Schema Matching for Context-Aware Computing

Wenwei Xue, Hungkeng Pung, Paulito P. Palmes
School of Computing
National University of Singapore
3 Science Drive 2, Singapore 117543
{dcsxw, dcsphk, dcsppp}@nus.edu.sg

Tao Gu
Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore 119613
tgu@i2r.a-star.edu.sg

ABSTRACT
Context-aware computing is a key paradigm of ubiquitous computing in which applications automatically adapt their operations to dynamic context data from multiple sources. Managing a number of distributed sources, a middleware that facilitates the development of context-aware applications must provide a uniform view of all these sources to the applications. Local schemas of context data from individual sources need to be matched into a set of global schemas in the middleware, upon which applications can issue context queries to acquire data. In this paper, we study this problem of schema matching for context-aware computing. We propose a multi-criteria algorithm to determine candidate attribute matches between two schemas. The algorithm adaptively adjusts the priorities of different criteria based on previous matching results to improve the efficiency and accuracy of succeeding operations. We further develop an algorithm to categorize a new local schema into one of the global schemas whenever possible via a shared attribute dictionary. Our results based on schemas from real-world websites demonstrate the good matching accuracy achieved by our algorithms.

Categories and Subject Descriptors
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous; H3.3 Information search and retrieval: Clustering; H2.1 Logical design: Schema and subschema.

General Terms
Algorithms, Management, Performance, Design.

Keywords
Context awareness, ubiquitous computing, middleware, context attributes, context schemas, schema matching.

INTRODUCTION
Context awareness is an essential enabler for the unattended operation/behavior adaptation of applications in ubiquitous computing [1]. A context-aware application is conscious of the context of many data sources in the physical world, such as a person or a shop, and adapts to such dynamic data automatically and continuously. We adopt the definition of *context* by Dey [2] as “any information that can be used to characterize the situation of an entity”. We then define a *context source* as a logical component from which context data related to some physical-world entity can be acquired.

Examples of real-world context data include the location of a person, the temperature in a house and the opening hours of a shop. Correspondingly, the context sources can be the PDA of the person, the PCs located in the house or shop. A context source often collects context data from a local sensor network or legacy database [4,6]. It then provides the data to external applications in a common format based on some context model such as key-value pairs or ontologies [1].

There has been a lot of recent research effort in context-aware computing on building generic middleware systems to support the development of various applications [7]. One main function of a context-aware middleware is to manage data from numerous context sources and provide the data via different application interfaces, such as services [4] or declarative queries [6]. In this paper, we investigate the problem of matching data schemas among multiple context sources in the infrastructure of a context-aware middleware under development in our research project.

From the perspective of data management, a *schema* is a specific description of data in terms of a general data model [18]. Each context source in our infrastructure maintains a local schema that describes all context data it can provide. The schema is submitted to the middleware when the source is registered. We call such a schema a *context schema*. We define a *context attribute* as a kind of context data described in a schema, e.g., *location*, *temperature* and *opening_hours*.

Our problem of *context schema matching* is how to integrate individual local schemas from different context sources into a set of global schemas in the middleware, as depicted in Figure 1. Given the global schemas, the applications on top of the middleware are able to see a unified abstract view of underlying heterogeneous sources and access context data in a consistent manner. The global schemas dynamically evolve when the context attributes provided by numerous sources are incrementally added and clustered into them.

Applications can clearly view these related attributes and quickly access new attributes when they become available. To the best of our knowledge, there is no previous work [7] that studies this problem of automatic schema matching in a context-aware middleware.

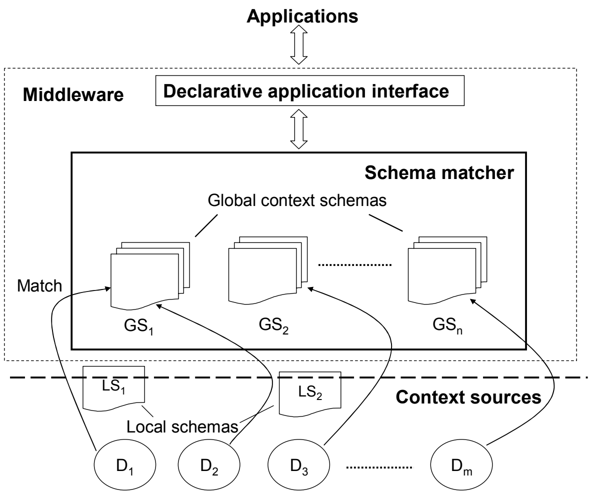

Figure 1. Schema matching in context-aware middleware

Schema matching has been widely studied in the database literature [3,9,12,13,17]. Compared to traditional database scenarios, the novel features of context sources in ubiquitous computing pose several notable challenges to our problem of context schema matching:

