An Approach for Service Discovery and Recommendation using Contexts

Hua Xiao and Ying Zou

Abstract Given the large amount of existing Web services nowadays, it is time-consuming for users to find appropriate Web services to satisfy their diversity requirements. Context-aware techniques provide a promising way to help users obtain their desired services by automatically analyzing a user's context and recommending services for the user. Most existing context-aware techniques require system designers to manually define reactions to contexts based on context types (*e.g.*, location) and context values (*e.g.*, Toronto). Those context-aware techniques have limited support for dynamic adaptation to new context types and values. Due to the diversity of user's environments, the available context types and potential context values are changing overtime. It is challenging to anticipate a complete set of context types with various potential context values to provide corresponding reactions. In this chapter, we present an approach which analyzes dynamic changing context types and values, and formulates search criteria to discover desired services for users. More specifically, we use ontologies to enhance the meaning of a user's context values and automatically identify the relations among different context values. Based on the relations among context values, we infer the potential tasks that a user might be interested in, then recommend related services. A case study is conducted to evaluate the effectiveness of our approach. The results show that our approach can use contexts to automatically detect a user's requirements in given context scenarios and recommend desired services with high precision and recall.

Hua Xiao
IBM Canada Laboratory, Markham, Ontario, Canada, e-mail: huaxiao@ca.ibm.com

Ying Zou
Dept. of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada
e-mail: ying.zou@queensu.ca

1 Introduction

With the growing prevalence of Service Oriented Architecture (SOA), more Web services become available for users to enrich their daily online experience. It is time consuming for users to find appropriate services to satisfy their various requirements. Context-aware techniques provide a promising way to help users obtain their desired services by automatically analyzing a user's context and recommending services for the user. Specifically, a context characterizes the situation of a person, place or the interactions between humans, applications and the environment [1]. One way to model contexts is to use pairs of context types and context values. A context type describes a characteristic of the context. A context type is associated with a specific context value. For example, "location", "identity", and "time" are context types of a user. "New York" is a context value of the context type "location". Furthermore, a context scenario is the combination of different context types with specific values to reflect a user's situation. To manage different context types and values captured by a context-aware system, a context model is used to specify the relations and the storage structure of various context types and values.

Context-aware systems are designed to react to a user's context without their intervention. A context aware system generally consists of two parts: sensing a context scenario, and adapting the system to the changing context scenario. Most context-aware systems require the designer of context-aware systems to predict the context types. Moreover, the designer needs to manually establish the relation between the sensed context scenario and the corresponding reactions in the form of IF-THEN rules which specify how a system should respond to context changes. However, due to the diversity of user's environments, the available context types and potential context values are changing overtime. For example, if a user travels from her home to another city "Los Angeles". The user's environment changes accordingly. The location is changed from "home" which is a context value of context type "location" to "Los Angeles". And the activity of the user in the new location is "driving", whereas the context-aware system may not detect the activity of the user when she was at home. It is challenging to anticipate a complete set of context types with various potential context values to provide corresponding reactions. Moreover, fixed rules are not flexible enough to accommodate the changing environment and various personal interests. To recommend services for a context scenario, this chapter presents an approach which analyzes dynamic changing context types and values, and formulates search criteria to discover desired services for users. Different from existing approaches which depend on static context models to know the relations among context types (or values) and use predefined rules to infer user's requirements, we seek an automatic approach to recognize the relations between context values and a user's requirements. For example, luxury hotel and limited budget are two context values in conflict. Therefore, the services for booking luxury hotels are automatically filtered when a user has limited budget. We expect that such relations can be used to express more accurate searching criteria that better reflect a user's context. When a new value of a user is detected, our approach can automatically compute the relations between the new context value with existing context

values. Instead of manually defining IF-THEN rules using specific context values as the traditional context-aware systems [2], our approach automatically identifies the semantic relations among context values to infer user's requirements. Then we generate service searching criteria based on user's requirements to discover and recommend services. This book chapter extends our earlier work [3] published in the International Conference on Web Services (ICWS) 2010. We enhance our earlier publication at ICWS 2010 in the following aspects:

1. Improve the algorithm for identifying the relations among different context values by considering domain knowledge and semantics of phrases used to describe the meaning of context values;
2. Extend the approach for generating service searching criteria to search for desired services; and
3. Conduct a larger case study to evaluate our extended approach.

To facilitate the presentation of this chapter, we use the following travel scenario as an illustrative example throughout this chapter. Tom is a graduate student living in Toronto. Tom is interested in watching Hollywood movies and National Basketball Association (NBA) games. Especially, Tom is a fan of Kobe Bryant who is an American professional basketball player and plays for the NBA team, Los Angeles Lakers. Tom plans to travel to Los Angeles and spend his vacation in Los Angeles next month. When examining the context in this scenario, we find that some contextual information can be helpful for Tom to plan his trip. For example, as a graduate student who has low income, Tom might prefer budget hotel for the trip. As a fan of NBA, Tom might be glad to know the NBA game schedules of "Los Angeles Lakers" when he is in Los Angeles.

The remainder of this chapter is organized as follows. Section 2 gives an overview of our approach. Section 3 introduces the background of ontologies. Section 4 presents our approach to find matching ontologies from ontologies databases. Section 5 discusses the details of inferring relations among different context values. Section 6 presents our approach that identifies user's requirements in a given context scenario and generates searching criteria to search for services. Section 7 and section 8 present an overview of our prototype and discuss the case study. In section 9, we present the related work. Finally, Section 10 concludes the chapter and presents the future work.

2 Overview of Our Approach

Figure 1 gives an overview of our approach. Context types and context values can be dynamically added and removed to reflect a user's situation. The value of a context type can also be changed over time. To correctly model relations among context values, it is critical to understand the semantic meanings of each context value. Ontologies capture the information related to a particular concept using expert knowledge. To identify the semantics of a context value, we search for publicly available

ontologies to extend the meaning of the context value. Figure 2 illustrates an example ontology for defining the concept “Los Angeles”. In particular, “Los Angeles” is a context value for the context type “Location”. The ontology of “Los Angeles” shown in Figure 2 expands the semantic meaning of “Los Angeles” with additional information, such as “Geographic Location”, “Sports Team”, and “Tourist Attraction”. When a new context value for a user is detected, our approach automatically searches for ontologies that expand the se-mantic meanings of the new value and computes the relations with other context types and values.

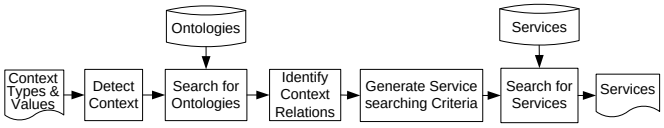

Fig. 1: Steps for context-aware service recommendation

We use the identified context relations to discover user’s requirements for a given context scenario and generate the corresponding service searching criteria. For example, when the semantics (*i.e.*, ontologies) of several context values share a same concept, the common concept might reflect the potential requirements of the user. In the travel scenario, Tom is going to travel to “Los Angeles”, and he is interested in watching NBA games. The ontologies of “Los Angeles” and “NBA” have a same concept “Los Angeles Lakers”. It indicates a high likelihood that Tom would be interested in watching the basketball game played by “Los Angeles Lakers”. Finally, we use the generated service searching criteria to discover and recommend services to the user.

