SPortS: Semantic + Portal + Service

Chenxi Lin, Lei Zhang, Jian Zhou, Yin Yang, and Yong Yu

APEX Data and Knowledge Management Lab
Department of Computer Science and Engineering
Shanghai JiaoTong University, 200030, China
{linchenxi,tozhanglei,priest_zhou,yini,yyu}@sjtu.edu.cn

Abstract. Ontology-based web portal generation and management is an active field of research and development. Recently, many systems have been developed. However, many of them lack the integration of web services which may provide more dynamic information and richer functionalities. In this paper, we describe SPortS, an OWL-DL based portal generation system that integrates semantic web services into generated portals at the presentation level, the data level and the function level. We present the portal generation process, the query decomposition and the service recommendation algorithms through which semantic web services are integrated tightly into the generated portal.

1 Introduction

Because of its clear advantages over conventional methods[1], ontology-based web portal generation and management has become an active field of research and development. SEAL[2] provides a general framework for developing semantic portals. Numerous tools such as KAON portal[3], ODESeW[4], OntoWeaver[5], and OntoWebber[6] have been developed to help design, generate and maintain ontology-based web portals. However, these systems lack the integration of web services which may provide more dynamic information and richer functionalities. The integration of web services can bring the following benefits:

– Integrating information providing services may provide broader and more up-to-date information that serves as an important extension to the domain knowledge.
– Integrating world changing services may transform the current portal that is mostly limited to organizing and presenting domain knowledge to one that can also help user act based on contextual knowledge.
– Exposing the portal itself as web services makes it possible for other programs and agents on the Semantic Web to visit the portal.

Motivated by the above considerations, we have developed SPortS – a prototype ontology-driven system that automatically generates service-integrated web portals based on declarative specifications. The major features of the system are the following:

– Based on the common OWL-DL[7] formalism, semantic web services[8] and domain knowledge are processed and presented uniformly in our system. Hence, web services are treated as first-class citizens.
– By decomposing complex queries until they can be answered using available web services and domain resources, we unified the two information sources. The result is that, besides domain resources, the information providing services can also transparently feed information to the portal. The information serves as an important extension to the domain knowledge.
– Through capturing recently visited concepts to form a semantic context, SPortS can dynamically recommend world changing services that the users may be interested in under the current context. In this way, not only the portal's functionalities are greatly enriched by these services but also the portal's contextual knowledge is exploited to help users act in the portal.
– SPortS uses OWL-DL as the underlying knowledge representation formalism. Compared with RDFS, OWL-DL is more expressive in describing the constraints on and relationships among ontologies. Meanwhile, unlike OWL-FULL, OWL-DL is decidable and computationally complete with support from many existing DL reasoners (e.g. FaCT[9] and Racer[10]).

The rest of the paper is organized as follows. Section 2 gives an overview of the SPortS system. Section 3 discusses the web portal generation and the web service presentation process. Section 4 shows how to decompose complex queries to integrate information providing services. In section 5, we describe the recommendation process of world changing services and finally we conclude the paper in section 6.

2 Overview of the SPortS System

In the design of the SPortS architecture(Fig.1), the main objective is to integrate web services at the presentation level, the data level, and the function level. Among them, the data level integration plays a central role of driving the integration on the other two levels. In SPortS, data is stored in four knowledge bases which are all based on the OWL-DL formalism.

Domain Knowledge Base (Domain KB) stores the domain classes, properties, and individuals. It serves as the primary knowledge source for the portal.

Service Knowledge Base (Service KB) stores OWL-S[1] descriptions of all available semantic web services. Through this KB, SPortS can retrieve services' specifications to render or invoke them.

Site Knowledge Base (Site KB) contains the site ontology used to model the portal at a conceptual level and stores the declarative specification of the portal as the ontology's instance.