- *No instance-level matching*. A context schema describes many sensory attributes whose values are collected from physical sensors on-the-fly and in general not stored into databases due to the unbound data volume and update cost [11,21]. Furthermore, these real-time sensor readings are intrinsically unreliable while it is impractical to assume all context sources are internally equipped with sufficient mechanisms for sensor data cleaning. Therefore, instance-level approaches in traditional databases [17] that utilize attribute values from schema instances for matching are error-prone and do not appeal to our scenario.
- *No constraint-based matching*. According to our real-world case studies, multiple context sources usually define different data properties, such as types and value ranges, for equivalent attributes. For instance, one source provides its *temperature* in integer values with a pre-defined unit (e.g., 30°C), while the other may provide the value and unit as a string (e.g., "30°C"). The goal of our context schema matching is to explore the high-level semantic equivalence between context attributes and schemas while the low-level implementation details of individual schemas are of less concern. Therefore, constraint-based matching in traditional databases [17] that utilizes data properties in the schemas does not appeal to our scenario either.
- *Dynamic joining of context sources*. Numerous sources in various context domains, such as persons, houses and shops, may register to the middleware continuously over time. A context schema matcher in our scenario should require tiny computation cost to ensure the timeliness of matching a new local schema into the global schemas. In the meanwhile, the matcher must be adaptive to changing domain patterns of schema inputs. Many schema matchers for traditional databases [17] employ a supervised learning approach, which makes them inappropriate for our scenario due to the heavy computation cost or the human effort to obtain and incrementally update the training dataset.
- *Autonomous context sources.* Our middleware allows any networked context source to register for data provision. These sources are indifferent to one another and can be owned by multiple organizations at different geographical locations. Each source composes and submits its context schema to the middleware separately. It is unreasonable for the middleware to impose a common name space among all sources for schema composition. Moreover, there is no sharing of system-level processing capabilities among the sources as those in federated databases [18].

Addressing these challenges, we propose a lightweight and adaptive schema matcher for context-aware computing in this paper. Our matcher consists of two main algorithms: (i) an algorithm for Context Attribute Matching (*CAM*), and (ii) an algorithm for Context Schema Matching (*CSM*).

CAM computes the similarity between a pair of context attributes by comparing their names, and text descriptions if available. The similarity is measured by one of several pre-defined matching criteria, such as stemming and substring detection. The criteria are invoked in a decreasing order of their priorities, whose values are periodically recomputed based on the previous matching accuracy each criterion has achieved. This enables the algorithm to be adaptive to the current patterns of schema inputs for efficiency and accuracy improvement: the criterion with better performance recently is used first to seek for a possible match for an attribute pair.

The candidate matches found by CAM are handed over to the system administrator of our middleware for confirmation. Redundant attributes in a new local schema with a confirmed correct match are replaced by corresponding attributes in the global schemas. Such user feedback is the basis for CAM to adaptively adjust the priorities of its multiple criteria.

CSM examines the similarity between a local schema and each global schema based on their accumulated attribute similarities output from CAM. If the two schemas have a similarity score larger than a threshold, the local schema can be merged into the global schema. Again multiple candidate matches are possible and human confirmation is involved.

The human confirmation of matching algorithm outputs is motivated by the fact that only the middleware administrator knows or has the right to tell "what is a correct match". Since the matcher provides all candidate matches while the administrator only confirms which are correct, the human effort involved is minor. If confirmation is undesirable, this option can be disabled in our schema matcher. It suggests the matcher is given the full power to regard every matching decision it makes as correct. In this case, for each local attribute or schema, the matcher automatically regards the candidate match for it with the highest similarity score as the ground truth and incorporates this match into the global schemas. The administrator can also specify missing matches to amend the matcher outputs whenever convenient.

We have evaluated the performance of our context schema matcher using real-world schemas extracted from a number of websites. The experimental results demonstrate that even without human confirmation, our proposed algorithms can achieve high matching accuracy with tiny computation cost.

