2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing

Semantic Web Services Annotation and Composition based on ER Model

ChengZhi Xu，Peng Liang
State Key Laboratory of Software Engineering
Wuhan University
Wuhan, China
xcz911@gmail.com

Taehyung (George) Wang
Department of Computer Science
California State University
Northridge, USA
twang@csun.edu

Qi Wang, Phillip C-Y Sheu
Department of EECS
University of California
Irvine, USA
psheu@uci.edu

***Abstract*—Ontology makes traditional web service to be semantic web services, where semantic annotation can be employed to discover, reason and composite web services semantically. The construction of ontology is a key factor to implement semantic web service applications. Traditionally, ontology is constructed and maintained by domain experts with considerable effort, which is difficult to proceed in an engineering way and to reuse existing domain knowledge. In this paper, we propose to use the ER model, which is widely employed in database management, to construct domain ontology and semantic web services efficiently and effectively.**

Keywords-ER model; ontology; semantic web services; web service composition

I. INTRODUCTION

Web services are application program interfaces designed to support interoperable machine-to-machine interactions over the Internet. Web services can be registered, discovered and invoked by a third-party user, and be composed into a composite service with more complex functionality. The supporting protocols (SOAP, WSDL and UDDI) [5] of web services provide no capability to describe semantic information about services, which results in the difficulties in accurately retrieving and automatically composing web services. Semantic web service (SWS) [15] was proposed to address these issues by providing a mechanism to construct machine-understandable services using ontology (i.e. ontological annotation of services). Semantic Web and semantic web service are based on SWS, and OWL-S (Web Ontology Language for Services) [14], which provide the semantic annotation mechanism for web services. SWS are constructed based on knowledge base and knowledge representation, which makes it possible for a machine to understand the user requests.

Web services can be generally classified in two types [9]: simple services, which do not depend on other services; and composite services, which are composed by simple services in certain logical relationships. Due to the functional limitation of simple web services, composite web services are widely used in practice to satisfy user requirements.

Web services composition can be conducted in either static or dynamic ways [17]. In static composition, user pre-defines a process model which includes activities and data dependencies among the activities. Standard process languages have been proposed to define a process model, such as Business Process Execution Language (BPEL) [1]. But BPEL is not sufficient to cope with the situation when it is required to retrieve a web service from a large repository. Orriens *et al*. proposed a model driven approach for service composition [18]. Their model separates the composition of logic and specification, and can be mapped directly to the BPEL. The shortcoming of this approach is that it requires integrating a planning tool which is difficult to be customized and applied in practice. Dynamic composition can be performed automatically by providing constraints and relationships among services without pre-defining a process model [19]. A typical example of dynamic composition is constructed based on artificial intelligence (AI) planning, where web services are mapped to actions by formal representations. A composition plan is generated dynamically by formal reasoning, such as a rule-based planning [16]. The deficiency of an AI planning method is difficulty in evaluating and validating plans. A detailed survey on web service composition can be found in [7].

Both static and dynamic compositions are heavily based on domain knowledge which is employed to generate logical relationships among web services and data. In large and complex system, domain knowledge is scattered around the entire system that is possessed by various stakeholders. It is a great challenge to construct and maintain domain knowledge with extensibility, and heterogeneous knowledge representation frameworks make this situation more complex [3]. In this paper, we propose to use the ER (entity-relationship) [4] model to construct domain ontology, with which the SWS are annotated, reasoned and composed. The proposed approach can explore the semantic resources, which widely exist in most of (legacy) information systems, and construct accepted knowledge base effectively. We also demonstrate that SWS and OWL/OWL-S can achieve the goal of automatic SWS composition.

This research has been partially supported National Science Foundation of China (NSFC) under grant No. 60873007 and 111 Project of China under grant No. B0737

978-0-7695-4049-8/10 $26.00 © 2010 IEEE
DOI 10.1109/SUTC.2010.73

The rest of this paper is organized as follows. In Section 2, the background of this work, including ontology language and SWS description language are reviewed and discussed. The transformation rules from ER to OWL and OWL-S are presented in Section 3. The different transformation scenarios and issues of SWS composition based on ER model are further discussed in Section 4. The application of our approach is demonstrated by a case study in Section 5. The paper concludes with future work in Section 6.