[1] http://www.daml.org/services/owl-s/1.0/

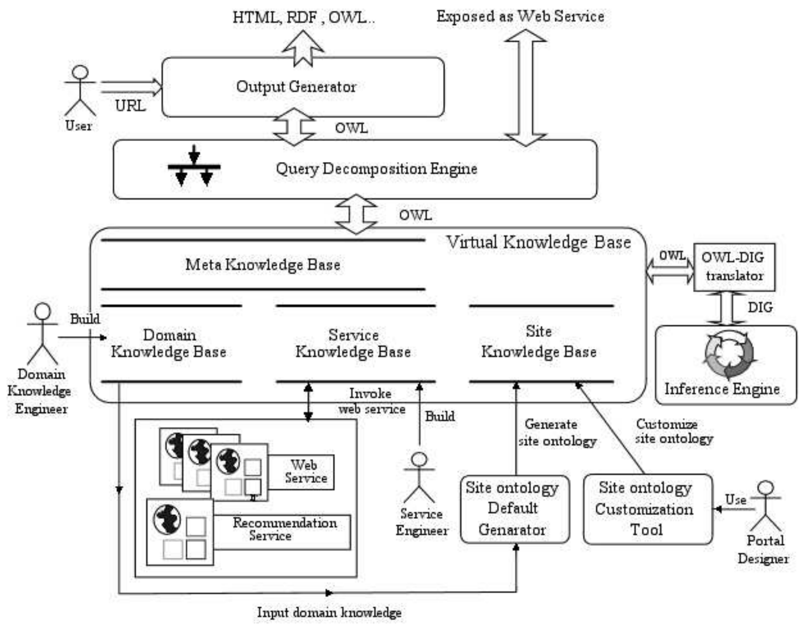


Fig. 1. SPortS architecture

Meta Knowledge Base (Meta KB) is designed to enable us to directly manipulate classes and properties in other KBs as individuals. This capability is necessary in a semantic portal system because of the need to show and process both classes and individuals at the same time.

Based on the common OWL-DL formalism, the four knowledge bases are grouped into a virtual knowledge base(Virtual KB) with a unified ontology. Queries issued to the Virtual KB can be expressed in the unified ontology without caring how knowledge from different KBs and services is composed to answer them. This task is actually accomplished by the Query Decomposition Engine. In this way, knowledge from different KBs and even services are integrated and can be accessed transparently.

The portal creation process consists of two phases, the design phase and the generation phase. The major task of the design phase is to prepare the four knowledge bases. The domain knowledge engineer builds the Domain KB and then the default site KB is automatically generated from it. After that, the portal designer designs the content, layout and navigational structure of the portal by customizing the default site KB using the customization tool. At the same time, the service integration engineer is responsible for populating the Service KB with available semantic web services.

During the generation phase, the Output Generator runs as a daemon. Upon receiving a URL, it dynamically constructs a web page. This construction process achieves presentation level integration, which is described in Section 3. The data presented in the web page is retrieved via the Query Decomposition Engine. The query decomposition process is discussed in Section 4. Decomposed queries that can be answered by a single knowledge bases are ultimately processed with the help of the inference engine.

When the user browses web pages in the portal, the Recommendation Service intelligently analyzes context knowledge and recommends world changing services that the user may be interested in. The recommended service can be presented in the portal for invocation, which achieves function level integration. Section 3.2 and Section 5 give more details about this process.