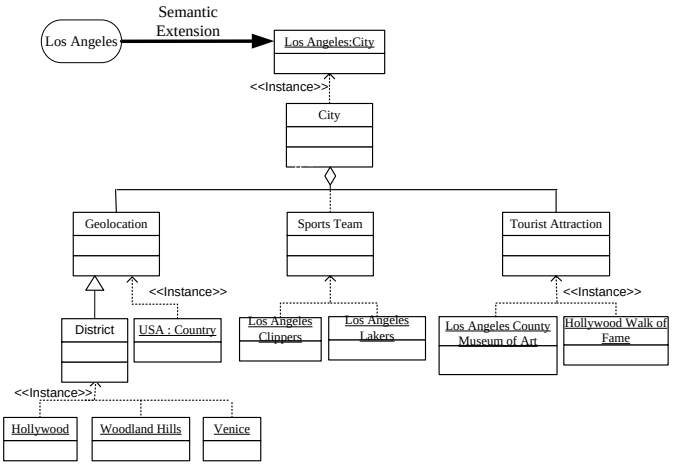

Fig. 2: An example of extending context value using ontology

3 Background of Ontology

Ontologies are described using ontology specification languages, such as Web Ontology Language (OWL) [4], Resource Description Framework (RDF) [5] and DAML+OIL [6]. We use ontologies to understand the meanings of context values. The ontologies found for context values can be described in different ontology specification languages. To ease the inference of the relations among context values, we define a simplified model which summarizes the structures and concepts of ontologies needed for our context analysis. Figure 3 illustrates the major entities in our ontology definition model. Essentially, our ontology definition model contains the following four major components.

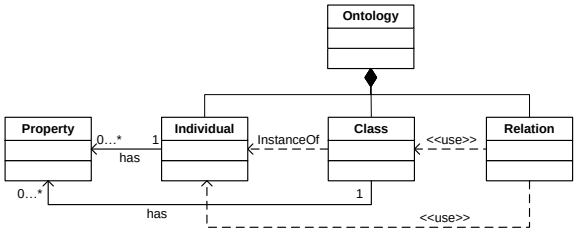

Fig. 3: Major entities defined in ontologies

- Class is an abstract description of a group of concepts with similar characteristics. A class has a name and a set of properties that describe the characteristics of the class. For example shown in Figure 2, “Tourist Attraction” as a class contains the common characteristics of tourist attractions. Class is also called “concept”, “type”, “category” or “kind” in ontology specification languages.
- Individual refers to an instance of a class. For example, “Hollywood Walk of Fame” in Figure 2 is an instance of class “Tourist Attraction” and therefore it is an individual.
- Property describes an attribute of a class or an individual. A property can also be composed by other properties. Atomic properties are the lowest level of properties without other properties. Atomic properties include property name and property value. In our ontology definition model, we use properties to express specific relations among classes and among individuals. For instance, to express that “Los Angeles” is in “USA”, we define a property “isIn” for “Los Angeles” and assign it with the value “USA”. Property is also referred to as “attribute”, “feature” or “characteristic” in ontology specification languages.
- Relation defines ways in which classes or individuals can be associated with each other. In our ontology definition model, the types of relations are predefined. Four types of relations are defined to connect classes and individuals: 1) Subclass extends an abstract class to convey more concrete knowledge; 2) PartOf means a class or an individual is a part of another class or individual. For example, class

"Tourist Attractions" is a part of the class "Location"; 3) Complement expresses that the instances of a class do not belong to another class; and the two classes together contain all the instances in a given domain; and 4) Equivalence means that two classes, individuals or properties are the same. For example, class "Nation" could have an equivalence relation with "Country". To express specific relations (*e.g.*, isIn) other than the four types of relations between classes or individuals, we use properties.

4 Searching for Matching Ontologies

There are few ontologies named using long phrases, such as "Plan a trip to Los Angeles" which is the context value for context type "activity". We use the following steps to find an annotated ontology for each context value.

1. We treat the context value as a searching string, and use the entire searching string to search for ontologies from ontology databases, such as Freebase [7]. Freebase is an ontology database which extracts structured information from Wikipedia [8]. If we can find a matching ontology, we annotate this ontology to this context value. Otherwise, we go to step 2.
2. We use an adjective and adverb dictionary to identify and remove the first adjective or adverb in the searching string. Adjectives and adverbs are constraints for the describing entity. Therefore, we can keep the important information in the searching string without the adjectives and adverbs. Meanwhile, if the phrase of a context value contains another context value, we remove the repeated words from the long phrase. Thus, in our example, we can remove "Los Angeles" from the context value "plan a trip to Los Angeles". If the removed word is followed by a stop word, we also remove the stop word. A stop word is a commonly used word (such as "by", "the", and "about") that does not contain important significance and some search engines have been set to ignore. Then we use the remainder part of the searching string to search for ontologies.
3. If we can find a matching ontology, we annotate this ontology to the context value. For example, if we cannot find ontology for the context value "luxurious travel" but an ontology of "travel" is available, we annotate the ontology "travel" to the context value "luxurious travel".
4. If we cannot find a matching ontology, repeat (2) and (3) until we find a matching ontology or the string is empty.

Finally, if we cannot find any relevant ontology using the context value, we use synonyms of the context value to search for ontologies and repeat above steps. In our research, we use WordNet [9] to identify the synonyms of the context value. WordNet is a lexical database which groups words into sets of synonyms and connects words to each other via semantic relations. After trying above steps, if we still cannot find any matching ontologies for a context value, we create an empty ontology and set the context value as the only entity of the new ontology.

5 Identifying Context Relations

Our approach uses the relations among context values to identify a user's requirements in the given context scenarios. We use two steps to identify the relations among multiple context values.

1. *Identifying the relations between two context values*. We compare the corresponding ontologies which represent the semantics of context values to identify the relations between two context values.
2. *Integrating all the relations of two context values*. To get the relations among multiple context values, we integrate the relations between two context values to construct a relation map that describes the relations of multiple context values.

