An Approach for Context-aware Service Discovery and Recommendation

Hua Xiao
School of Computing
Queen's University
Kingston, Ontario, Canada
huaxiao@cs.queensu.ca

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

Joanna Ng, Leho Nigul
IBM Toronto Lab
Markham, Ontario, Canada
{jwng, lnigul}@ca.ibm.com

***Abstract*— Given the large amount of existing services and the diversified needs nowadays, it is time-consuming for end-users to find appropriate services. To help end-users obtain their desired services, context-aware systems provide a promising way to automatically search and recommend services using a user's context. However, existing context-aware techniques have limited support for dynamic adaption to newly added context types (e.g., location, time and activity). Due to the diversity of user's environment, the available context types may change over time. It is challenging to anticipate a complete set of context types while we design a context aware system. In this paper, we propose a context modeling approach which can dynamically handle various context types and values. More specifically, we use ontologies to enhance the meaning of a user's context values and automatically indentify the relations among different context values. Based on the relations among context values, we capture the potential services which the user might need. A case study is conducted to evaluate the effectiveness of our approach. The results show that our approach can use contexts to find users' needs and recommend their desired services with high precision and recall.**

Keywords-Context modeling; service recommenddation; service discovery

I. INTRODUCTION

With the growing prevalence of Service Oriented Architecture (SOA), more services become available for end users to use in their daily online experience. Due to the large number of available services, it becomes time consuming for end users to find appropriate services to satisfy their various needs. To help end-users obtain their desired services, context-aware systems provide a promising way to automate service discovery and recommendation. Specifically, a context characterizes the situation of a person, place or the interactions between humans, applications and the environment [6]. A context can be further described as a set of 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, the context types for a user include location, identity, and time. "New York" is a context value for 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 the context-aware system, a context model is used to specify the relations and the storage structure of various context types and values.

A context-aware system is 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 by providing desired services for a user. Most context-aware systems require the designer of context-aware systems to manually define all the context types. Moreover, the designer needs to manually establish the relation between the sensed context scenario and the corresponding services in the form of if-then rules which specify how a system should respond to context changes. However, the context types and values may vary considerably between users. It is challenging to anticipate a complete set of context types and values to satisfy all possible users. To provide a desired service to meet a user's context, fixed rules are not flexible enough to accommodate the changing environment and various personal interests.

To recommend services for a context scenario, we propose an approach that captures dynamic changing context scenarios and formulate searching criteria to discover the desired services. Different from existing approaches which depend on context models to know the relations among context types and values, and then use predefined rules to infer user's needs, we seek an automatic approach to recognize the relations between context values and a user's needs. 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 a limited budget. We expect that such relations can be used to express more accurate searching criteria which better reflect a user's context. When a new context type or a new value for a user is detected, our approach can automatically compute the relations between the new context type (or value) with other context types (or values). Instead of manually defining if-then rules using specific context types or values as the traditional context-aware systems [4], our approach uses the relations among context values to infer user's needs. Then we generate service searching criteria based on user's needs to discover and recommend services.

To facilitate the presentation of the paper, let us consider a travel scenario as an illustrative example throughout this paper. Tom is a graduate student living in Toronto. Tom is interested in watching Hollywood movies and National Basketball Association (NBA) games. 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 is useful to help Tom plan his trip. For example, as a graduate student who has low income, Tom might prefer budget hotel for the trip. As a fan

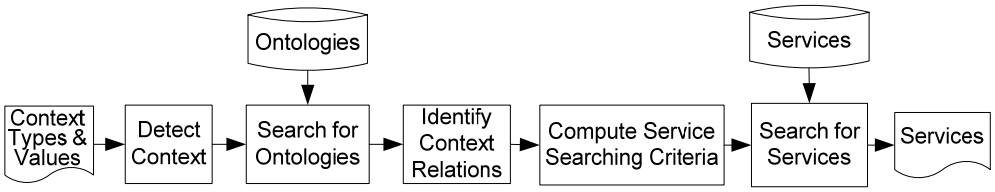

Figure 1. Steps for context-aware service recommendation

of NBA, Tom might know that the NBA team "Los Angeles Lakers" is in Los Angeles. In the practice, a user may have several preferences and various background information. She/he may not realize that one of the preferences is connected to a specific activity.