3 Portal Generation and Service Presentation

3.1 Site Ontology and Portal Generation

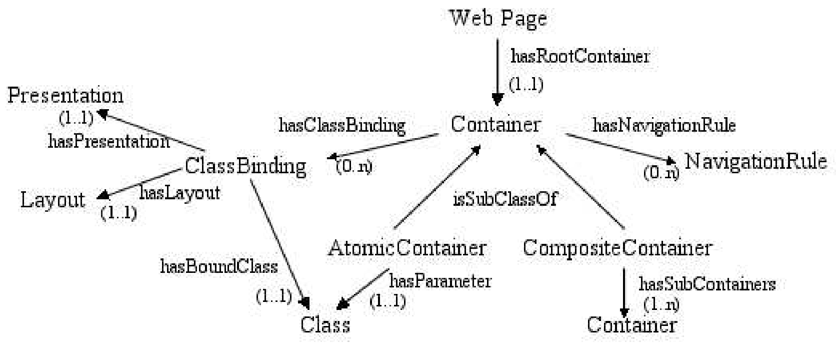


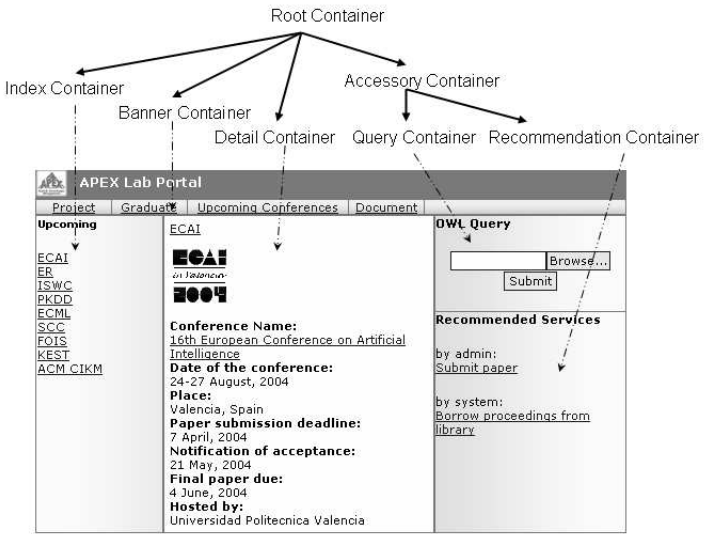
Fig. 2. Overview of the Site Ontology

The site ontology is designed to model web portals at a conceptual level. As shown in Fig.2, the site ontology includes the following core classes: *container*, *class binding*, *layout*, *presentation* and *navigation rule*.

In SPortS, the content of a web page is grouped into a tree of containers. Atomic containers are mainly used to present the concrete information, while composite containers group and layout the contents. In this paper, we use our lab's internal portal, Apex-Lab-Portal, as an example. Fig.3 shows a web page of the portal and its container tree structure.

The content of an atomic container is specified as the result of a query over the virtual KB. Currently, queries can only be expressed in the form of OWL-DL class descriptions. Hence, the content of an atomic container is the set of all individuals of the query class. This limitation is further discussed in section 4.1. For example, in Fig.3, the BannerContainer represents its content using the following class description:

$$\{\text{Project, Graduate, UpcomingConference, Document}\} \tag{1}$$



Upcoming

ECAI
ER
ISWC
PKDD
ECML
SCC
FOIS
KEST
ACM CIKM

ECAI

Conference Name:
16th European Conference on Artificial Intelligence
Date of the conference:
24-27 August, 2004
Place:
Valencia, Spain
Paper submission deadline:
7 April, 2004
Notification of acceptance:
21 May, 2004
Final paper due:
4 June, 2004
Hosted by:
Universidad Politecnica Valencia

OWL Query

Recommended Services

by admin:
Submit paper

by system:
Borrow proceedings from library

Fig. 3. A web page of the Apex-Lab-Portal example

It is an OWL-DL enumerated class in the Meta KB. Note that Project, Graduate, etc. are individuals in the Meta KB, while in the Domain KB they are classes. As another example, the index container use the following class description:

$$\exists\,\text{focusedBy}.\{\text{apexLab}\} \wedge \text{Conference} \wedge \exists\,\text{hasDeadline}.\,(\exists\,\text{laterThan}.\{\text{today}\}) \quad (2)$$

This is actually a class in the Virtual KB because the apexLab individual only exists in the Domain KB, while the laterThan property only exists in the capability descriptions in the Service KB. The two example classes as queries will be answered by the Query Decomposition Engine and we can see that the atomic container can easily compose its content with information from the Virtual KB regardless how and from where the information is obtained, even from the invocation of web services. Section 4 has more details for this example.

To further customize what properties of and how the individuals in an atomic container are displayed, class bindings are attached to containers. Every class binding has a bound class, a layout, and a presentation. When an individual i in an atomic container O is to be presented, i must use a class binding B whose bound class C satisfies $i \in C$ and B must be attached to O or O's ancestors. For example, in Fig.3, the class binding of the IndexContainer determines that only the names of the conferences are displayed. In contrast, in the DetailContainer, the names, date, place, etc. of the conference are displayed using the following DetailConfView class binding:

Class Binding:	DetailConfView
Bound Class:	Conference
Properties:	Logo, Name, Date, Place, Submission Date, Notification Date, Final Date, Host
Layout:	Vertical
Presentation:	Default

In the above example, the Vertical layout arranges the properties to display, and the values of these properties are presented recursively until a presentation can fully handle them. A presentation is actually a piece of program that completely handles the rendering of the matched individual. In this way, using a service presentation, SPortS can also render semantic web services because they are described in OWL-DL in the Virtual KB. We further discuss this issue in Section 3.2.