II. BACKGROUND

The OWL (Web Ontology Language) [2] and its variations, knowledge representation languages for authoring ontology, are endorsed by the W3C. OWL DL was designed to provide the maximum expressiveness possible while retaining computational completeness and decidability. OWL-S, an ontology built on top of OWL, is the semantic markup (annotation) language for WS. OWL-S is a combination of web services and semantic web, which enables machines to automatically and semantically discover, invoke, compose, and monitor services under specified constraints. There are three components in OWL-S: *ServiceProfile*, describing what function the service provides; *ServiceModel*, describing how the service is implemented; and *ServiceGrounding*, describing how to access the service. The standards of web services standards (WSDL, UDDI) enable automatic web service description, discovery, and binding, and OWL-S makes web serivces intelligent.

Web services become intelligent by introducing semantic descriptions such as ontology into the services, so that machine can automatically recognize, select, reason and compose SWS. The underlying reason is that ontology can represent domain knowledge in formal logic (e.g. description logic), including the classes, relationships between classes, and constraints, etc. The difficulty is that ontology construction and maintenance are non-trivial tasks which require considerable human effort [12]. Our approach to address this issue is to use ER model in order to construct SWS ontology. In this case, the ER model is the underlying conceptual model for most of information systems (local and web-based). The ER modeling, a classical database modeling method, is used to produce a conceptual schema or semantic data model of a system, often a relational database. This model has been employed successfully as a de-factor standard in database modeling for decades. In other words, most of databases models, which are now being upgraded to web-based systems, are based on ER. This makes the ER model a perfect candidate for constructing ontology for SWS by reusing the knowledge which has been recognized in certain domain for a long time. Consequently, it is a research question how to transform the ER model into OWL and OWL-S, which are ready for the construction of SWS. In the next section, we present transformation rules from ER to OWL and OWL-S.

III. SEMANTIC WEB SERVICES BASED ON ER

A. Transformation from ER to OWL

The ER model is an abstract and conceptual representation of data and their relationships. The model does not include all the semantic information about a domain, but useful semantic information can be retrieved from the ER model [8], and mapped to OWL. In this paper, we adopt the OWL DL as an ontology specification language, which has maximum expressiveness while retaining computational completeness and decidability. The basic concepts in OWL include: *Class*, abstract of things; *subClass* of certain class; *Individuals* of class; *Property* between individuals; and *subProperty* of certain property. An example of OWL representation is shown in Figure 1, in which **Person** is a *Class*; **hasChild** is a *Property*; **Tom** is an *Individual* of Class **Person**. There are two types of properties: *ObjectProperty* (the property is a Class) and *DataTypeProperty* (the property is a DataType).

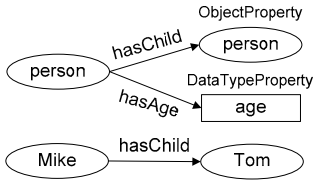

Figure 1. An example of OWL representation.

Classes are major players in OWL, which represent entities in certain domain. The entities in ER model can be readily mapped to the *Classes* in OWL. The shortage of ER model is that there is no *subClass* relationship, but we can create the subclasses by attribute-based classification. For example, in a wine ER model, the entity **Wine** has attributes, **color**, **sweetness**, **vintage** and **region**, in which **color** (**red** | **rose** | **white**) and **sweetness** (**dry** | **semi-dry** | **semi-sweet** | **sweet**) have discrete values, then we can create (define) the subclasses of entity **Wine** based on these attribute values, such as **RedWine** and **DryWine**, etc. Note that these subclasses are not the subclass in common sense, because they essentially represent the subset of the instances of class **Wine**, and they have not their own distinguished attributes. But they can be used to construct the basic ontology, and facilitate the reasoning based on that.

The other important concept in OWL is *Property*, which is corresponding to the property of entity in ER model. For example, the entity **Wine** has properties: **hasColor** (*string*), **hasSweetness** (*string*), **vintageOf** (*integer*), and **locatedIn** (*Region*). The first three properties are *DataTypeProperties*, and the last one is an *ObjectProperty*. The **locatedIn** property links the entity **Wine** with another entity **Region**. In the ER model, there is a key property in an entity, *ForeignKey*, whose attribute

should be an *ObjectProperty* since the *ForeignKey* denotes the relationship between entities, which corresponds to the relationship between *Classes* in OWL. The relationships in the ER model can only be represented in the form of 1:1, 1:*n*, and *n*:*m*, while the relationships in OWL can represent more meaningful information, which can be used for the services reasoning and composition.