The remainder of this paper is organized as follows. Section II presents an overview of our approach. Section III introduces the background of ontologies. Section IV discusses the details of inferring relations among different context values. Section V presents our approach that identifies user's needs in a given context scenario and computes searching criteria to search for services. Section VI gives an overview of our prototype. Section VII discusses the case studies. Section VIII presents the related work. Finally, Section IX concludes the paper and presents the future work.

II. Overview of our Approach

Figure 1 gives an overview of our approach. Context types 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 type or a new value for a user is detected, our approach can automatically search for ontologies that expand the semantic meanings of the new context type or the new value and compute the relations with other context types and values.

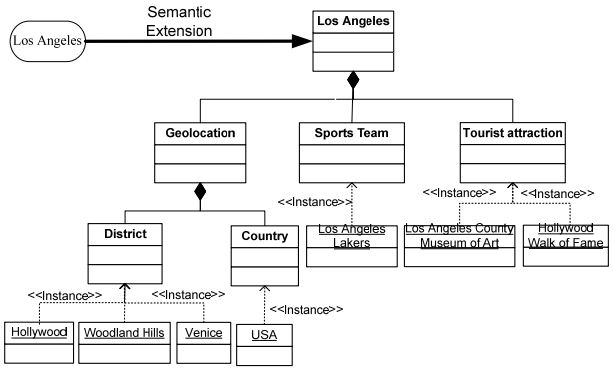

Figure 2. An example of extending context value using ontology

We use the identified context relations to discover a user's needs for a given context scenario and generate the corresponding service searching criteria. When the semantics (i.e., ontologies) of several context values share the same concepts, the common concepts would reflect better the user's interest. For example, if Tom is going to travel to "Los Angeles", and he is interested in watching NBA games, the ontologies of "Los Angeles" and "NBA" have the 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 services and recommend them to the user.

III. Modeling an Ontology Definition

Ontologies can be formally described using ontology specification languages, such as Web Ontology Language (OWL) [27], Resource Description Framework (RDF) [20], F-Logic [2] and DAML+OIL [10]. We use the ontologies to understand the meanings of a context type and a value. However, the ontologies found for context types and values can be described in different specification languages. To ease the inference of the relations among context types, we create a generic ontology definition model that captures the common structures and concepts of the ontologies defined in various languages. Figure 3 illustrates the major entities of the generic ontology definition model. Essentially, an ontology contains entities, such as classes, individuals, properties and relations.

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" is a class which has the common characteristics of tourist attractions. Class is also called "concept", "type", "category" and "kind" in some ontology specification languages.

Individual is the 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*. *Properties* which do not include other pro*perties* are called atomic properties. Atomic properties have two parts: 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 instance

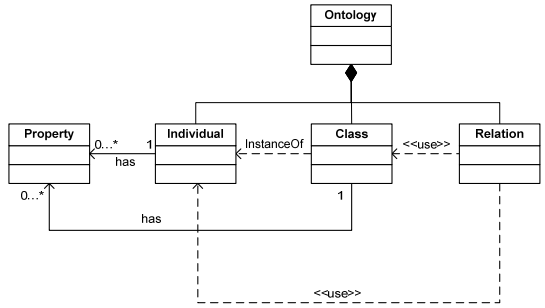

Figure 3. Scheme for our Generic Ontology Definition Model

"A" is the mother of instance "B", we define a property "MotherOf" for A and assign it to the value "B". *Property* is also referred to as "aspect", "attribute", "feature" or "characteristic".

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. To express specific relations (e.g., MotherOf), we need to use properties. 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 Attraction" is a part of the class "Los Angeles"; 3) *Complement*: expresses that the members of a class do not belong to another class; and the two classes together contain all the members in a given domain; and 4) *Equivalence*: means that two classes, individuals or properties are exactly the same. For example, class "Nation" could have an equivalence relation with "Country".