To model the navigational structure of the portal, the class navigation rule is defined in the site ontology. Recall that a web page is modelled as a tree of containers. The navigation from one page to another is thus modelled as a tree transformation that replaces a subtree with another one. The transformation is triggered when an item on a web page is clicked. A navigation rule bound to an appropriate container is then activated to actually transform the web page.

A navigation rule consists of a content condition, a structural condition, the subtree to replace, and a template for generating a new subtree. In SPortS, all clickable items are individuals in the Virtual KB. The content condition is met if the clicked individual is an instance of a Virtual KB class defined in the content condition. The structural condition is met if the clicked individual is presented in a container defined in the structural condition. For instance, the following navigation rule is bound to the RootContainer:

Content Condition:	Conference
Structural Condition:	IndexContainer
Subtree to Replace:	DetailContainer
New Subtree:	DetailContainer({clickedIndividual})

When the user clicks any conference in the IndexContainer, e.g. ECAI, the conference is then presented in the DetailContainer, as shown in Fig.3. Since the content condition is actually a class defined in the Virtual KB, web services can also be used to activate navigation rules just like other kinds of individuals. An example is given in Section 3.2.

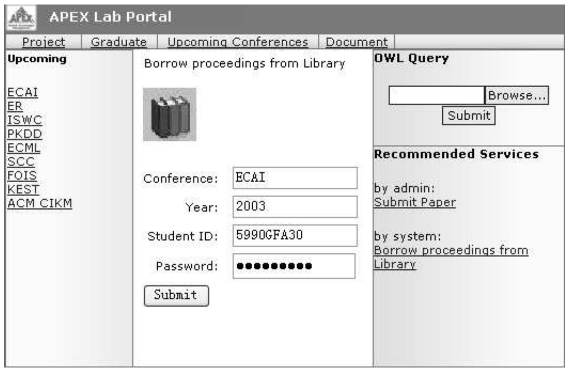
3.2 Presentation of Semantic Web Services

Based on the common OWL-DL formalism, semantic web services and domain knowledge are processed and presented uniformly in our system. In this subsection, we show in detail how web services are presented in exactly the same manner using the above mentioned containers, class bindings, layouts, presentations, and navigation rules.

In Fig.3, there is a RecommendationContainer that lists dynamically recommended world changing web services. The OWL-DL class description of its content is RecommendedService, whose individuals are retrieved by the Query Decomposition Engine (ref. Section 4) via the Recommendation Service (ref. Section 5). When the "Borrow proceedings from library" service link is clicked, the following navigation rule bound to the RootContainer is activated:

Content Condition:	RecommendedService
Structural Condition:	AccessoryContainer
Subtree to Replace:	DetailContainer
New Subtree:	DetailContainer({clickedIndividual})

The DetailContainer is then replaced to show the details of the clicked service. Fig.4 illustrates the result page. The DetailContainer is attached with a class



Borrow proceedings from Library

Conference:	ECAI
Year:	2003
Student ID:	5990GFA30
Password:	●●●●●●●●●

Fig. 4. Detailed View of the Borrow-Proceeding Service

binding for semantic web services shown below:

Class Binding:	DetailServiceInputView
Bound Class:	Service
Properties:	NULL
Layout:	NULL
Presentation:	ServiceInputPresentaion

Since the Borrow-Proceeding service is an instance of the Service class, it is covered by this class binding and is shown using the ServiceInputPresentation. Because the service is fully handled by the presentation, there is no need to specify the Properties and Layout for this class binding.

Portal users then can input service parameters by filling the form in the DetailContainer. When the form is submitted, SPortS automatically invokes the Borrow-Proceeding service according to its OWL-S's WSDL grounding. Finally, the effect of this invocation is again shown in the DetailContainer as an individual of the "Effect" class.

4 Query Decomposition on Virtual KB

The primary motivation for designing the Query Decomposition Engine is to provide a unified query mechanism for the Virtual KB in order to achieve data level integration. The engine enables users to issue queries in the unified ontology without worrying about which KB or information providing service actually answers the query. Furthermore, the engine not only decomposes the queries but also combines query results from different KBs or information providing services to answer the original complex query. Through this method, potentially broader and more up-to-date information from services can be integrated into the portal and presented to the users.