During the transformation from the ER model to OWL ontology, the rich semantics of OWL DL representation should be explored in order to describe the semantics in the ER model, such as *equivalentClass*, *intersectionOf* class, *sameAs* individual, *differentFrom* individual, *equivalentProperty*, *inverseOf* property, etc. More semantic information ontology can deliver, more easily the automatic service reasoning and composition can be achieved. A mapping definition from the concepts in the ER model to OWL is shown in Table 1, with an example in Figure 2.

TABLE I. MAPPING FROM ER MODEL TO OWL

ER	OWL	Semantic Enrichment Elements
Entity	Class	subclass, equivalentClass, …
Common Property	DataTypeProperty	equivalentProperty, inverseOf, …
Foreign Key	ObjectProperty	equivalentProperty, inverseOf, …

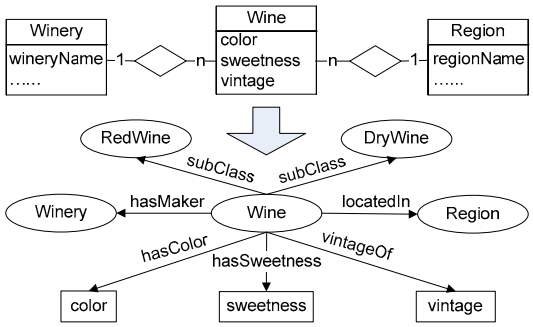

Figure 2.Transformation from ER model to OWL.

B. Transformation from ER to OWL-S

The purpose for transforming the ER model to ontology is to construct SWS. OWL-S is one of major ontology description languages for SWS. In the three components of OWL-S as mentioned in Section 2, the *ServiceModel* describes the way the service being implemented. *Process* is a subclass of *ServiceModel*, and *Process* has three subclasses: *AtomicProcess*, *SimpleProcess*, and *CompositeProcess*. *AtomicProcess* is the process which can be directly invoked, and cannot be decomposed further. *CompositeProcess* is composed of a set of *AtomicProcesses* by following certain control structures. *SimpleProcess* is also composed by several *AtomicProcesses*, and it is normally abstracted as a single-step action. *SimpleProcess* can be regarded as a static composition, in which the control structures are fixed, while *CompositeProcess* is composed dynamically, in which the control structures are generated at runtime.

We construct the SWS described by OWL-S using the ER model in the following steps. First, we create a query service for single entity, which is similar to a single-table query in SQL. For example, to query the **Wine** which satisfies the condition **color**=**red**, a service can be described in OWL-S as an *AtomicProcess*. Similarly, we can use *SimpleProcess* to describe a query service for a view-based query in SQL. Although a view-based query looks like a single-table query, but view is usually constructed based on a multi-table query or a nested query, which is essentially a multiple-step operation. *CompositeProcess* service provides input and output, and a semantic relationship between them. But the execution steps of *CompositeProcess* are generated dynamically. As soon as the intermediate steps are fixed, *CompositeProcess* can be transformed into *SimpleProcess*. Note that an *AtomicProcess* service can query the individuals of certain entity, and it can also query the individuals which are related entities to the queried entity. OWL-S itself does not provide reasoning facility, but the elements in OWL-S are imported from OWL, including the input and output of process. OWL is capable of reasoning, which can be used for automatic service composition.

IV. SEMANTIC WEB SERVICES COMPOSITION BASED ON ER

SWS can provide more complex functionality when composed effectively. Current methods for SWS composition require human intervention (e.g. domain experts), which also need to understand domain knowledge and underlying data structure. This makes SWS composition a challenging task. The method proposed in this paper addresses these issues by exploring domain knowledge in the ER model, and use it to create logical relationships among services. It also support partial service reasoning by the ontology corresponding to the entities in the ER model, and achieve service composition automatically.

For single entity query service, we invoke directly the related *AtomicProcesses*. For the query service related to multiple entities, we first check whether a corresponding *SimpleProcess* exists or not, if no such process available, then we generate a composite service dynamically. We present the scenario of dynamic service composition according to two ER models in the next subsections.