IV. IDENTIFING CONTEXT RELATIONS

The relations among context values can help us identify a user's needs in the given context scenario. In the following subsections, we discuss our approach to identify the relations of two context values. Then we present our approach to combine all the relations between two context values to construct a relation map that describes the relations among multiple context values.

A. Identifying Relations of two Context Values

To compare two ontologies, we need to compare the entities defined in two ontologies. However, ontologies may be defined by various people in different perspectives. The entities (i.e., classes, individuals or properties) defined in two different ontologies are not exactly the same even they refer to the same thing. For example, in Figure 4, the class "Tourist Attraction" defined in ontologies "Los Angeles" and "Travel" contains different levels of details although they all refer to a place of interest where tourists visit. To identify same entities defined in different ontologies, we define the term *similarity* between two entities E1 and E2 as follows.

1) When E1 and E2 are atomic properties, E1 and E2 are similar if and only if they have the same property name and property value;
2) When E1 and E2 are classes, individuals or non-atomic properties, E1 and E2 are similar if and only if
   - E1 and E2 have the same name; and
   - Any property defined in E1 can have a *similarity* property in E2, or any property defined in E2 exists a *similarity* property in E1.

As specified in (2), E1 and E2 might have different number of properties. When describing the same concept, some ontologies may provide more detailed information than others due to the different levels of granularity in ontologies. If the properties of one entity (i.e., class or individual) in one ontology are the subset of the properties of another entity in the other ontology, we treat them as *similarity*. Furthermore, we identify 4 types of relations between two context values based on *similarity* among entities:

I) *Intersection*: refers to the ontologies of two context values which contain *similarity* classes, individuals, or properties. 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 class "Tourist Attraction". Context value "Los Angeles" and "NBA" contains the common class "Los Angeles Lakers". Moreover, context value "Graduate Student" and "Travel" have the same property "priceRange: economy". When a context value is a part of another context value, such context values are also in an intersection relation. In the travel example, ontology "Los Angeles" contains an individual "Hollywood". Therefore, "Hollywood" is a part of "Los Angeles". The context values "Hollywood" and "Los Angeles" have an interaction relation.

To maintain the scope of the common properties of two ontologies, we annotate each property with the *class* or *individual* which directly defines the property. For example, the *property* "PriceRange:Economy" defined in Figure 4 can only be meaningful when it is asscoated with the relevant class "BudgetHotel" or "Consumption Style".

We use class names, properties and individuals to describe the common classes among two ontologies. The children entities (e.g., sub-classes, indirect prosperities, and individuals of sub-classes) of the common entities are ignored if the children entities are not shared by the two ontologies. This can make the description of common classes simple, since children entities contains too much details and could become noises of the common entities.

II) *Complement*: indicates that all members (classes or individuals) defined in one ontology do not belong to another, and both context ontologies define all the elements in the given domain. The *complement* relations can directly derived from the ontology definitions since *complement* is a relation defined in an ontology definition model. For example, context values "Economy Hotel" and "Luxury Hotel" have a complement relation as defined the ontology of "Travel".

III) *Equivalence:* defines that two context values describe the same concept. Equivalence relations should