4.1 Problem Definition

Queries sent to the Query Decomposition Engine are expressed as OWL-DL class descriptions using the unified ontology in the Virtual KB. We admit that this is actually a simplification of the problem. More complex query expressions are actually possible. [11] proposed a formal description logic query language and its algorithm for the Semantic Web. Although our compromise constrains the capability to express queries to some extent, we find in practice that most of the queries in semantic portals can still be expressed. Besides, most current DL reasoning engines provides support for answering this type of queries. Precisely, the query is an OWL-DL class expression C, and the answer set of the query is $\{i \mid \Omega \models C(i)\}$ where Ω is the entire Virtual KB. In DL terms, the problem is thus an ABox *retrieval problem* .

We roughly divide the information sources in the Virtual KB into two kinds. One is those information sources whose query capability can be declaratively specified. For example, most information providing services are of this kind. The query capability can be described by a set of input class descriptions $\mathcal{I}$ and an output class description O. Each input class description $I \in \mathcal{I}$ describes the parameter type/class of an input i. Given all inputs of a service, O describes the output of the service as a class of individuals. Both $\mathcal{I}$ and O can be expressed in OWL-S profiles. A concrete example of $\mathcal{I}$ and O is given in the next subsection.

The other kind of sources, however, have very powerful query capabilities that can not be expressed using a single class expression. The Domain KB is such an example. Backed by a DL inference engine, it can retrieve instances for any class in its ontology. Nonetheless, given a query class, we require this kind of sources be able to tell whether the query can be answered independently by themselves.

Given a query class expression and the information sources in the Virtual KB, the SPortS Query Decomposition Engine should make the most of the available sources to answer the query. For example, it is possible that a query can not be answered alone either by the Domain KB or the information providing services. In this case, the Query Decomposition Engine should do its best to combine them for answering the query. Recall that in our Apex-Lab-Portal example the query class of the IndexContainer (as formulated in Section 3.1 formula 2) can only be answered by combining Domain KB and information providing services.

4.2 Query Decomposition Algorithm

Queries and capabilities of information providing services are expressed as OWL-DL class descriptions. Therefore, in theory, the query decomposition algorithm needs to find a semantically-equivalent logic combination of several OWL-DL class descriptions for a given OWL-DL query class. To perfectly solve the problem of query decomposition is very difficult if not impossible. Currently in SPortS, we attack the problem via a syntactical approach.

Any OWL-DL class description can be parsed into a syntax tree. According to the OWL-DL abstract syntax definition in [7], the tree has 6 basic elements shown in Fig. 5. In our running Apex-Lab-Portal example (Fig.3), in order to

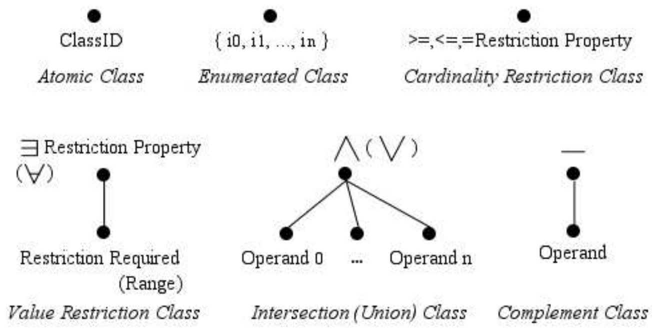


Fig. 5. OWL-DL Syntax Tree Elements

show a list of upcoming conferences focused by our lab in the IndexContainer, the query class as formulated in Section 3.1 formula 2 is

$$\exists\,\text{focusedBy}.\{\text{apexLab}\} \wedge \text{Conference} \wedge \exists\,\text{hasDeadline}.\left(\exists\,\text{laterThan}.\{\text{today}\}\right) \quad (3)$$

Its syntax tree is shown in Fig.6. The conferences must satisfy two requirements. Firstly, they must be focused by the Apex lab. This can be expressed as

$$\exists\,\text{focusedBy}.\{\text{apexLab}\} \wedge \text{Conference} \quad (4)$$

which can be answered by the Domain KB. Secondly, the paper submission deadline must be later than "today". In fact, the paper submission deadline of a conference is not fixed because it may change from year to year and even in one year it may still be postponed. In our Apex-Lab-Portal example, a semantic web service is built that periodically crawls the conferences' web sites to collect the paper submission deadlines of conferences. It then has the capability to return a list of conferences whose paper submission deadlines are later than an input date. The input class description of the service's capability is $\mathcal{I} = \{\text{Date}\}$. The output class description O is

$$\text{Conference} \wedge \exists\,\text{hasDeadline}.\,(\exists\,\text{laterThan}.\{input\}) \qquad (5)$$

where $input \in$ Date. The syntax tree of O is shown in Fig.7.