A. ER Transformation in Tree View

When the ER model is in a tree view, it means there is no cycle in the model. In such ER model, the relationship path between two entities is unique. When composing the service that queries the related classes of a certain class and

no such *SimpleProcess* available for the query, we can reason composite service based on a relationship path between the entities in the ER model corresponding to the classes. In Figure 3, we show an example for the Teaching Management System. The upper half in this figure is the ER model, and the lower half is the transformed OWL model.

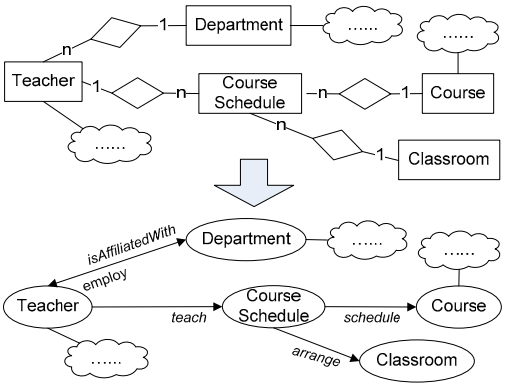

Figure 3. ER transformation in tree view.

Example 1: *A user wants to query the* ***courses*** *provided by a* ***computer science department***. The input is the name of the department, and output is the names of the courses name provided by the department. First, we try to search for *AtomicProcesses* that can fulfill this requirement. If there is no such *AtomicProcesses*, then a *CompositeProcess* needs to be composed. Through the analysis of the ER model in Figure 3, there is only one relationship path from the entity **Department** to the entity **Course**: **Department → Teacher → Course Schedule → Course**. Then, we can search for *AtomicProcesses* that match with these entities. The example search results are shown in the Table 2 (WS_i denotes a WS):

TABLE II. EXAMPLE OF ATOMICPROCESSES SEARCH RESULTS

WS	*AtomicProcess* Entity	Input	Output
WS_1	Department	Department Name	Teacher ID
WS_2	Teacher	Teacher ID	Course Schedule ID
WS_3	Course Schedule	Course Schedule ID	Course ID
WS_4	Course	Course ID	Course Name

The four WS can be composed sequentially in a service chain: $WS_1 \rightarrow WS_2 \rightarrow WS_3 \rightarrow WS_4$. Apparently, this composite service is composed by the matching of input and output of WS, which is only a necessary condition for this composition. The sufficient condition is that these *AtomicProcesses* depend on the relationship path and entities in the ER model: **Department→Teacher→Course Schedule→Course**. In other word, if there is a WS_x, which satisfies the input and output matching condition, but does not belong to the entities in the relationship path, it can not be selected as part of the composition WS since it does not depend on the entity in this relationship path.

The composition service described above can also be optimized. For example, if there is a *SimpleProcess* which can substitute several *AtomicProcesses* in the relationship path, then the execution efficiency will be increased dramatically. Suppose that there is *SimpleProcess* WS_5, the input is **Teacher Name** and output is **Course Name**, then WS_5 can substitute the WS_2 and WS_3, the service chain will be $WS_1 \rightarrow WS_5 \rightarrow WS_4$. This service composition process can be conducted automatically under the guidance of the ER model as the steps below:

- Step1: analyze user requirements, and identify the entities (*Class* in OWL) related to the input and output;
- Step2: identify a unique relationship path in the ER model as the service composition path (chain) using input and output entities;
- Step3: retrieve the entities (mapped to the *Classes* in OWL) in this relationship path, and collect *AtomicProcesses* which correspond to these entities as the candidate services for composition;
- Step4: construct a composite service chain by the input and output matching of services *AtomicProcesses*;
- Step5: optimize the composite service chain, e.g. substitute some *AtomicProcesses* by a *SimpleProcess*.

B. ER Transformation in Network View

When the ER model is in a network view, there is one or more cycles in the model. In such ER model, the relationship path between two entities is not unique. In such case, a relationship path should be selected from available relationship paths. Figure 4 shows another ER model example in network view for the Teaching Management System.