5.1 Identifying Relations of two Context Values

5.1.1 Similarity of Entities in Ontologies

Ontologies may be defined by various people from different perspectives. The entities (*i.e.*, classes, individuals or properties) defined in two different ontologies may have different names for the same concept. Moreover, the entities of two ontologies can be defined in different granularities, even though both ontologies refer to the same thing. For example, "United States", "USA" and "America" are different names for a same entity. As shown in Figure 2, the class "Tourist Attraction(s)" defined in ontologies "Los Angeles" and "Travel" contains different levels of details although both classes of "Tourist Attraction(s)" refer to places of interest where tourists visit. To identify the same entities defined in different ontologies, we define the term *similarity*. We describe the *similarity* between two entities in ontologies as follows:

1. Two phrases (*e.g.*, entity names, property values) are *similar*, when the words are identical, synonyms or originated from the same stem. In this case, we use WordNet to identify synonyms and stems of words. For the example shown in Figure 4, phrases "Tourist Attractions" and "Tourist Attraction" are *similar* since both are stemmed from the phrase "Tourist Attraction".
2. E1 and E2 are atomic properties. E1 and E2 are *similar* if and only if the property names and property values of E1 and E2 are *similar*. For example, atomic properties "Price Range: budget" and "Price Range: cheap" are *similar* since both properties have the same property name "Price Range" and have *similar* properties values "budget" and "cheap".
3. E1 and E2 are classes, individuals or non-atomic properties. E1 and E2 are *similar* if and only if

   a. The names of E1 and E2 are *similar*; and

b. All the properties defined in entity E1 exist in entity E2, or all the properties defined in entity E2 exist in E1.

For example, class "Tourist Attractions" with properties "location: Los Angeles" and another class "Tourist Attraction" which does not have properties are *similar*, since the class name "Tourist Attractions" and "Tourist Attraction" are *similar*, and the properties defined in the latter class (*i.e.*, no properties) belong to the former class.

We use WordNet [9] to identify the synonyms and stem of words. WordNet is a lexical database which groups words into sets of synonyms and connects words to each other via semantic relations. By considering the synonyms and stems of words, we can discover that two entities are *similar* even if the entities are not de-scribed using the same words. In (3), E1 and E2 might have different numbers of properties. When describing the same entity, some ontologies may provide more detailed information than others due to the different levels of granularity in ontologies. If the properties of E1 (*i.e.*, class or individual) are a subset of the properties of E2, E1 and E2 are treated as *similar* entities.

5.1.2 User-Defined Relations Using Domain Knowledge

By comparing the similarity of entities, we can discover the semantic relations between context values. However, the similarity of entities cannot identify the relations which require domain knowledge. For example, in the travel scenario, Tom is a graduate student with low income. We can infer that he might prefer budget hotel instead of luxury hotel while he is traveling. From the ontology of graduate students, we may know that graduate students have low income, but the ontology of graduate students would not specify that he prefers budget hotels. To overcome this problem, we use LinQL language [10] to specify links between entities. LinQL is an extension of SQL and defines the conditions that two given entities must satisfy before a link of two entities can be established.

$$\begin{aligned} Linkspec\_stmt = {} & CREATE\ LINKSPEC\ linkspec\_name \\ & AS\ link\_method\ opt\_args\ opt\_limit; \end{aligned} \tag{1}$$

Eq. (1) shows the main structure of defining a link specification statement (linkspec for short) using LinQL. As shown in Eq. (1), a CREATE LINKSPEC statement defines a new linkspec which specifies the name of the linkspec and a method to establish the link. For example, Eq. (2) defines that if a person's income is low, then the person would prefer economical consumption style.

$$\begin{aligned} & CREATE\ LINKSPEC\ consumption\_style \\ & AS\ LINK\ \ low\_income\ WITH\ target \\ & WHERE\ synonym(term,\ economy) \\ & \quad AND \\ & \qquad target\ LIKE\ '\%term\%' \end{aligned} \tag{2}$$

The economical consumption style is defined as terms with a property of economy. The details of defining LinQL are described in the publication of Hassanzadeh *et al.* [10]. In our approach, the administrator of the context-aware system can use LinQL to provide the domain knowledge. Meanwhile, we could develop a graphic user interface to visualize LinQL and enable users to create some simple relations using their knowledge.

5.1.3 Relations between Two Context Values

Based on the definitions of similarity and user-defined relations, we identify the following 5 types of relations between two context values extended by ontologies:

1. Intersection: refers to the fact that the ontologies of two context values con-tain similar entities (*i.e.*, classes or individuals). Figure 4 shows three examples of intersection relations. In Figure 4, the context value "travel" (*i.e.*, its relevant context type is "activity") and context value "Los Angeles" share the same entity "Tourist Attraction". Context values, "Los Angeles" and "NBA", contain a common entity, "Los Angeles Lakers". When a context value is a part of another context value, such context values are in an intersection relation. In the travel example, ontology "Los Angeles" contains an entity "Hollywood". Therefore, "Hollywood" is a part of "Los Angeles". The context values "Hollywood" and "Los Angeles" have an intersection relation.
   We use entity names, properties and individuals to describe the common entities among two ontologies. The children entities (*e.g.*, sub-classes, and individuals of sub-classes) of the common entities are ignored if the children entities are not defined in one of the ontologies. This can make the description of common classes simple, since children entities contain too many details and could become noises of the common entities.
2. Complement: indicates that all members (*i.e.*, classes or individuals) defined in one ontology do not belong to another; and both context ontologies define all the elements in a given domain. The complement relations can be directly derived from the ontology definitions. For example, context values "Economy Hotel" and "Luxury Hotel" have a complement relation as defined in the ontology of "Travel".
3. Equivalence: defines that two context values describe the same concept. Equivalence relations should be explicitly defined in one of the ontologies. Explicit defined equal entities are treated as similar entities when we compare the entities from two different ontologies.
4. Domain Specific relation: means that the corresponding ontologies of two context values contain entities which are linked by user-defined relations. As shown in Figure 4 (3), the domain specific relation is identified by a user-defined relation which links the low income to budget items, *i.e.*, budget hotel in the travel scenario.
5. Independence: means that two context values do not have any connection.