Although in this paper we present the design of our schema matcher under the setting of context-aware middleware, the proposed matching algorithms are generic and can be applied to other application scenarios of context-aware computing. One example is the local schema matching between two neighboring peers of context sources in a P2P database [13].

The remainder of the paper is organized as follows. We first introduce some background knowledge of our context-aware middleware related to schema matching. Next, we present our algorithms for context attribute and schema matching in detail. We then present initial performance evaluation results for our matching algorithms. Finally, we discuss related work and conclude the paper with future research directions.

SYSTEM BACKGROUND

Overview of Middleware Infrastructure

Figure 2 shows the overall four-layered infrastructure of our middleware. The schema matcher is a component of the context data management layer. This layer provides a SQL-based query interface over the global schemas that allows services/applications in the upper layers to acquire context data from sources in the lower layer. SQL-based queries have been investigated to be comparable to more complex context queries such as RDF queries [5]. Distributed query dissemination, execution and optimization techniques are also equipped in the layer. The service management layer provides functionalities of service organization, discovery and workflow-based composition. A detailed description of the middleware components in each layer is beyond the scope of this paper.

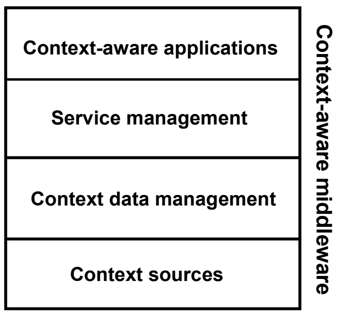

Figure 2. Infrastructure of our context-aware middleware

Context Modeling

The current design of our schema matcher assumes a simple attribute-value approach to context modeling. Similar to the schema of a table in relational databases [18], a context schema in our middleware contains a schema name as well as the names, types, text descriptions and other additional information about a set of context attributes.

The motivation is to make the initial design of our schema matcher generic and indifferent to specific context modeling approaches. Extending the matcher to incorporate and utilize the complex features of more expressive context models, such as the widely-used ontology model in today's semantic web [1,4], is an ongoing direction of our research work.

Schema Templates

A schema template is an XML file provided by a context source upon registration. It specifies the context schema of the source to our schema matcher. The implementation details of the schema templates are omitted here.

We make the current snapshot of global schemas in our middleware publicly-available via a web interface, together with example templates for these schemas. They serve as a guidance and reference for the context sources to compose specific templates for their local schemas.

In the schema template, a context source can explicitly specify the exact matching between attributes in its local schema and those in the global schemas. The source can also indicate which global schema matches the local schema. This alleviates the schema matcher's workload however in most cases a context source is not able or willing to do such manual matching.

Schema Mapping

After a local schema is matched with the global schemas, a schema mapping is sent to the corresponding context source. The mapping specifies the following information to the source: (i) for each attribute in the local schema, the attribute has been merged to which attribute in the global schemas, or it defines a new attribute in a global schema, (ii) whether the local schema is merged into one of the existing global schemas, or it becomes a new global schema. The matcher does not store the mapping between the global schemas and individual local schemas. Such information is distributed to corresponding context sources and stored locally instead.

The syntax of a context query issued to our middleware is based on the set of global schemas. When the context query is routed to a source of required data, the source converts the query syntax from the global schemas to its local schema using the schema mapping it stores and processes the query.

Our matching algorithms ensure that only new attributes or schemas will be added into the global schemas, whereas no existing attribute or schema will ever be deleted. This ensures consistency of previous schema mappings when the global schemas are continuously evolving upon registration of new sources. Otherwise, the schema matcher must send updates for previous mappings to all related sources upon removal of any existing global attribute or schema. This brings heavy extra workload to the system.

MATCHING OF CONTEXT ATTRIBUTES

Our context schema matcher is a name-based matcher [17]. CAM matches a pair of attributes according to the similarity between the attribute names, or text descriptions if available. CSM matches a new local schema with the global schemas based on the attribute match outputs of CAM. In this regard, our schema matching problem maps to a text processing problem over the schema templates.

We present our adaptive, multi-criteria matching of context attributes in this section. The matching of context schemas will be described in the next section.

Multiple Matching Criteria

In order to test whether a pair of context attributes forms a candidate match, CAM applies a list of pre-defined matching criteria. These criteria measure the similarity between two attributes in different ways based on the linguistic similarities in their names and descriptions. Each criterion is associated with a priority in [0,1], whose value is periodically adjusted. When a criterion reports a match for an attribute pair, the similarity score of the pair is assigned to be the current priority of the criterion.