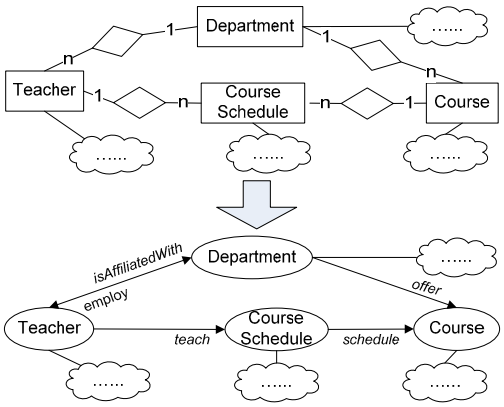

Figure 4. ER transformation in network view.

Example 2: *A user wants to query the **courses** taught by **teacher Tom***. The value of input is the name of the teacher, and the value of output is the courses name taught by this teacher. Through the analysis of the ER model in Figure 4, there are two relationship paths from the entity **Teacher** to **Course**: L_1: **Teacher→Department→Course**, and L_2: **Teacher→Course Schedule→Course**. We should select one relationship path according to the user requirement. Through detailed analysis of Example 2, we found that there is *Teach* relationship from **Teacher** to **Course**, which is an *ObjectProperty* defined in the OWL class, **Teacher**. Then, we can further identify the relationship path that contains the *Teach* relationship from the candidate relationship paths. The *Teach* relationship is found in the relationship path L_2: **Teacher → Course Schedule**, and consequently L_2 is selected as the relationship path for the composition of service chain.

Similarly, if the user requirement in Example 2 is modified as: *A user wants to query the **courses** offered by the **department** which **teacher Tom** is affiliated with*. Then the relationship path that contains the *isAffiliatedWith* relationship should be selected, that is L_1: **Teacher → Department**.

C. Uncertainty in ER

A well-defined ER model is a fundamental condition for correct reasoning of services, and uncertainty issues in the model should be eliminated as much as possible. We introduce the uncertainty issues by the example below. The ER model in Figure 5 specifies the *m*:*n* relationship between **Teacher** and **Course**, and **Course** and **Student**. This model can not reflect the accurate relationship between **Teacher** and **Student** since same course can be taught by different teachers. This model only specifies the relationship between **Student** and **Course** without specifying who teaches this **Course**. Therefore, this model cannot implement the service reasoning from **Teacher** to **Student**, or vice verse. For example, *a user want to query all the **students** who attend the **courses** taught by **teacher Tom***, or *a user want to query the **courses** attended by **student John***.

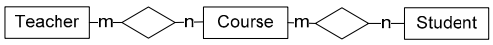

Figure 5. ER model from Teaching Management System.

This example ER model can be improved by adding an entity **Course schedule** as shown in Figure 6, which shows the relationship among **Teacher**, **Student** and **Course**. By the **Course schedule**, the accurate relationship path between **Teacher** and **Student** can be constructed, which can be used for the service composition reasoning from **Teacher** to **Student**, and vice verse.

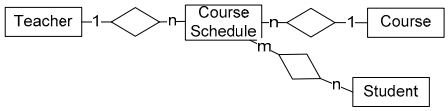

Figure 6. Improved ER model from the Teaching Management System.

This ER model can be improved by adding an entity **Course schedule** as shown in Figure 6, which includes the relationship among **Teacher**, **Student** and **Course**. By the **Course schedule**, the accurate relationship path between **Teacher** and **Student** can be constructed, which can be used for the service composition reasoning from **Teacher** to **Student**, and vice verse.

D. Multi-input in Services Composition

The examples described above are all single-input user requirements. There are also many user requirements which require multi-input. In such requirements, input conditions usually have *unionOf* and *intersectionOf* relationships. The steps for composing multi-input service are similar to the way of composing single-input service. First, search for the *AtomicProcesses* and *SimpleProcesses* matched with the requirements, and if there is no such process available, compose the *CompositeProcess*. A simple method is to decompose the user requirements into sub-requirements with single-input, and then search for *AtomicProcesses* and *SimpleProcesses* matched with these sub-requirements, and finally compose these services by *unionOf* and *intersectionOf* operations based on the output of the services. Similar to single-input service composition, *SimpleProcess* can be selected by input and output matching to substitute intermediate *AtomicProcesses* in order to improve the execution efficiency.

E. Architecture of SWS Composition System based on ER

The general architecture of SWS composition system based on ER is presented in Figure 7.