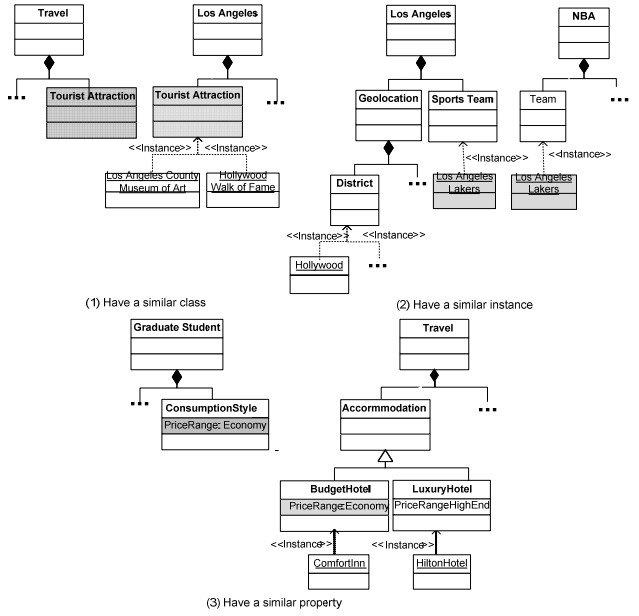

Figure 4. Examples of intersection relations

explicitly define in one of the ontologies. As specified the definition of *similarity* between entities, two entities in ontologies must have the same name. But two context values do not have the same name when they are defined as equivalence.

IV) *Independence*: means two context ontologies do not have any connection.

B. Inferring Relations among Multiple Context Values

To provide a global view of relations among two or more context values, we use entity-relationship (E-R) models [8] to describe the relations among multiple context values. E-R models provide a formal description for a set of entities and relationships among the entities.

For each relation of two context values, we convert the two context values into two entities in E-R model. The relation type (e.g., intersection and complement) is converted into a relationship node in the E-R model. A relationship node connects its relevant entities together. If the relation type is *intersection*, the common entities are converted into attributes of the *intersection* relationship in the E-R model. *Equivalence* relations are used to combine entities in the E-R model. To simplify an E-R model, *independence* relations

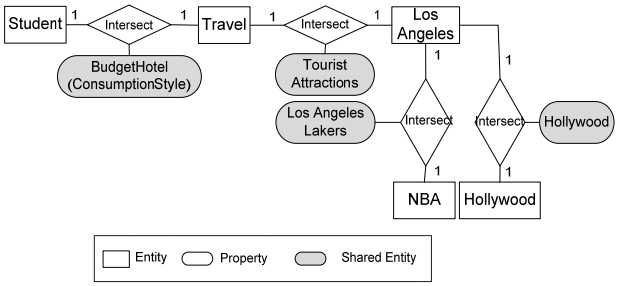

Figure 5. An example Integrated E-R model

are not explicitly described in an E-R model. If two entities do not connected by a relation node in the E-R model, it indicates that the entities are *independent*.

We combine relations of two ontologies into an integrated E-R model in the following steps:

1) Filter out *independence* relations since they are not useful for service recommendation.
2) Remove *equivalence* relations from the context value relation list.
3) Set the integrated E-R model as empty.
4) For each relation in the remainder relation list, we repeat the following steps:
   — Convert the relation into an independent E-R model.
   — Add the independent E-R model into the integrated E-R model. 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 model. If there exist *subset* or *complement* relations, we add a relationship node in the integrated E-R model 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 an E-R model and the common entities in the intersection relations are transformed into properties in the E-R model. Figure 5 shows an example of an integrated E-R model for the context values in the travel example. In Figure 5, context ontologies "Student" and "Travel" have an intersection relation due to the common property of "PriceRange: Economy". 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 those individuals to recommend specific tourist attractions (i.e., services) in Los Angeles.

V. GENERATING SEARCHING CRITERIA

To recommend services, we need to identify user's needs then generate searching criteria to search for services. A user's needs describe the potential tasks to perform in a given context scenario. We define generic rules to infer a user's needs from the E-R model. Then we extract service searching criteria (e.g., keywords) from the description of user's needs to search for services.

In our approach, a user's needs in his/her current context scenario and environments are identified based on the common entities among different context values. However, certain relations among context values generally exist among any entities. However, such general relations are not useful for service recommendation. For instance, the current "city" (e.g., Toronto) always belongs to the current "country" (e.g., Canada). To find the actual needs in a given context scenario, we ignore the general relations between two context values when the context values of the same context type are detected in the same time.

We describe the following rules to derive user's needs from the integrated E-R model. Suppose E_{c1}, E_{c2}, …, E_{cn} are entities in the integrated E-R model. *SharedElementsSet* represents the set of a user's needs.