The following matching criteria are employed for CAM in the current prototype of our schema matcher:

(1) *Equality*. The criterion reports a match if the names of the two attributes are equivalent.

(2) *Stemming*. The criterion reports a match if the names of the two attributes are equivalent after stemming. We use the Porter's Algorithm for stemming [16].

(3) *Substring*. The criterion reports a match if one attribute name is a substring of the other after stemming.

(4) *Longest Common Substring (LCS)*. Suppose the names of the two attributes are s_1 *and* s_2 and the longest common substring of these two strings is s. The criterion reports a match if $s.len > N$ and $s.len / min(s_1.\text{len}, s_2.\text{len}) > \tau$. We use the dynamic programming algorithm for LCS detection [10]. The integer $N > 0$ and the real number $\tau \in (0,1)$ are two pre-defined thresholds. The default values of N and τ are 2 and 0.8 in our schema matcher.

(5) *Synonym*. The criterion reports a match if the names of the two attributes are synonyms (e.g., *house* and *home*, *phone* and *telephone*). We download and use the WordNet [20] lexical database for synonym checking.

(6) *Description*. The criterion is only used when both of the two attributes have a text description. Language keywords are extracted from the descriptions by removing those words appeared in a stop word list [19]. The keywords are then stemmed and stored with corresponding attributes.

Suppose the two attributes are a and b and their keyword sets are K_a and K_b. $K = K_a \cap K_b = \{k_i\}$ ($i = 1, 2, \ldots, n$).The criterion reports b matches a if Equation (1) is satisfied. f_{kai} in the equation is the appearance frequency of k_i in K_a and N_{ka} the size of K_a. $\rho \in (0,1)$ is a pre-defined threshold whose default value is 0.7 in our schema matcher.

$$\frac{\sum_{i=1}^{n} f_{Kai}}{N_{Ka}} > \rho \qquad (1)$$

As revealed by Equation (1), the matching reported by this criterion may not be symmetric. In other words, b matches a does not mean a matches b in terms of their descriptions.

Description matching is very useful to match an attribute to an equivalent combination of other attributes. For instance, our schema matcher can determine two attributes *latitude* and *longitude* in a local schema match the attribute *location* in the global schemas if and only if all attributes are specified with similar text descriptions. Therefore, we suggest context sources to provide attribute descriptions in their local schema templates to enhance the opportunity of matching.

Criterion (1) is a special case of Criterion (2). We separate them since Criterion (2) is more time-consuming and there is no need to invoke it if Criterion (1) has been satisfied. The same case applies to Criteria (3) and (4).

The six matching criteria are sorted in a decreasing order of their priorities in the list all the time. The initial priorities of all criteria are one. The default order of Criteria (1)-(6) is the same as the order we present them above. We have tried several other criteria, such as edit distance and hypernym [17], but excluded them from our schema matcher due to the unsatisfactory accuracy they achieved on trial datasets.

Algorithm for Attribute Matching

Based on the list of matching criteria, the CAM algorithm in our schema matcher is depicted in Algorithm 1. An equivalent illustration of the algorithm is given in Figure 3.

Given a pair of context attributes, the algorithm invokes the matching criteria in order (Line 1) to look for a candidate match between the pair. If the option of confirmation is enabled, a match achieved by some criterion (Line 2) is presented to the administrator to decide whether it is correct (Line 3). Otherwise, the function *attributeConfirm* simply returns *true* every time it is called.

Algorithm 1: Context Attribute Matching

```
Input: a pair of context attributes ($a$, $b$)
Output: candidate matches between $a$ and $b$
 1: for each $mc_i$ ($1 \le i \le n$) in mcList do
 2:   if attributeMatch($a$, $b$, $mc_i$) then
 3:     isCorrect = attributeConfirm($a$, $b$, $mc_i$);
 4:     if isCorrect then
 5:       $mc_i$.matchCount++;
 6:       append ($a$, $b$, $mc_i$) to matchList;
 7:     else $mc_i$.matchCount−− ;
 8:     if | $mc_i$.matchCount | == $N_a$ then
 9:         updateMatchingCriterion($mc_i$);
10:     if isCorrect then return;
```