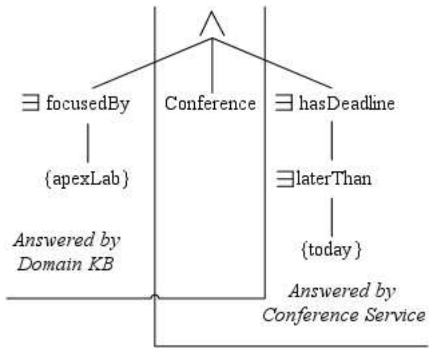


Fig. 6. Syntax Tree of Query

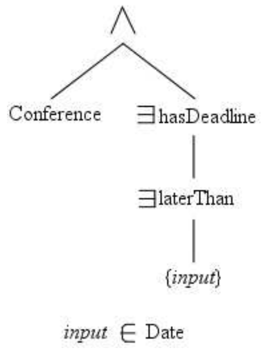


Fig. 7. Syntax Tree of Service Output

It is clear that the class expression (3) can not be answered alone either by the class expression (4) or (5). Yet the intersection of the latter two is exactly the answer. This is also reflected on the query's syntax tree (Fig.6). The left two subtrees can be answered by the Domain KB and the right two subtrees can be answered by the service. Thus all the subtrees are covered by at least one information source and the entire query can be answered. This idea can be seen as a horizontal decomposition. Note that in Fig.5, only intersection and union syntax tree has more than one sub-trees, therefore this idea is used to decompose intersection and union queries. On the other hand, we also need a vertical decomposition method, i.e. query result of a sub-tree may be used as input to feed another service.

The query decomposition algorithm works recursively on the query syntax tree by repetitively doing horizontal and vertical decomposition until the entire query can be answered. If there are more than one way to decompose the query, currently we assume that any of them is acceptable. Before presenting the details of the algorithm, we first give definitions for the "match" and "semi-match".

Match: $match(Q, S)$ is an ordered binary relation. A query (sub)tree Q matches a syntax (sub)tree S of a service's output class description if one of the following conditions is met.

– S is an atomic class with a ClassID and Q is an equivalent class of S.

- S is $\{s_0, s_1, \ldots, s_{n-1}\}$, Q is $\{q_0, q_1, \ldots, q_{n-1}\}$, there is a one-to-one mapping between s_i and q_j that satisfies:
  - if s_i is an input, then q_j is an instance of the input class description of s_i
  - if s_i is not an input, then s_i is equivalent to q_j.
- both S and Q are cardinality restriction classes and they have exactly the same class description.
- S is $\forall R_s.C_s$, Q is $\forall R_q.C_q$, R_s is an equivalent property of R_q, and $match(C_q, C_s)$.
- S is $\exists R_s.C_s$, Q is $\exists R_q.C_q$, R_s is an equivalent property of R_q, and $match(C_q, C_s)$.
- S is $\bigcap_i S_i$, Q is $\bigcap_j Q_j$, $\forall S_i \, \exists Q_j \; match(Q_j, S_i)$, and $\forall Q_j \, \exists S_i \; match(Q_j, S_i)$.
- S is $\bigcup_i S_i$, Q is $\bigcup_j Q_j$, $\forall S_i \, \exists Q_j \; match(Q_j, S_i)$, and $\forall Q_j \, \exists S_i \; match(Q_j, S_i)$.
- S is $\bar{s}$, and Q is $\bar{q}$, $match(q, s)$.
- S is an service input, Q is an instance of the input class description of S (note that this instance check may be performed in the Meta KB), and Q can be further decomposed and answered.

Semi-Match: $semimatch(Q, S)$ is an ordered binary relation. A query (sub)tree Q semi-matches a syntax tree S of a service's output class description if one of the following conditions is met.

- S is $\bigcap_i S_i$, Q is $\bigcap_j Q_j$, and $\forall S_i \, \exists Q_j \; match(Q_j, S_i)$.
- S is $\bigcup_i S_i$, Q is $\bigcup_j Q_j$, and $\forall S_i \, \exists Q_j \; match(Q_j, S_i)$.