Rule 1: Derive needs from intersection relations:

If $E_{Ci1} \cap E_{Ci2} \cap \cdots \cap E_{Cim} = \{e_1, e_2, \ldots, e_k\} \neq \emptyset$, ***Then***

SharedElementsSet = {e_1, e_2, …, e_k}

Where E_{c1}, E_{c2}, …, E_{cn} are entities in the integrated E-R model, and e_1, e_2, …, e_k are entities or relationship attributes in the integrated E-R model.

Rule 2: Extract relations of user's needs from Complement relations*:*

If $\overline{E_{ci}} = E_{cj}, e1 \in E_{ci}, e2 \in E_{cj}$ *and* $e1, e2 \in SharedElementsSet$,

Then

e1 and e2 have an OR relation.

Where E_{ci} and E_{cj} are entities in the integrated E-R model; e_1 and e_2 are entities or relationship attributes in the integrated E-R model; and $\bar{E}$ represents the complement of E.

Rule1 collects the common entities and properties from the E-R model. The entities in the *SharedElementsSet* are contained in two or more ontolgoies corresponding to the context values. Each entity in the *SharedElementsSet* represents a part of a user's needs. Complement relations show that two entities cannot co-exist at the same time. In Rule 2, we use complement relations to split the entities in *SharedElementsSet* and identify them as "OR" relation.

Once the rules are applied on the E-R model, we obtain a *SharedElementsSet* which contains a set of entities and properties of the entities in the E-R model. Some entities in the *SharedElementsSet* may describe the same concept at different levels of details. For example, one entity is the subclass of another entity. To reduce the redundancy of service recommendation, we classify the entities in *SharedElementsSet* into different groups to merge similar user's needs. Each group maps to a specific service searching criteria and eventually maps to a service. We use the following steps to group entities in *SharedElementsSet*:

1) Each entity in *SharedElementsSet* is treated as a group.
2) In the integrated E-R model, if the entities in one group are a subset of the entities in another group, we combine these two groups together.
3) Repeat step 2 until no groups can be combined.

Figure 6. An annotated screenshot for our service recommendation page

The entities in the above groups are described using structured data. However, existing service searching engines only support search by keywords. Therefore, we extract keywords from each group as the searching criteria. For each entity in the group, if it is a class in the context ontology, we extract the class name, relevant properties and directly defined individuals; if it is an individual, we extract the individual name and relevant properties.

VI. IMPLEMENTATION

A prototype of the proposed approach was implemented and integrated into our smart service composition system [30]. The prototype is developed in Java and uses OWL API [25] as the ontology/RDF parser.

Figure 6 shows an annotated screenshot of our service recommendation page. A list of services interested 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 in our travel example) are automatically displayed in the service selection panel on the right side of the Web page. The user can select the services best fit their needs.

We use Freebase [15] as the ontology database. Freebase is an ontology database which extracts structured information from Wikipedia [29]. In Freebase, there are some common entities which are 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 do not reflect the real relations of ontologies. To increase the accuracy of relation identification, we developed a component to filter meaningless entities.

In the development of the prototype, we find that not all the context values have relevant ontology. There are usually no matching ontologies for long phrases, such as context value "plan a trip to Los Angeles" for the context type "activity". We use the following steps to find relevant ontologies for each context value.

(1) We treat the context value as a searching string, and use the entire searching string to search for relevant ontologies from ontology database. Wikipedia (i.e., the

major source of Freebase) defines a large number of entities which include extensive knowledge relevant to contexts. We use Wikipedia as a tool to identify entities in the searching string. If we can find a matching ontology, we annotate this ontology to this context.

(2) If we cannot find a matching ontology for the searching string, we use an adjective & 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 still can keep the important information in the searching string without the adjectives and adverbs. If the removed adjective or adverb is followed by a stop word, we also remove the stop word. 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 and convert the removed adjectives and adverbs to constraints of the annotated ontology. For example, if we cannot find ontology for the context value "luxurious travel", we search for the ontology of "travel" and treat "luxurious" as a constraint for the ontology of "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 relevant ontology using the searching string of a context value, we use synonyms of the context value to search for ontologies and repeat above steps.