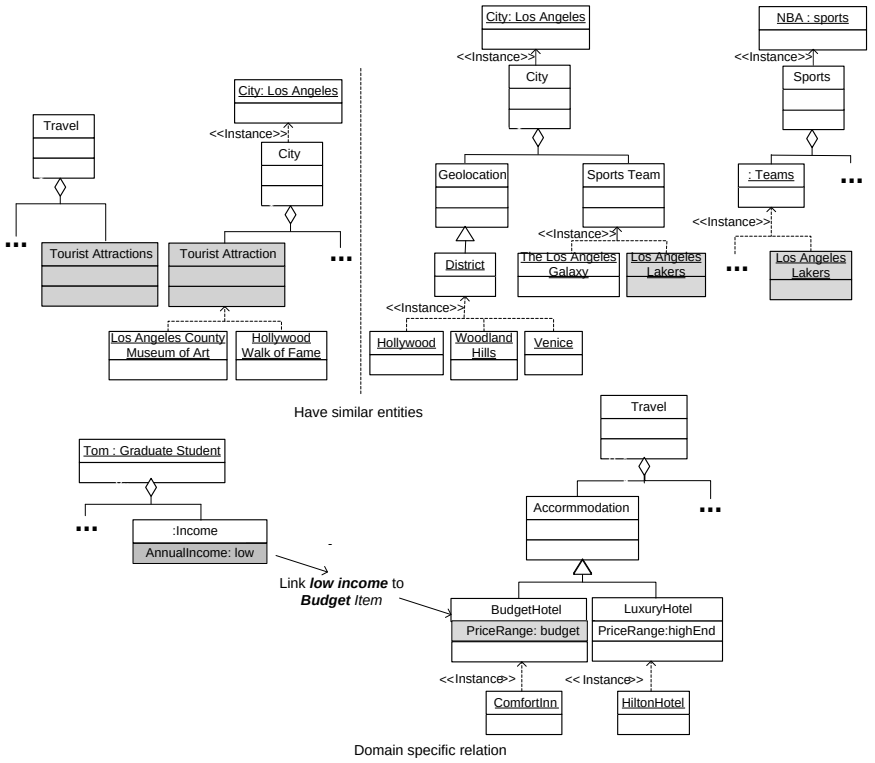

Fig. 4: Examples of relations between two context values

5.2 Inferring Relations among Multiple Context Values

We use entity-relationship (E-R) diagrams [11] to create a global view of relations among multiple context values. E-R diagrams provide a formal description for a set of entities and relationships among entities.

For each relation of two context values, we convert the two context values into two entities in the E-R diagrams. The relation type (*e.g.*, intersection and complement) is converted into a relationship node in the E-R diagrams. A relationship node connects its relevant entities. If the relation type is *intersection*, the common entities are converted into attributes of the *intersection* relationship in the E-R diagram. *Equivalence* relations are used to combine entities in the E-R diagram. To simplify an E-R diagram, *independence* relations are not explicitly described in an E-R diagram. If two entities are not connected by a relation node in the E-R diagram, it indicates that the entities are *independent*.

We integrate the relations of two context values into an integrated E-R diagram in the following steps:

1. Initialize the integrated E-R diagram as empty.
2. For each relation in the relation list, we repeat the following steps:
   a. Convert a relation of two context values into an E-R diagram.
   b. Add the E-R diagram created in step 2.a to the integrated E-R diagram. If there exist *similarity* or *equivalence* entities, we merge the *similarity* and *equivalence* entities by keeping the one with the richer information in the E-R diagram. If there exist *subset* or *complement* relations, we add a relationship node in the integrated E-R diagram to indicate the corresponding relation. If two relationship nodes contain the same relation type and relationship attributes, we merge them into one relationship node.

Following the aforementioned steps, all the context values are converted into entities in the integrated E-R diagram and the entities which are associated to relations of context values are transformed into properties in the E-R diagram. Figure 5 shows an example of an integrated E-R diagram for the context values in the travel scenario. In Figure 5, context ontologies "Student" and "Travel" have an domain specific relation due to a user-defined relation which links "Income: Low" to "Budget Hotel". The NBA team "Los Angeles Lakers" is shared by three context values "Los Angeles", "Kobe Bryant" and "NBA". Context ontology "travel" shares the same class "Tourist Attractions" with ontology "Los Angeles". Class "Tourist Attractions" contains a set of individuals such as "Hollywood Walk of Fame" and "Los Angeles County Museum of Art". We can use the individuals to recommend specific tourist attractions (*i.e.*, services) in Los Angeles.

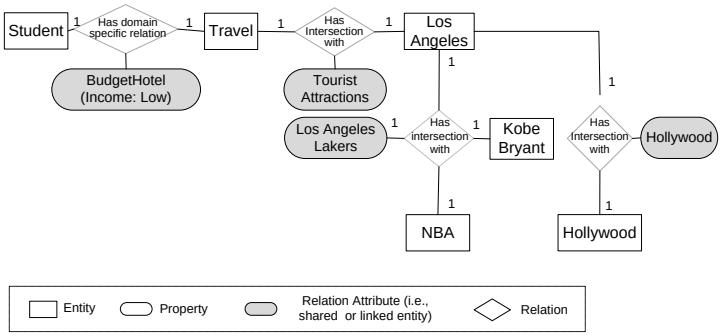

Fig. 5: An example integrated E-R diagram

6 Generating Service Searching Criteria

To recommend services, we need to identify user's requirements, and then generate searching criteria to search for services. A user's requirements describe the potential

tasks to perform in a given context scenario. We define generic rules to infer user's requirements from the E-R diagram. Then we extract service searching criteria from the description of user's requirements to search for services.

6.1 Identify User's Requirements in Given Context Scenarios

In our approach, the requirements of a user in a context scenario are identified based on the relations among different context values. However, some relations among context values generally exist in all the scenarios of a user due to the long-term existence of certain context values or the inherent relations of context types. For instance, in the travel scenario, Tom's preferences involve "NBA" and "Kobe Bryant". These preferences can be explicitly specified by Tom and generally exist for a long time. Our approach might always need to recommend the service of "Los Angeles Lakers" since the ontologies of "NBA" and "Kobe Bryant" share the same entity "Los Angeles Lakers". Another example is a case where, the current "city" (*e.g.*, Toronto) always belongs to the current "country" (*e.g.*, Canada). To avoid repeated recommendations, we ignore the relations among context values when the relations are derived from the context values that exist for a long time or inherently exist in the associated context types.

We design 3 generic rules to derive user's requirements from the integrated E-R diagram as shown in Table 1. Suppose E_{c1}, E_{c2}, ..., E_{cn} are entities in the integrated E-R diagram. Potential Task Set represents a set of a user's requirements.

Table 1: Generic rules to derive user's requirements

Rule No.	Relations	Potential Task Set	Description
1	Intersection relations: $E_{C1} \bigcap E_{C2} \ldots \bigcap E_{Cm} = \{e_1, e_2, , e_k\} \neq \emptyset$	$\{e_1, e_2, \ldots, e_k\}$	$E_{c1}, E_{c2}, \ldots, E_{cm}$ are entities in the integrated E-R diagram. $e_1, e_2, \ldots, e_k$ are entities or relationship attributes in the integrated E-R diagram.
2	Domain specific relations: E_{C1} is linked to E_{C2} by user-defined relations $\{(e_{11} \rightarrow e_{21}), \ldots, (e_{1k} \rightarrow e_{2k})\}$	$\{(e_{11} \rightarrow e_{21}), \ldots, (e_{1k} \rightarrow e_{2k})\}$	$E_{c1}, E_{c2}, \ldots, E_{cn}$ are entities in the integrated E-R diagram, and $(e_{1i} \rightarrow e_{2i})$ are a pair of linked entities between entity E_{c1} and entity E_{c2}. In the E-R diagram, $(e_{1i} \rightarrow e_{2i})$ represents a property of the user-defined relation.
3	Complement relations: $\bar{E_{ci}} = E_{cj}, e_1 \in E_{ci}, e_2 \in E_{cj}$ and $e_1, e_2 \in potentialTaskSet$	e_1 and e_2 have an OR relation.	E_{ci} and E_{cj} are entities in the integrated E-R diagram. e_1 and e_2 are entities or relationship attributes in the integrated E-R diagram; and $\bar{E}$ represents the complement of entity E.