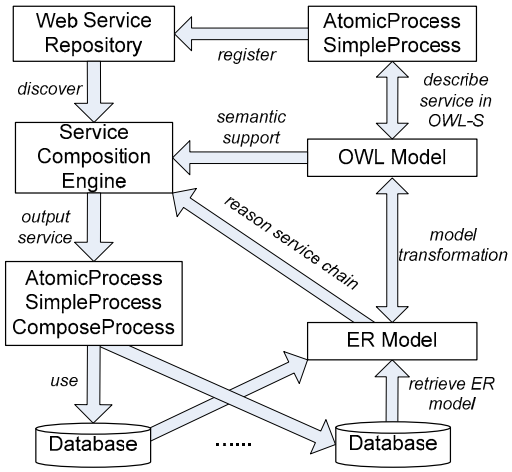

Figure 7. An architecture of semantic Web Services composition system based on ER model.

This architecture also describes the process of integrating the database and the web service based systems via the ER model. The dashed boxes in this figure represent core work such as ER and OWL model for SWS annotation, and Service Composition Engine for SWS composition.

V. CASE STUDY

In this section, we demonstrate how our approach can be applied in concrete applications by a case study. The scenario of this case study is described as follows: A restaurant originally manage their menu in a menu database system, which includes data for the wines provided by this restaurant. To satisfy wine requests of various customers, this restaurant tries to provide customers more flexibility by introducing the SWS to query wine from various providers. In this case study, SWS for wine query is annotated and composed based on the ER model. The ER model of the wine database is shown in Figure 8.

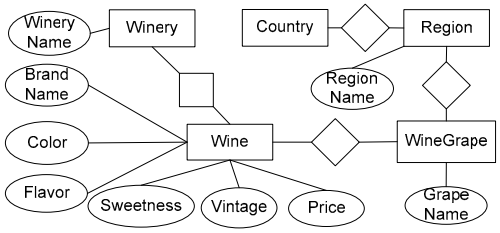

Figure 8. ER model of the Wine database.

This ER model is translated into OWL ontology model based on the translation rules presented in Section 3. The *Entities* in ER model are translated into *Classes* in OWL; the *attributes* of entity are translated into *DataTypeProperties* in OWL; and the *Relationships* between entities are translated into *ObjectProperties* in OWL. Partial translated OWL description of the wine ontology is shown below:

```
<owl:Class rdf:ID="Wine">
    <owl:DatatypeProperty rdf:ID="hasColor">
        <rdfs:domain rdf:resource="#Wine"/>
        <rdfs:range rdf:resource="string"/>
    </owl:DatatypeProperty>
    <owl:ObjectProperty rdf:ID="hasMaker">
        <rdf:type rdf:resource="Functional Property"/>
        <rdfs:domain rdf:resource="#Wine"/>
        <rdfs:range rdf:resource="#Winery"/>
    </owl:ObjectProperty>
    …
</owl:Class >
```

The *subClasses* can be created by the discrete value of attributes (*DataTypeProperties*) of entity. For example, the *Class* **Wine** has *subClasses* **WhiteWine**, **DryWine**, and **TableWine**, etc. The OWL description of the **WhiteWine** *subClass* is shown below:

```
<owl:Class rdf:ID="WhiteWine">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Wine"/>
        <owl:Restriction>
            <owl:onProperty
rdf:resource="#hasColor"/>
                <owl:hasValue rdf:datatype="string"> White
                                        </owl:hasValue>
        </owl:Restriction>
    </owl:intersectionOf>
    …
</owl:Class>
```

The *relationships* between *subClasses* can also be established based on the attribute values of entity when the *subClasses* are created. For example, **SweetWine** is a *equivalentClass* of **DessertWine** since they have same value of attribute *sweetness=sweet*. **DryRedWine** is the *intersectionOf* **DryWine** and **RedWine** since the attributes of **DryRedWine** *sweetness=sweet*, and *color=red* is the *intersectionOf* the attributes *sweetness=sweet* of **DryWine** and *color=red* of **RedWine**. Parts of these *relationships* in OWL are shown below:

```
<owl:Class rdf:ID="DryRedWine">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#DryWine" />
        <owl:Class rdf:about="#RedWine" />
    </owl:intersectionOf>
    …
</owl:Class>
```

The final OWL model of the wine ontology by following the translation steps above is shown in Figure 9.