VII. CASE STUDIES

The objective of our case study is to evaluate the effectiveness of our approach. We want to examine 1) whether our approach can effectively detect meaningful users' needs represented as classes, individuals and properties in the grouped *SharedElementsSet*; and 2) Whether the generated searching criteria can find the desired services.

Table I lists the context types used in our case study. By providing different values on each context type, we can detect various needs of the user and get different service recommendation. Each set of context values of the context types in Table I is called a scenario in our case study.

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

$$Precision = \frac{|\{relevant\ items\} \cap \{retrieved\ iterms\}|}{|\{retrieved\ items\}|},$$

$$Recall = \frac{|\{relevant\ items\} \cap \{retrieved\ items\}|}{|\{relevant\ items\}|}$$

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

TABLE I. CONTEXT TYPES USED IN OUR CASE STUDY

Context source	Context types
Previous environment	Location (city and county)
Current environment	Location (city and country)
	Activity (described by keywords)
Future environments	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

TABLE II. RECALL AND PRECISION OF DETECTED CONTEXT RELATIONS

Scenarios	Recall	Precision	Number of ignored but existing relations
1	91%	100%	1
2	88%	82%	4
3	91%	80%	5
4	90%	89%	1
5	100%	89%	1
Average	92%	88%	2.4

A. Evaluation of the detected context relations

We use five context scenarios to evaluate the relations detected by our approach. For each scenario, we manually examine its context and identify the potential needs of the user based on our knowledge and the information gathered from the Internet. Then we use our prototype to automatically find user's needs. By comparing the results, we calculate the precision and recall of each scenario. We notice that some relations are ignored by us but are identified by our prototype. For example, when we set "Michael Jordan" as a Favorite celebrity of the user, and one of the context values is the city "New York". Our prototype finds that "New York" is the place of birth for "Michael Jordan". When calculating recall and precision, we also add the missed relations into the relevant items set, i.e., treated as desired needs although we do not manually find them.

The results in Table II show that the ontologies can help identify most of relations among context values. It effectively identifies almost all the relations between cities and their country, celebrity names and their born city, as well as between the spoken language and countries. It is worth to mention that our prototype can identify missed relations in the manual inspection process.

B. Evaluation of Service Recommendation

We use the approach described in Section V to get a set of keywords groups, then we use the keywords in each group

TABLE III. EVALUATION RESULT OF SERVICE RECOMMENDATION

Scenarios	Recall	Precision
1	100%	83%
2	75%	91%
3	67%	75%
4	100%	83%
5	100%	88%
Avarage	88%	84%

as searching criteria to search for online resources. Due to the limited number of public available Web services [2][22] in some specific domains, we do not limit the returned services to Web services in our case studies. The returned services could be any Web resources which contain the desired services, such as Web services, Web applications, and Web pages. We use Google [16] and Seekda [26] as the search engine to search for Web pages and Web services. We manually examine the available services in Seekda for each topic. If there are available Web services in Seekda for a given topic, we use Seekda. Otherwise, we use Google.

Since each scenario in our case study is designed for a certain situation, we manually provide the description of desired services based on each scenario. Then we use our prototype to recommend services. For each query, our prototype chooses the first two returns from the search engine to calculate the precision. By comparing the desired services and recommending services, we calculate the recall and precision as listed in Table III.

The results show that our approach can recommend most of needed information for users in the given context scenarios. However, in Table III, we also can find that in some scenarios, the recall and precision are not stable. Here are some explanations:

1) Some ontologies do not describe all the perspectives of a context value. The incomplete ontologies cause incomplete service recommendation. Meanwhile, we simply use the relations of context values without additional knowledge. If we combine our approach with existing approaches (e.g., [31]), such as providing static service recommendation rules, the precision and recall could be higher.
2) Google and Seekda are mainly designed to search using keywords. However, the collection of keywords provided by our prototype contains also semantics. For example, from the relations of context values, we can know that "Michael Jordan (people, person)" was born in "Brooklyn, New York (place lived, place of birth, location)". The full meaning is expressed by the words and their orders. When search engines which take keywords cannot fully understand the semantic meaning among the keywords. As a result, the results returned from Google and Seekda are not very relevant instead of offering more information relevant to "Brooklyn" and "Michael Jordan".
3) When the number of keywords increases, the results returned by Google or Seekda are likely to detruncate. Especially, we may extract general terms, such as "people", "person", and "location", adding those terms in the searching keywords often drastically reduces the quality of searching results.

