	ShowAltProductType	ProductName ProductType1	ProductName ProductType2	ProductNotAvailable ProductTypeNotFound	
s23	ShowAltProduct	Product1	Product2 Edition	ProductNotAvailable	
s24	FindAltProduct	Product1	Product2 Publisher	ProductNotAvailable	
s25	GetAltProduct	Product1	Product2	ProductNotAvailable	
s26	GetPriceCode	ProductName ProductType	PCode(float) Price(float)	ProductTypeFound	

Table A.2. Hash Table Entries based on (a) I/O (b) I/O/P/E (c) Data Type

(a)Input and Output			(b) Input, Output, Precondition, Effect		(c) Based on data type	
key	**Comp**	**Subs**	**Comp**	**Subs**	**Comp**	**Subs**
s1	s2, s10,s21,s24	s15,s26	s2, s10,s21,s24	s15,s26	s2,s10,s21,s24, s21,s24	s15,s26
s2	s14,s22,s23,s25	s21	s14,s22,s23,s25	s21	s14,s22,s23,s25	s21
s3	s1, s7, s10,s24,s26		s1, s7, s10,s26,s24,s26		s1,s7,s10,s26,s24,s26	
s4	s2, s10, s11, s21,s24		s2, s10, s11, s21,s24		s2,s10,s11, s21,s24	
s5	s1, s2, s7, s10, s11,s21,s24,s26	s16	s1, s2, s7, s10, s11,s4,s12, s21,s24,s26	s16	s1, s2, s7, s10, s11,s4, s21,s24,s26	
s6	s1, s2, s7, s10, s11, s21,s24,s26	s5, s16, s17	s1, s2, s7, s10, s11,s4,s12, s21,s24,s26	s5, s16, s17	s1, s2, s7, s10, s11,s4, s21,s24,s26	s5
s7	s1,s26		s1,s26		s1,s26	
s8	s3, s5, s6	s18	s3, s5, s6, s9, s13	s18	s3, s5, s6, s9, s13	
s9		s20	s4, s5, s6	s20	s4, s5, s6	
s10	s2, s19, s21	s12, s14,s22,s23,s25,s24	s2, s19, s21	s12, s14,s22,s23, s25,s24	s2, s21	s12,s14,s22,s23,s25,s24
s11	s7	s2, s19, s21	s7	s2, s19	s7	s2
s12	s2, s19, s21		s2, s19, s21		s2, s21	
s13		s9, s20	s4, s5, s6	s9, s20	s4, s5, s6	s9
s14		s22,s23,s25	s4, s2, s21	s22,s23,s25	s4, s2, s21	s22,s23,s25
s15	s2, s10, s21,s24	s1,s26	s2, s10,s24			
s16	s1, s2, s7, s10, s11, s21,s24,s26	s5	s1, s2, s7, s10, s11, s4, s12, s21,s26	s5	s15	
s17	s1, s2, s7, s10, s11, s21,s24,s26	s5, s6, s16	s1, s2, s7, s10, s11, s4, s12, s21,s24,s26	s5, s6, s16	s16	s16
s18	s3, s5, s6	s8	s3, s5, s6	s8		
s19	s7	s2, s11, s21	s7	s2, s11		
s20			s4,s5,s6		s18	
s21	s14,s22,s23,s25	s2	s14,s22,s23,s25	s2	s14,s22,s23,s25	s2
s22		s14	s4, s2, s21	s14	s4, s2, s21	s14
s23		s14	s4, s2, s21	s14	s4, s2, s21	s14
s24	s2, s19, s21	s10,s12, s14,s22,s23,s25	s2, s10,s19, s21	s10,s12, s14,s22,s23, s25	s2, s21	s10,s12,s14,s22,s23,s25
s25		s14	s4, s2, s21	s14	s4, s2, s21	s14
s26	s2, s10,s21,s24	s1, s15	s2, s10,s21,s24	s1,s15	s2,s10,s21,s24, s21,s24	s1,s15

```
<profile:Profile rdf:ID="RequestOrderProduct">
<profile:hasInput>
<profile:hasInput  rdf:resource="#_PRODUCTNAME"/>
<profile:hasInput rdf:resource="#_CCNUM"/>
<profile:hasOutput  rdf:resource="#_DEBITEDAMOUNT"/>
<profile:hasOutput rdf:resource="#_BILL-ID"/>
</profile:Profile>
<process:Input rdf:ID="_PRODUCTNAME">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
</process:parameterType>
<process:Input rdf:ID="_CCNUM">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
</process:parameterType>
</process:Input>
<process:Output>
<process:Output  rdf:ID="_DEBITEDAMOUNT">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
</process:parameterType>
<process:Output  rdf:ID="_BILL-ID">
<process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
</process:parameterType>
</process:Output>
```

Fig A.1 OWL-S Request

```
<profile:has_process rdf:resource="ORDERPRODUCT__PROCESS"/></profile:Profile>
<!--<process:ProcessModel rdf:ID="ORDERPRODUCT__PROCESS_MODEL">
<service:describes rdf:resource="#ORDERPRODUCT__SERVICE"/>
<process:hasProcess rdf:resource="#ORDERPRODUCT__PROCESS"/>
</process:ProcessModel>-->
<process:AtomicProcess rdf:ID="ORDERPRODUCT__PROCESS">
<service:describes rdf:resource="#ORDERPRODUCT__SERVICE"/>
<process:hasInput  rdf:resource="#_PCODE"/>
<process:hasInput  rdf:resource="#_PRODUCTTYPE"/>
<process:hasInput  rdf:resource="#_PRICE"/>
<process:hasOutput rdf:resource="#_ORDERID"/>
<process:hasOutput rdf:resource="#_ORDERAMOUNT"/>
<process:hasPrecondition>
<expr:SWRL-Condition rdf:ID="ProductAvailability">
<expr:expressionLanguage rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#SWRL"/>
<expr:expressionBody rdf:parseType="Literal">
<swrl:AtomList> <rdf:first> <swrl:ClassAtom>
<swrl:classPredicate rdf:resource="http://127.0.0.1/ontology/ontosem.owl#be-available"/>
<swrl:argument1 rdf:resource="#_PRODUCT"/>
</swrl:ClassAtom> </rdf:first>
<rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/> </swrl:AtomList>
</expr:expressionBody>
</expr:SWRL-Condition>
</process:hasPrecondition>
<process:hasResult>
<process:Result rdf:ID="ProductOrdered">
<process:hasEffect>
<expr:SWRL-Expression>
<expr:expressionLanguage rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#SWRL"/>
<expr:expressionBody rdf:parseType="Literal">
<swrl:AtomList><rdf:first><swrl:ClassAtom>
<swrl:classPredicate rdf:resource="http://127.0.0.1/ontology/ShoppingCart.owl#ShoppingCartRequestItems"/>
<swrl:argument1 rdf:resource="#_PRODUCT"/>
</swrl:ClassAtom></rdf:first>
<rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
</swrl:AtomList></expr:expressionBody></expr:SWRL-Expression>
</process:hasEffect></process:Result></process:hasResult></process:AtomicProcess>
```

Fig A.2 OWL-S for OrderProduct Service