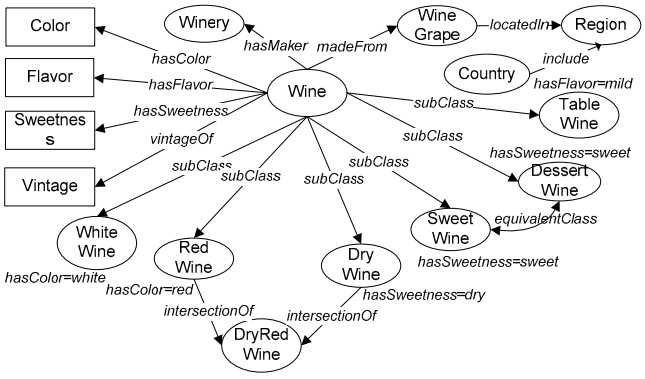

Figure 9. OWL model of the Wine Ontology.

This OWL ontology can be used to annotate the query WS constructed by the attributes of entities using OWL-S. Now, the original user requirement is *to request the **DryRedWine** produced in **Italy***. This requirement can be further decomposed into two sub-requirements: *query the **wine** produced in **Italy***, and *query the **dry** and **red wine***.

Through the analysis to the first sub-requirement, the reasoner can detect that the service input is **Country**, and output is **Wine**, and the relationship between the input and output is **LocatedIn**. The relationship path for this service by the analysis to the wine ontology in OWL is: **Country → Region → WineGrape → Wine**, then the system will search for the WS which can implement this service relationship path.

Through the analysis to the second sub-requirement, the reasoner can detect that the **DryRedWine** is the *intersectionOf* **DryWine** and **RedWine**, in which the attribute of **DryWine** *hasSugar=dry*, and the attribute of **RedWine** *hasColor=red*. Consequently, the input of the second sub-requirement is *hasSugar=dry* and *hasColor=red*, and output is **Wine**, then the system will search for the WS which can implement the input and output. The final decomposition from the original requirement to services is shown in Figure 10.

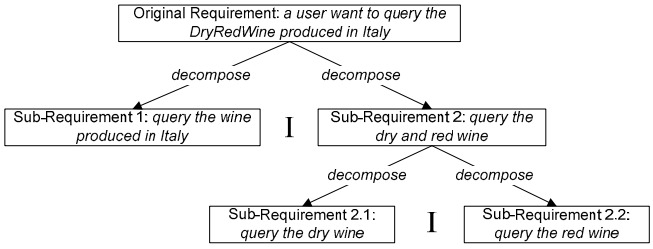

Figure 10. Decomposition of the original requirement to services.

The original requirement is decomposed into three sub-requirements (1, 2.1, 2.2), and each one corresponds to one WS: WS_1, the input is **Country** and output is **Wine**; WS_2, the input is **Sweetness** and output is **Wine**; WS_3, the input is **Color** and output is **Wine**. In the end, the system will search for *AtomicProcesses*, *CompositeProcesses* or service relationship path to implement WS_1, WS_2 and WS_3 by the matching of service input and output. The original requirement can be implemented by the intersection of the results got by these three WS.

VI. Conclusions and Future Work

In this paper, we propose the SWS construction (annotation) and composition method based on the ER model, which transforms existing knowledge into ontology in OWL for the service annotation. This method also provides an integrated framework for SWS discovery, reasoning and composition. There are also other methods for domain knowledge construction. Gong *et al*. propose a method to construct a relationship topology among web services using input and output domain matching of web services [9]. The ontology constructed in [9] is not accurate since the domain matching is different from the semantic matching. The method in [10] can generate a business process based on the logical business relationships. This method can construct sound semantic ontology for service annotation, but it requires lots of pre-effort to understand the business relationships, which is great burden for normal service users.

Our method relies on the ER model, which is sufficient to address most of web services requests [13], and is especially beneficial for small and medium enterprises. Most of business process information and data structures can be automatically retrieved from the ER model. Complex service composition, which requires decision-making, can also be partially supported by our method with human intervention.