VIII. RELATED WORK

There exist a few context models and context-aware systems in literature [4][7][28][18]. Strang and Linnhoff-Popien [28] survey existing context models and classify them into different types based on the data structures. They classify the context models into 6 types: key-value models, markup scheme models, graphical models, object oriented models, logic based models, and ontology based models. They also evaluate those context models using six requirements and show that ontologies are the most expressive model and fulfill most of the requirements. Chen and Kotz [7] investigate the research on context-aware mobile computing. They 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. [4] present a layered conceptual design framework to describe the common architecture principles of context-aware systems. Based on their proposed design framework, they compare the context sensing, context models, context processing, resource discovery, historical context data, security and privacy of various existing context-aware system. The context types mentioned in the aforementioned approaches cannot be dynamically modified. In this paper, we can generate and adjust the context relation models automatically according to different available context values.

Applying context-aware techniques to discover and recommend services has gained lots of attentions. Yang et al. [9][31] design an event-driven rule based system to recommend services according to people's context changes. They provide an ontology-based context model to represent context and utilize the context to assist service discovery. Balke and Wagner [5] 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. Abbar et al. [1] provide an approach to recommend services using the log files of a user and the current context of the user. To select and recommend services, those approaches either require historical data which are usually not available in practice, or need to predefine the specific reactions on context using rules. In our paper, we automatically extend the semantics of the context value using publicly available ontologies, and use the semantics to detect user's needs to recommend services.

IX. CONCLUSION AND FUTURE WORK

In this paper, we present an approach to dynamically derive context model from ontologies and recommend services using context. Given a set of available context types and values, our approach can dynamically detect their relations and construct a context relation model. By discovering the semantic relations among context values, our

approach can identify a user's needs hiding in the context values and generate searching criteria for service discovery. The case study shows the effectiveness of our approach. In the future, we plan to use a larger number of context scenarios to better evaluate the benefit of our approach.

ACKNOWLEDGMENT

This work is financially supported by NSERC and the IBM Toronto Centre for Advanced Studies.

REFERENCES

[1] S. Abbar, M. Bouzeghoub, S. Lopez, "Context-aware recommendation systems: a service-oriented approach", 35th International Conference on Very Large Data Bases (VLDB), Lyon, France, Auguest 24-28, 2009

[2] E. Al-Masri, Q. Mahmoud, "Investigating web services on the world wide web: An empirical study", In Proceedings of 17th ACM International World Wide Web Conference, Beijing, China, 2008, pages: 795-804

[3] J. Angele and G. Lausen, "Ontologies in F-logic," In S. Staab and R. Studer (editors), Handbook on Ontologies in Information Systems, Springer Verlag, Berlin, Germany, 2004, pages: 29-50

[4] M. Baldauf, S. Dustdar and F. Rosenberg, "A survey on context-aware systems", International Journal of Ad Hoc and Ubiquitous Computing, Volume 2, Issue 4, June 2007, pages: 263-277

[5] W. T. Balke, M. Wagner, "Towards personalized selection of web services," In Proceedings of the International World Wide Web Conference (WWW) 2003, Budapest, Hungary, 2003

[6] P. Brézillon, "Focusing on context in human-centered computing," IEEE Intelligent Systems, Volume 18, Issue 3, May 2003, pages: 62-66

[7] G. Chen and D. Kotz, "A survey of context-aware mobile computing research," Dartmouth Computer Science Technical Report TR2000-231, 2000

[8] P. P. Chen, "The entity-relationship model— toward a unified view of data", ACM Transactions on Database Systems (TODS), volume 1, issue 1, March 1976, pages: 9-36