The following is the query decomposition algorithm.

```
QueryResult decompose (Subtree Q) {
  if (domainKB.canAnswer(Q)) return domainKB.answer(Q);
  for each service S do
    if (match(Q,S)) return S.invoke();
  if (Q.isIntersectionOrUnion())
    QueryResult R = NULL;
    mark all operands of Q unmatched;
    for each operand O of Q do
      if (decompose(O)!=NULL)
        R = doIntersectionOrUnion(R,decompose(O));
        mark O matched;
    repeat
      for each Service S do
        if (semimatch(Q,S))
          R = doIntersectionOrUnion(R, S.invoke());
          mark all the operands used in S matched;
    until all operands matched or no more operands can be matched
    if all operand matched return R;
  return NULL;
}
```

Line 4 to 17 in the above algorithm is doing a horizontal decomposition using semi-match judgements in line 13. The judgements actually depend on the order of services in the for loop of line 12. For example, though at first one service may not be semi-matched, using the output of another semi-matched service as its input may enable it to be semi-matched. However, this order can not be determined ahead of time. Hence, line 11 to 16 repeats until a fix point is reached. In this algorithm, line 3, 8, 13 recursively calls match, semi-match and decompose functions and may lead to vertical decompositions.

4.3 Discussion

The above query decomposition method leaves some open problems. First, describing query and service capabilities using OWL-DL class descriptions is not enough. Work on DL query languages such as [11] may provide solutions for this.

Second, because of our syntactical approach, the current algorithm can not decompose some queries that can actually be decomposed at the semantic level. For example, $\overline{A \cap B}$ can not be answered using $\overline{A} \cup \overline{B}$ in the current algorithm.

Third, there may be more than one way to decompose the query and give the answer and it is possible that different services give different information about the same thing. How the decomposition engine chooses the best one according to the reputation and other attributes of the services is thus a problem. Research on trust and reputation on the Semantic Web may help solve this problem. In addition, the engine itself may learn from past experience to choose one.

Finally, even if the query can not be answered accurately, giving a close subset or superset of the query result may also be very helpful for the user. In SPortS, both syntax structure and DL inference engine can provide help on judging sub-class relation. We are now working on utilizing them to improve the current algorithm.

5 Recommendation of World Changing Services

In web portals, especially E-Commerce portals, world changing services may play a pivotal role because they can perform real world (business) transactions. Many current semantic portals are limited to only presenting domain knowledge. Adding world changing services will greatly enrich the portal's functionality and help user act based on contextual knowledge. However, only listing all available world changing services on the portal is far from an appropriate integration method. Users may be overloaded with lots of irrelevant or uninteresting services. SPortS solves the problem by listing only those services that are contextually relevant. The function is provided by another information providing service – Recommendation Service.

Intuitively, the semantic context that determines which service is relevant includes the concepts/classes that the user recently visits. For instance, in our running Apex-Lab-Portal example, when the user clicks a conference link to show its details, he/she is probably interested in those services that involve related concepts, such as conferences, papers, proceedings, etc at that moment. Borrow-Proceeding is such a service. However, with time, the user may gradually shift his/her focus to other concepts in the portal during his/her visit and the list of relevant services should change accordingly. We simulate this process by assigning temperatures to concepts that reflect their degrees of user attention. "Hotter" concepts receive more user visits and attention and the temperature will be dropped if the concepts are not visited by users for a while.

In SPortS, the semantic context is constructed based on an un-directional graph $G = (V, E)$ where V is the set of all the classes in the unified ontology of the Virtual KB and the edges in E are the semantic relations between classes. The graph can be obtained by converting the OWL-DL descriptions of the Virtual KB into an RDF graph. Currently in SPortS, we obtain the graph using a special algorithm that analyzes the OWL-DL class descriptions. For each vertex $c \in V$, a temperature $t(c)$ is assigned and for each edge $r \in E$ a thermal conduction factor $d(r)$ is assigned. All $t(c)$ are initialized to 0 and $d(r)$ are initialized according to the semantic types of r. For a path p consisting of edges $r_0, r_1, \ldots r_n$ in the graph, its thermal conduction factor $d(p) = \prod_{i=0}^{n} d(r_i)$. To measure the mutual influences between any pair of vertices c_1 and c_2, we define the maximum thermal conduction factor between them as $md(c_1, c_2) = max\{d(p) \,|\, p \text{ connects } c_1 \text{ and } c_2\}$ and we define $md(c, c) = 1$ for any c. Whenever a user visits some concepts by browsing a web page, the following procedure is used to update the temperatures of concepts:

```
void update_temperature () {
  let $C$ be the set of the currently visited concepts;
  let $G = (V, E)$ be the entire graph;
  for each $v \in V$, $t(v) = \alpha * t(v)$; // $\alpha$ is a cooling coefficient.
  for each $c \in C$ do
    for each $v \in V$ do
      $t(v) = t(v) + \Delta * md(c, v)$; // $\Delta$ is the temperature increment.
  for all $v \in V$ normalize $t(v)$ into the range of 0..1;
}
```