Rule 1 collects the common entities and properties from the E-R diagram. The entities in the *Potential Task* Set are contained in two or more ontologies corresponding to the context values. Each entity in the *Potential Task* Set indicates a part

of a user's requirements. For example, the context value "Los Angeles" and context value "NBA" have a common entity "Los Angeles Lakers". The common entity "Los Angeles Lakers" is a NBA team in Los Angeles, and there is a high chance that the user would be interested in the services related to this team.

A user-defined relation connects two entities from two different ontologies. The linked entities are represented as pairs (*e.g.*, $e_{1i} \rightarrow e_{2i}$) in Table 1. If two entities from different context values are linked by a user-defined relation, it means these two entities are different from other entities in the ontologies of context values and the information in these two entities might be interesting for the end-users. Therefore, in rule 2, we extract the linked entities and add them to *Potential Task* Set. In the example of planning a trip, "Budget Hotel" has a high chance to be of interest to Tom since the entity "Budget Hotel" is linked by an attribute of the occupation of the user.

Complement relations show that two entities cannot co-exist at the same time. In Rule 3, we use complement relations to split the entities in *Potential Task* Set and identify them as "OR" relation. For example, if a *Potential Task* Set contains both the entities "Budget Hotel" and "Luxury Hotel", we can use the complement relations to identify them as a "OR" relations. Therefore, when an end-user choose one recommendation (*e.g.*, Budget Hotel), we stop to recommend the complement recommendation (*e.g.*, Luxury Hotel) since the user has made a decision between these two types of hotels.

Once the rules are applied on the E-R diagram, we obtain a *Potential Task* Set which contains a set of entities and properties of the entities in the E-R diagram. Some entities in the *Potential Task* Set may describe the same concept at different levels of details. For example, one entity can be a subclass of another entity. To reduce the redundancy of service recommendations, we classify the entities in a *Potential Task* Set t into different groups to merge similar user's needs. Each group maps to a specific service searching criterion.

6.2 Generate Service Searching Criteria

The entities and properties in a user's requirements (*e.g.*, *Potential Task* Set) are described using structured data defined in ontologies. We use the mapping rules specified in Table 2 to convert structured data to service searching criteria. A class name in Table 2 refers to the name of a class defined in an ontology. Furthermore, the generated searching criteria are submitted to existing search engines, such as Google [12].

In Table 2, the first column contains the entities from the extracted user's requirements (*i.e.*, *Potential Task* Set). The second column lists the associated query to find matching Web services described in WSDL. The third column shows the generated query submitted to a Web search engine. As shown in Table 2, a class contains a class name and prosperities. In a WSDL query, a class name is used to match a service name or operation name in a WSDL document since the class name is the major

object name involved in a Web service and it is generally used to describe service names and operation names in WSDL. In most cases, a service name and an operation name are not identical to the class name defined in ontologies. For example, an operation used to search for budget hotels can be named as "GetBudgetHotel" or "BookBudgetHotel". The operation names contain the class name "BudgetHotel" with additional verbs (*i.e.*, "Get" or "Book"). Therefore, in the generated WSDL query, instead of specifying that we need to find a WSDL having the service name or operation exactly matching with the class name, we check if a class name appears in the service name or the operation name. We apply the same requirements to other ontology entities in the conversion process.

Table 2: Mapping ontology entities in *potential task* set to WSDL query and general query

Entities in the *PotentialTaskSet*		WSDL Query	General Query for WebPages
Type	Involved data		
Class	Class name: $name_{class}$	$(name_{class} \subseteq (name_{operation} \bigcup name_{service}))$ &&$(name_{property_i} \subseteq (name_{inputPar} \bigcup name_{outputPar}))$ *where* $name_{property_i} \in (\bigcup name_{property})$	$involvedContextValue$ *AND* $name_{class}$ *AND* $(property_1$ *OR* $property_2$ $\ldots$ *OR* $property_k)$
	Property names of the class: $\bigcup name_{property}$		
Individual	Individual name: $name_{individual}$	$(name_{individual} \subseteq (name_{operation} \bigcup name_{service}))$ &&$(name_{property_i} \subseteq (name_{inputPar} \bigcup name_{outputPar}))$ *where* $name_{property_i} \in (\bigcup name_{property})$	$involvedContextValue$ *AND* $name_{individual}$ *AND* $(property_1$ *OR* $property_2$ $\ldots$ *OR*$property_k)$
	Property names of the individual: $\bigcup name_{property}$		
$User-defined$ $relation$ $(e_1 \rightarrow e_2)$	Name of the class that entity e_2 belongs to: $name_{class}$	$(name_{class} \subseteq (name_{operation} \bigcup name_{service}))$ &&$(name_{property_i} \subseteq (name_{inputPar} \bigcup name_{outputPar}))$	$involvedContextValue$ *AND* $name_{individual}$ *AND* $(property_1$ *OR* $property_2$ $\ldots$ *OR*$property_k)$
	Properties of entity e_2 : $\bigcup name_{property}$		

Properties of a class specify the detailed attributes of the class. There is a high chance that the properties of classes are required input for performing an operation or are the output data after executing an operation in WSDL services. For example, in our travel scenario, the property “price” in class ”Budget Hotel” becomes a parameter of the operation “BookBudgetHotel”. As listed in Table 2, the names of the properties are used to match parameters of operations in WSDL service. However, a service may not need to use all the properties defined in the class. Therefore, we use the OR relation to connect all the properties. The searching criteria for Web search engines are focused on keywords. In column 3, we convert the class name into a keyword and the properties of classes to the optional (*i.e.*, OR relations) keywords in the query. For the individuals in the *Potential Task* Set, we use the same way as classes of ontologies to convert them to two different queries.

When we specify user-defined relations, entities with more generic meanings are generally used to search for specific entities. For example, we use the generic entity “low income” to find all the budget (or economic, cheap) items. The entity with relevantly more specific meanings plays a more important role in identifying a potential task since the specific entity contains more concrete information. Therefore, we convert the specific entity instead of the general entity to search query as shown in the fourth row of Table 2.