[9] I. Y. L. Chen, S. J. H. Yang, J. Jiang, "Ubiquitous provision of context aware web services," IEEE International Conference on Services Computing (SCC) 2006, Chicago, USA, September 18-22, 2006, pages: 60-68

[10] 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, available at http://www.w3.org/TR/daml+oil-reference, last accessed on January 11, 2010

[11] DBpedia, available at http://dbpedia.org/, last accessed on January 11, 2010

[12] Dublin Core, available at http://dublincore.org/, last accessed on January 11, 2010

[13] A. K. Dey, D. Salber, G. D. Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," Human-Computer Interaction (HCI) Journal, Vol. 16(2-4), 2001, pages: 97-166

[14] Expedia, http://www.expedia.com/, last accessed on January 30, 2010.

[15] Freebase, http://www.freebase.com/, last accessed on January 6, 2010

[16] Google, http://www.google.com, last accessed on Januray 12, 2010

[17] H.G. Hegering, A. Kupper, C. Linnhoff-Popien, and H. Reiser. "Management challenges of context-aware services in ubiquitous computing environments," 14th IFIP/IEEE International Workshop on Distributed Systems, Operations and Management (DSOM), Lecture Notes in Computer Science(LNCS) 2867, Heidelberg, Deutschland, Oct. 2003, Springer, pages: 246–259

[18] C. Hesselman, A. Tokmakoff, P. Pawar, S. Iacob, "Discovery and composition of services for context-aware systems," 1st European Conference on Smart Sensing and Context 2006, Enschede, The Netherlands, 2006, pages: 67-81

[19] IBM Mashup Center, http://www-01.ibm.com/software/info/mashup-center/, last accessed on January 11, 2010

[20] G. Klyne and J. J. Carroll, "Resrouce Description Framework(RDF): Concepts and Abstract Syntax," W3C Recommendation, Feb. 10, 2004

[21] M. Krause, C. Linnhoff-Popien, M. Strassberger, "Concurrent inference of hight level context using alternative context construction trees," 3rd international conference on automomic and autonomous systems (ICAS), Athens, Greece, June 19-25, 2007, page 7

[22] Y. Li, Y. Liu, L. Zhang, G. Li, B. Xie, and J. Sun, "An exploratory study of web services on the Internet," In Proceedings of 2007 IEEE International Conference on Web Services, Salt Lake City, Utah, USA, 2007, pages:380-387

[23] Z. Maamar, D. Benslimane and N. G. Narendra, "What can context do for Web services," Communications of ACM, Vol. 49. No. 12, December 2006, Pages: 98-103

[24] F. Manola, E. Miller, B. McBride (editors), "RDF primer," W3C recommendation 10 February 2004, available at http://www.w3.org/TR/rdf-primer/, last accessed on January 10, 2010

[25] OWL API, http://owlapi.sourceforge.net/, last accessed on January 7, 2010

[26] Seekda, http://webservices.seekda.com/, last accessed on January 12, 2010

[27] M. K. Smith, C. Welty, D. L. McGuinness (editors), "OWL Web ontology language guide," W3C Recommendation (2004), available at http://www.w3.org/TR/owl-guide/, last accessed on January 11, 2010

[28] T. Strang, and C. Linnhoff-Popien, "A context modeling survey," The First International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham, England, September, 2004

[29] Wikipedia, http://en.wikipedia.org/wiki/Wikipedia:About, last accessed on January 6, 2010

[30] H. Xiao, Y. Zou, R. Tang, J. Ng, L. Nigul, "An automatic approach for ontology-driven service composition," Proc. IEEE International Conference on Service-Oriented Computing and Applications (SOCA) 2009, Taipei, Taiwan, 14-15 December 2009, pages: 1-8

[31] S. J. H. Yang, J. Zhang, I. Y. L. Chen, "A JESS-enabled context elicitation system for providing context-aware Web services," Export Systems with Applications, Volume 34, Issue 4 (May 2008), pages: 2254-2266