When a sequential service chain can not satisfy complex user requirements, other control structures such as switch and loop structures should be introduced. The introduction of multiple control structures leads to the introduction of other composition algorithms. Currently, there is no effective approach to express semantic information for algorithms. It is necessary to include human intervention in the composition process. In such situation, a service composition language is used to define algorithms, and the operational objects in the algorithms are web services to be retrieved. The frequently-used composition languages are BEPL [11], SOBL [6], and SPL+ [20], etc.

There is still enormous work to be done for improving the SWS composition mechanism based on the ER model. We outline future work in the following points: (1) ER model is frequently related with and easily mapped into SQL queries in databases as the examples presented in this paper. But user requirements are not always SQL-like statements, so how to analyze the user requirements and transform the user requirements into SQL (or SQL-like) queries is a challenge task; (2) there is only one relationship predicate *equal* employed in our work for the SWS composition, on which all the input and output matching and reasoning are based. Other predicates should be introduced to enrich the application scope of this method; (3) all the examples presented in this paper are the query operations without modifying the data in database. For other operations, such as data update, the execution process should be secured in transaction, in order to retain the data consistency in case of execution failure of composite service; (4) the ER model is the underlying conceptual model for relational databases, on which most of (web) information systems are based. With the development of database technology, other database model, e.g. object

model and space model become more popular in database modeling. It is also an interesting topic to investigate how to transform (map) these emerging models into ontology models for the SWS annotation and composition.

ACKNOWLEDGMENT

This research has been partially supported by National Science Foundation of China (NSFC) under grant No. 60873007 and the 111 Project of China under grant No. B0737.

REFERENCES

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, et al. Business process execution language for web services, version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.

[2] S. Bechhofer, F. Van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. Stein, et al. OWL web ontology language reference. W3C Recommendation, 10: 2006–01, 2004.

[3] W. Borst. Construction of engineering ontologies for knowledge sharing and reuse. PhD thesis, Universiteit Twente, 1997.

[4] P. Chen. The entity-relationship modelŧtoward a unified view of data. ACM Transactions on Database Systems, 1(1): 9–36, 1976.

[5] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana, I. Center, and Y. Heights. Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, 6(2): 86–93, 2002.

[6] D. Deng, G. Zhang, Z. Gong, Z. Guo, and P. Sheu. Semantic programming of web-enabled database applications. In Proceedings of the IEEE International Workshop on Semantic Computing and Applications (IWSCA), pages 51–60, 2008.

[7] S. Dustdar andW. Schreiner. A survey on web services composition. International Journal of Web and Grid Services, 1(1): 1–30, 2005.

[8] M. Gogolla and U. Hohenstein. Towards a semantic view of an extended entity-relationship model. ACM Transactions on Database Systems, 16(3): 369–416, 1991.

[9] H. Gong, S. Wang, Q. Wang, and P. Sheu. Synthesis of relational web services based on SCDL. In Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), volume 2, 2008.

[10] C. Hu, M. Wu, G. Liu, and D. Xu. An approach to constructing Web service workflow based on business spanning graph. Journal of Software, 18(8): 1870–1882, 2007.

[11] M. Juric. Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition. Packt Publishing, 2006.

[12] L. Khan and F. Luo. Ontology construction for information selection. In Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), pages 122–127, 2002.

[13] S. Kim and M. Rosu. A survey of public web services. Lecture notes in computer science, pages 96–105, 2004.

[14] D.Martin,M. Burstein, J. Hobbs, O. Lassila, D.McDermott, S.McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, et al. OWL-S: Semantic markup for web services. W3C Recommendation, 2004.

[15] S. McIlraith and T. Zeng. Semantic web services. IEEE intelligent systems, 16(2): 46–53, 2001.

[16] B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Composing web services on the semantic web. International Journal on Very Large Data Bases, 12(4): 333–351, 2003.

[17] N. Milanovic and M. Malek. Current solutions for web service composition. IEEE Internet Computing, 8(6): 51–59, 2004.

[18] B. Orriens, J. Yang, M. Papazoglou, et al. Model driven service composition. Lecture Notes in Computer Science, pages 75–90, 2003.

[19] J. Rao and X. Su. A survey of automated web service composition methods. Lecture Notes in Computer Science, 3387: 43–54, 2005.

[20] G. Zhang, S.Wang, C. Xu, Z. Gong, and P. Sheu. A semantic programming language SPL+ - a preliminary report. In Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), volume 2, 2008.