7 Implementation

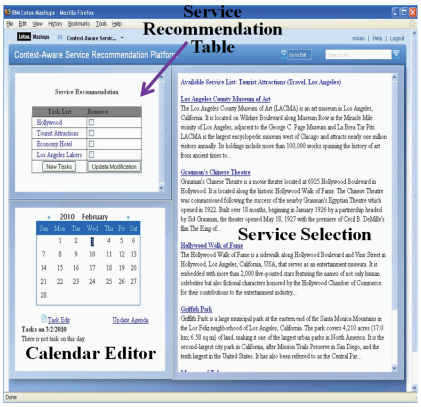

Fig. 6: An annotated screenshot for our service recommendation page

A prototype of the proposed approach was implemented. The prototype is developed in Java and uses OWL API [13] as the ontology/RDF parser. To evaluate the WSDL query generated by our approach, we implemented a component to support advanced search based on elements of WSDL (*i.e.*, service name, operation name and input/output parameters). Figure 6 shows an annotated screenshot of our service recommendation page. A list of services of potential interest to the user is provided in the services recommendation table. A service in the services recommendation table can be associated with one or more concrete services as shown in Figure 6. Once a user selects the "Tourist Attractions" in the service recommendation table, the associated services (*e.g.*, a list of tourist attractions in Los Angeles) are automatically displayed in the service selection panel on the right side of the Web page. The user can select the services the best fit its requirements.

We use Freebase [7] as the ontology database. In Freebase, there are common entities shared by most of the ontologies, such as "type.object.key", "namespace", and "common.topic". Those entities are used to organize the resources in the database, but are not useful for identifying user's requirements. To increase the accuracy of relation identification, we manually analyze the schema of ontologies defined in Freebase to identify and filter out those meaningless entities.

8 Case Study

The objective of our case study is to evaluate the effectiveness of our approach. In particular, we want to examine: 1) whether our approach can effectively recommend useful tasks represented as classes, individuals and properties in the set *Potential Task* Set; and 2) whether the generated searching criteria can find the desired services.

8.1 Setup

Table 3 lists the context types used in our case study. By providing different context values for each context type, we can create different user scenarios. Each scenario is composed of the context types listed in Table 3 with assigned context values. For each scenario, our approach automatically detects various potential tasks for the user and recommends different services. In our case study, we provide 5 different context values for each context type. Using different combinations of these context values, we generate 600 different context scenarios for our case study.

Due to the limitation of time and resources, we cannot evaluate all the 600 scenarios. In our case study, we randomly select 2% (*i.e.*, 12) context scenarios from the 600 context scenarios to evaluate our approach. To evaluate the identified potential tasks and the service searching criteria generated by our approach in different scenarios, we recruited 6 graduate students to participate in our case study. These

graduate students have many years of experiences using online services and possess basic knowledge on the context values that appeared in the context scenarios.

Table 3: Context types used in our case study

Context types	
Previous environment	Location (city and county)
Current environment	Location (city and country)
	Activity (described by keywords)
Future environment	Location (City and country)
	Activity (provided by calendar, described using keywords)
User's preferences and background	Favorite sports
	Favorite food
	Favorite celebrities
	Major
	Other preferences
	Income

8.2 Evaluation Criteria

Precision and *recall* are widely used in information retrieval. We use *precision* and *recall* to measure our approach. *Precision* and *recall* are defined as follows.

$$Precision = \frac{|\{relevant\ items\} \bigcap \{retrieved\ items\}|}{|\{retrieved\ items\}|}, \tag{3}$$

$$Recall = \frac{|\{relevant\ items\} \bigcap \{retrieved\ items\}|}{|\{relevant\ items\}|} \tag{4}$$

Precision and *recall* are defined in terms of a set of retrieved items (*e.g.*, the set of potential tasks found by our prototype for a given context scenario) and a set of relevant items (*e.g.*, the set of potential tasks existing in the context scenario). *Precision* is the ratio of the number of returned relevant items to the total number of returned items of a query. *Recall* is the ratio of the number of returned relevant items to the total number of existing relevant items.

8.3 Experiment Procedure

To evaluate the potential tasks identified by our approach, we assign 2 context scenarios to each subject described in the previous section. For each scenario, a subject manually examines the context values and uses her knowledge to identify the potential tasks that she would like to perform. Independent from the manual evaluation,

we also use our prototype to automatically identify the potential tasks by analyzing the context values and the relations among context values. We compare the task sets produced by the subjects and the ones generated by our prototype tool to calculate the precision and recall of each scenario.

To evaluate the service searching criteria generated by our approach, we use the approach described in Section 6 to generate the service searching criteria, then we submit the searching criteria to search engines Google [12] and Seekda [14] to search for online services. Seekda is a search engine to search for Web services described using WSDL. One of the authors manually examined the available services in Seekda for each scenario. If there are available Web services in Seekda for a given topic, we use Seekda. Otherwise, we use Google to search for services. We use the keywords in the generated searching criteria to search for services. In both cases, we use the generated WSDL query to check the top 20 returned services to identify the matching services. For each query, our prototype chooses the top two returned services to recommend to the subject. The 6 aforementioned subjects manually provided the description of desired services based on the given context scenarios. One of the authors manually compared the services recommended by our prototype with the desired services described by the subjects to evaluate if our prototype can correctly recommend services to a subject for a given context scenario.

8.4 Result Analysis

In the 12 context scenarios, 2 scenarios do not have any tasks recommended according to the results from the subjects as well as the results of our prototype. We manually examined both scenarios. We found that the context values in both scenarios do not have any relations. Table 4 shows the results for detecting potential tasks from the remainder 10 scenarios. We notice that some tasks in certain scenarios are not included in the result from the subjects due to the limitation of subject's knowledge. However, such tasks are identified by our prototype. For example, in a travel scenario, "Michael Jordan" is a favorite celebrity of a subject, and one of the context values is the city "New York". Our prototype can identify that "New York" is the birth place of "Michael Jordan". As a fan of Michael Jordan, the subject would be interested to know this information and purchase the related souvenirs using an on line shopping service. However, such information is overlooked by the subject. When calculating recall and precision, we add the missed tasks into the relevant items set and treat the missed tasks as desired potential tasks. The 94% of recall reveals that our approach can identify most of the potential tasks based on the semantics of context values. Moreover, our prototype can identify the tasks that are overlooked by the subjects.