The semantic context then consists of all the concepts and their current temperatures. Based on this context, the relevance value of a semantic web service s is calculated as

$$rel(s) = \frac{t(c_0) + t(c_1) + \ldots + t(c_{n-1})}{n}$$

where c_i is the concept in the OWL-S description of s. The services are then ranked according to the relevance values $rel(s)$ and recommended to the user. In addition to the dynamic recommendation, SPortS also supports static recommendation that always recommends specific services for certain concepts. System administrators can utilize this feature to enforce business policies on service recommendation.

6 Conclusion and Future Work

In this paper, we have presented SPortS, an OWL-DL formalism based semantic portal system that integrates semantic web services into generated portals at the presentation level, the data level, and the function level. At the presentation level, semantic web services can be uniformly presented as first class citizens with domain knowledge. At the data level, knowledge from information providing services can be retrieved and composed together with domain knowledge to answer complex queries. In this way, potentially broader and more up-to-date information from services can be synthesized. At the function level, the recommendation of world changing services greatly enriches the generated portal's functionality and can help users act in the portal based on contextual knowledge.

Currently, the SPortS prototype system is limited in the following aspects. At the presentation level, the generated portal is only in HTML, which makes it difficult for the portal itself to be exposed as semantic web services. At the data level, due to the lack of an appropriate OWL query language and the difficulties in decomposing queries, we cannot fully utilize the information to answer more queries with better results. At the function level, the portal personalization is not yet designed in the current prototype. For example, the semantic context used by the service recommendation does not include any personalization information. Our future work will be focused on these aspects.

References

1. Staab, S., Angele, J., Decker, S., Erdmann, M., Hotho, A., Maedche, A., Studer, R., Sure, Y.: Semantic community web portals. In: Proceedings of the 9th World Wide Web Conference (WWW9), Amsterdam, Netherlands (2000)
2. Maedche, A., Staab, S., Stojanovic, N., Studer, R., Sure, Y.: SEAL – a framework for developing semantic web portals. In: Proceedings of the 18th British National Conference on Databases. Volume 2097 of LNCS. (2001) 1–22
3. R.Volz, D.Oberle, B.Motik, S.Staab: KAON server - a semantic web management system. In: Proceedings of the 12th International Conference on World Wide Web (WWW2003). Alternate Tracks - Practice and Experience, Budapest, Hungary (2003)
4. Corcho, O., Gmez-Prez, A., Lpez-Cima, A., Lpez-Garca, V., Surez-Figueroa, M.: ODESeW: automatic generation of knowledge portals for intranets and extranets. In: The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings. Volume 2870 of Lecture Notes in Computer Science., Springer (2003)
5. Lei, Y., Motta, E., Domingue, J.: Design of customized web applications with OntoWeaver. In: Proceedings of the international conference on Knowledge capture, ACM Press (2003) 54–61
6. Jin, Y., Decker, S., Wiederhold, G.: OntoWebber: model-driven ontology-based web site management. In: Proceedings of SWWS'01, The first Semantic Web Working Symposium, Stanford University, California, USA, July 30 - August 1, 2001. (2001)
7. F.Patel-Schneider, P., Hayes, P., Horrocks, I.: OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation, W3C (2004)

8. McIlraith, S., Son, T., Zeng, H.: Semantic web services. IEEE Intelligent Systems (Special issus on the Semantic Web) **16** (2001) 46–53
9. Horrocks, I.: The FaCT system. (1998) 307–312
10. Haarslev, V., Moller, R.: Racer system description. In: Proceedings of the First International Joint Conference on Automated Reasoning, Springer-Verlag (2001) 701–706
11. Horrocks, I., Tessaris, S.: Querying the Semantic Web: A formal approach. In: Proceedings of the 1st International Semantic Web Conference (ISWC2002). LNCS 2342 (2002) 177–191