Table 5 lists the evaluation results of service recommendation. The results show that our approach can recommend most of the needed services desired by subjects. However, as listed in Table 5, the recall and precision are not very high in some

Table 4: Recall and precision for detecting potential tasks

Scenarios	# of Retrieved tasks	# of Retrieved relevant tasks	# of relevant tasks	Recall	Precision
1	2	2	2	100%	100%
2	1	1	1	100%	100%
3	3	2	3	67%	67%
4	3	3	4	75%	100%
5	2	2	2	100%	100%
6	3	1	1	100%	33%
7	3	3	3	100%	100%
8	3	2	2	100%	67%
9	4	4	4	100%	100%
10	1	1	1	100%	100%
Average				94%	87%

context scenarios. Here are some reasons which we plan to address in our future work:

1. Some ontologies do not describe all the aspects of a context value. The incomplete ontologies cause incomplete service recommendation. Meanwhile, we only define one user-defined relation which is Eq. 2 to capture the do-main knowledge of "Income: low". If we add more domain knowledge using user-defined relations, it could increase the recall and precision. For example, one subject in our case study lists "Tickets for Museums at Miami" as a potential task for a context scenario which specifies that the subject majors in "Art" and will attend a conference in "Miami". Due to the lack of domain knowledge of "Art", it is difficult for our prototype to automatically establish the relations between "Art" and "Museums".
2. Although WordNet can provide stems and synonyms for a single word, it cannot give the synonyms of phrases (*i.e.*, two or more words in sequences to represent a specific meaning) which are the most common expressions of entities in ontologies. The lack of phrases in our semantic analysis database (*i.e.*, WordNet) makes it challenging for our prototype to identify the similarity of phrases defined in ontologies.
3. When the number of keywords increases, the results returned by Google or Seekda are likely to diminish. Especially, we may extract general terms from ontologies, such as "people", "person", and "location". Such terms in the searching keywords often result in drastically reduction of the quality of searching results.

Table 5: Evaluation Results of service recommendation

Scenarios	Total # of retrieved services	Total # of retrieved relevant services	Total # of relevant services	Recall	Precision
1	4	4	4	100%	100%
2	2	2	2	100%	100%
3	6	4	6	67%	67%
4	6	6	8	75%	100%
5	4	3	4	75%	75%
6	6	2	2	100%	33%
7	6	5	6	83%	83%
8	6	4	4	100%	67%
9	8	8	8	100%	100%
10	2	2	2	100%	100%
Avarage				90%	83%

8.5 Threats to Validity

Construct validity is the degree to which the independent and dependent variables accurately measure the concepts which they are intended to measure. We have carefully designed our case study to avoid the threats of construct validity. To evaluate the effectiveness of identified context relations and recommended services, we use *recall* and *precision* which are well adopted evaluation criteria in literature. However, the potential tasks and relevant services of context scenarios contain subjective issues. For example, one subject may be satisfied by a recommended task while another user may not like the recommended task at all. In our case study, we ask the 6 subjects to provide the potential tasks and evaluate the returned services according to the relations among context values and their understanding of the context scenario. The identified potential tasks and relevant services recommended by the 6 objects may not reflect the potential tasks of all the users in practices. Especially, in our case study, all the 6 subjects are graduate students. In the future, we plan to hire more subjects with different backgrounds to participate in our case study.

External validity refers to the generalization of the results. In our case study, we automatically generated 600 different context scenarios and randomly selected 12 scenarios out of the 600 context scenarios. We believe that the automated generation and random selection of context scenarios can reflect the practical situations. However, there are various context types and many variations of context values in a context-aware system. Our case study only evaluates a limited number of context types and values. In the future, we plan to expand our context scenarios with more context types and values. When the number of context types and values increases in our case study, we expect that the *precision* and *recall* is likely to be lower than the result of our current experiment.

Internal validity is concerned with the cause-effect relationship between independent and dependent variables. In our case study, the retrieved tasks are automatically identified by our prototype, and the relevant potential tasks are identified by subjects who did not observe the results of our prototype. Therefore, we can rule out a learning effect of subjects that may impact the results of our case study.

9 Related Work

9.1 Context Modeling and Context-aware Systems

Several context models and context-aware systems are proposed in the literature [2, 15, 16, 17, 4]. Strang and Linnhoff-Popien [18] survey existing context models and classify them into different types based on the data structures. The context models are classified into 6 types: key-value models, markup scheme models, graphical models, object oriented models, logic based models, and ontology based models. The context models are evaluated using six requirements. Ontologies are the most expressive model that can fulfill most of the requirements. Sakurai *et al.* [17] propose a methodology to interpret and combine sensor outputs with contexts as sets of annotated business rules. Chen and Kotz [15] investigate the research on context-aware mobile computing. Chen and Kotz discuss the types of context used, the ways of using context, the system level support on collecting context, and approaches to adapt to the changing context. Baldauf *et al.* [2] present a layered conceptual design framework to describe the common architecture principles of context-aware systems. Based on their proposed design framework, Baldauf *et al.* compare different context-aware systems on various issues: the context sensing, context models, context processing, resource discovery, historical context data, security and privacy. In the aforementioned approaches, the context models are predefined and are not flexible to address the dynamical changing environment. In our approach, we can generate and adjust the context relation model automatically according to different available context values.

9.2 Discovering and Recommending Services using Context

Applying context-aware techniques to discover and recommend services has gained lots of attentions. Yang *et al.* [19, 20] design an event-driven rule based system to recommend services according to people's context. Yang et al. define an ontology-based context model to represent a context. Requester ontology and service ontology are developed for specifying the context of requesters and services respectively. Using rules, further contextual information can be inferred from the current contextual information. For example, a user's activity at a given time can be derived

by examining the time and calendar. When searching for Web services, Yang et al. identify the similarities of inputs/outputs between requests and published services using capability matching. If there are no matched services, a semantic matching component would decompose the request into sub-requests based on requester's contextual information and search for services for each sub-request. Balke and Wagner [21] propose an algorithm to select a Web service based on user's preferences. The algorithm starts with a general query. If there are too many results, it expands the service query using user's preferences. The algorithm expands the query with loose constraints extracted from user's preferences. If there are too many results, it extends the query with restricted constraints and searches for Web services again. By adding constraints step by step, the algorithm narrows down the number of service searching results to a small value. However, aforementioned approaches need to predefine the specific reactions on context scenarios using rules which are hard to provide in practice due to the diversity of context types and values in the real world. Our approach can automatically recommend services based on the semantics of context scenarios without requiring the designer of context-aware systems to provide specific rules.

Chen *et al.* [22] use a collaborative filtering technique to recommend services based on the Quality of services. Qi *et al.* [23] combine UDDI and OWL-S to describe semantic Web services. In OWL-S, class "process: local" allows users to define some local parameters. Qi *et al.* use "process: local" to describe context information. Qi *et al.* define 6 types of contexts: load of server, performance of server, response time of service, geographical position of client, geographical position of server, and distance between client end and server. Dynamic context can be updated on time. After finding services using semantic matching, Qi *et al.* use context data to evaluate the quality of services and rank the matching services. Mostefaoui *et al.* [24] present a CB-SeC (Context-Based Service Composition) service description model. In the CB-SeC service description model, Mostefaoui *et al.* define an optional part called the context function. The context function represents the context of the service (*e.g.*, the current workload of the service) and is shipped with other service description. The context function is used to select the best services from the matching Web service list if there is more than one matching Web service. The value of context function is not known in advance. It needs to be calculated during run time when it is needed. Different from Chen, Qi and Mostefaoui's approaches which use contexts to select services with high Quality of Service (QoS), our approach is intended to detect the requirements of users and recommend services with desired functions.

Abbar *et al.* [25] provide an approach to recommend services using the logs of a user and the current context of the user. To select and recommend services, the proposed approach requires historical data which are usually not available in the practice. Our approach only needs the context types and values to recommend services. Blake *et al.* [26] use an agent to detect the execution of applications and the behavior of human users, such as browsing the Internet. Then the agent extracts the context data from applications and users' behaviors. Based on the contextual data, the agent generates a query to search for available Web services. The agents

recommend services by matching the similarity of input/output and the operation name of Web services with the contextual information extracted by the agent. The approach by Blake *et al.* only analyzes the data that the user is currently processing. Their approach cannot combine and analyze two or more context values to recommend services. Our approach can analyze the relations of multiple context types and values and recommend services based on such relations.

10 Conclusion and Future Work

In this chapter, we present an approach to dynamically derive context relations from ontologies and automatically recommend services based on specific context values. By discovering the semantic relations among context values, our approach can identify user's tasks hidden behind the context values and generate searching criteria for service discovery. The case study shows that our approach can identify the context relations and user's potential tasks in different context scenarios with high precision and recall.

Context types and context values are interpreted from the outputs of sensors. For example, a GPS signal is mapped to abstract location such as at home or at work. Our current approach is based on the context types and context values which are provided by third part. In our next step, we plan to extend our approach to use or directly interpret the data from the outputs of different sensors. Meanwhile, we observe that some ontologies in FreeBase are not very suitable for extending the context values. As a result, it reduces the accuracy of service recommendation in our approach. To enhance our approach, we could try to use the ontologies from different ontology databases, such as DBpedia [27] and Swoogle [28]. There may have several matching ontologies for the same context value. Currently, there are no effective criteria to help us select the appropriate ontologies for the purpose of extending the context values. A further study can be conducted to evaluate the effectiveness of different criteria for ontology selection and identify the effective criteria for our work.

Acknowledgment

This work is financially supported by NSERC and the IBM Toronto Centre for Advanced Studies (CAS). We would like to thank Mr. Alex Lau, Ms. Joanna Ng and Mr. Leho Nigul at IBM Canada Toronto Laboratory and Dr. Foutse Khomh at Queen's University for their suggestions on this work. IBM and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. Other company, product, and service names may be trademarks or service marks of others.

References

1. P. Brézillon, IEEE Intelligent Systems **18**(3), 62 (2003). DOI 10.1109/MIS.2003.1200731. URL http://dx.doi.org/10.1109/MIS.2003.1200731
2. M. Baldauf, S. Dustdar, F. Rosenberg, International Journal of Ad Hoc and Ubiquitous Computing, Volume 2, Issue 4 (2007)
3. H. Xiao, Y. Zou, J. Ng, L. Nigul, in *Web Services (ICWS), 2010 IEEE International Conference on* (2010), pp. 163 – 170. DOI 10.1109/ICWS.2010.95
4. M.K. Smith, C. Welty, D.L. McGuinness. (editors) (2004), owl web ontology language guide, w3c recommendation. http://www.w3.org/TR/owl-guide/ (2012)
5. G. Klyne, J.J. Carroll. Resrouce description framework(rdf): Concepts and abstract syntax, w3c recommendation (2004)
6. D. Connolly, F. Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein. Daml+oil (march 2001) reference description, w3c note 18 december 2001. http://www.w3.org/TR/daml+oil-reference (2011)
7. Freebase. http://www.freebase.com/ (2012)
8. Wikipedia. http://en.wikipedia.org/wiki/Wikipedia:About (2012)
9. Princeton university, wordnet, 2010. http://wordnet.princeton.edu (2012)
10. O. Hassanzadeh, A. Kementsietsidis, L. Lim, R.J. Miller, M. Wang, in *Proceedings of the 18th ACM conference on Information and knowledge management* (ACM, New York, NY, USA, 2009), CIKM '09, pp. 1027–1036. DOI 10.1145/1645953.1646084. URL http://doi.acm.org/10.1145/1645953.1646084
11. P.P.S. Chen, ACM Trans. Database Syst. **1**(1), 9 (1976). DOI 10.1145/320434.320440. URL http://doi.acm.org/10.1145/320434.320440
12. Google. http://www.google.com (2012)
13. Owl api. http://owlapi.sourceforge.net/ (2012)
14. Seekda. http://webservices.seekda.com/ (2012)
15. G. Chen, D. Kotz, A survey of context-aware mobile computing research. Tech. rep., Hanover, NH, USA (2000)
16. C. Hesselman, A. Tokmakoff, P. Pawar, S. Iacob, in *Proceedings of the 1st European Conference on Smart Sensing and Context (EuroSCC'06)* (2006)
17. Y. Sakurai, K. Takada, M. Anisetti, V. Bellandi, P. Ceravolo, E. Damiani, S. Tsuruta, Sensors **12**(1), 632 (2012). DOI 10.3390/s120100632. URL http://www.mdpi.com/1424-8220/12/1/632
18. T. Strang, C. Linnhoff-Popien, in *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England* (2004)
19. S.J.H. Yang, J. Zhang, I.Y.L. Chen, Expert Syst. Appl. **34**(4), 2254 (2008). DOI 10.1016/j.eswa.2007.03.008. URL http://dx.doi.org/10.1016/j.eswa.2007.03.008
20. I. Chen, S. Yang, Z. Jia, in *Services Computing, 2006. SCC '06. IEEE International Conference on* (2006), pp. 60 – 68. DOI 10.1109/SCC.2006.110
21. W.T. Balke, M. Wagner, in *WWW (Alternate Paper Tracks)* (2003). URL http://dblp.uni-trier.de/db/conf/www/www2003at.html#BalkeW03
22. C. Xi, L. Xudong, H. Zicheng, S. Hailong, in *Web Services (ICWS), 2010 IEEE International Conference on* (2010), pp. 9 – 16. DOI 10.1109/ICWS.2010.27
23. Y. Qi, S. Qi, P. Zhu, L. Shen, in *Semantics, Knowledge and Grid, Third International Conference on* (2007), pp. 499 – 502. DOI 10.1109/SKG.2007.127
24. S. Mostefaoui, B. Hirsbrunner, in *Pervasive Services, 2004. ICPS 2004. IEEE/ACS International Conference on* (2004), pp. 71 – 80. DOI 10.1109/PERSER.2004.13
25. S. Abbar, M. Bouzeghoub, S. Lopez, in *Proc. of the International Conference on Very Large Data Bases (VLDB) Proflie Management and Context Awareness (PersDB) Workshop* (Lyon, France, 2009)
26. M.B. Blake, D.R. Kahan, M.F. Nowlan, Distrib. Parallel Databases **21**(1), 39 (2007). DOI 10.1007/s10619-006-7001-9. URL http://dx.doi.org/10.1007/s10619-006-7001-9

27. Dbpedia. http://dbpedia.org/ (2012)
28. Swoogle. http://swoogle.umbc.edu/ (2012)