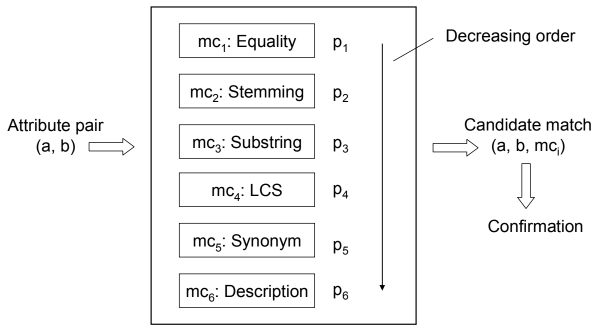

Figure 3. Multi-criteria context attribute matching

If a candidate match is correct, a local match counter of the corresponding criterion is increased (Line 5). In contrast, this counter will be decreased upon a wrong match (Line 7). Correct candidates are inserted into a global match list (Line 6) and are used later for context schema matching.

When the match counter of a criterion has its absolute value reach a threshold N_a (Line 8), the priority of the criterion is increased or decreased by $\Delta \in (0,1)$, depending on whether the counter value is positive or negative. The intuition is to give higher/lower priority to a criterion that has produced considerable number of correct/wrong matches recently. The default values of the two parameters are $N_a = 20$ and $\Delta = 0.1$.

After the priority of a criterion is updated, the match counter of the criterion is reset to zero. The criterion then switches its position up or down in the list based on the new priority until the correct order of the list is recovered. The priority of a criterion may become less than zero or larger than one after an update. In this case, the priority is set to zero or one after the list re-ordering is finished. All these operations are performed in the *updateMatchingCriterion* function (Line 9).

By dynamically updating the priorities of matching criteria based on their recent accuracy, the algorithm is adaptive to the changing patterns of schema inputs. It is expected that such adaptivity will enhance the accuracy and efficiency of the matching process. The adaptivity also makes the initial order of the criteria unimportant at all.

We have designed different adaptive tuning mechanisms for the threshold parameters in individual matching criteria and implemented them in the *updateMatchingCriterion* function. For instance, the τ value of LCS is decreased by Δ whenever the priority of the criterion is increased. The intuition is to allow more opportunities for a criterion to report candidate matches when its recent accuracy is good and to make the default values of the threshold parameters less significant.

If a criterion reports a matching for an attribute pair, its succeeding criteria in the list need not be invoked for the pair any more (Line 10). In this way, the algorithm always tries to use a single criterion with the highest recent accuracy to offer a candidate match for an attribute pair.

In addition to adaptivity, another advantage of the algorithm is its flexibility. At any time new matching criteria can be easily added to any position in the list with proper initial settings of their priorities, and old criteria can be removed. The execution of the algorithm will not be affected.

MATCHING OF CONTEXT SCHEMAS

Now we describe how our schema matcher integrates a local schema into the current set of global schemas. Algorithm 2 briefly depicts our algorithm for context schema matching, CSM.

Algorithm 2: Context Schema Matching

```
Input: a local context schema $LS$,
       the current set of global schemas $R_{GS}$
Output: candidate match of $LS$ in $R_{GS}$
1: selfMatching($LS$);
2: if $LS$ is specified to be a part of $GS$ in $R_{GS}$ then
3:     intraSchemaMatch($LS$, $GS$);
4: Else
5:     dictionaryMatch($LS$);
6:     $S$ = interSchemaMatch($LS$, $R_{GS}$);
7:     schemaConfirm($S$);
8:     updateSchema($S$, $R_{GS}$);
9:     updateDictionary($S$);
```

Intra-schema Matching

We first discuss a simple but restricted case in which the local schema is known to be part of a global schema (Lines 2-3 in Algorithm 2). As previously presented, this semantics can be specified in the schema template.

As shown in Figure 4, in this case the process of schema matching involves the matching of every pair of attributes in the local and global schemas using CAM. This process is very similar to the join operation in relational databases [17, 18]. All candidate matches of attribute pairs are available in the global match list after this "joining" process.

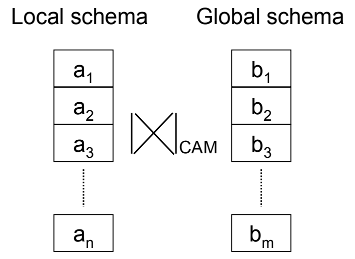

Figure 4. Match a local schema with corresponding global schema

For each attribute in the local schema, CSM only allows it to be finally merged to one attribute in the global schema. As a result, if two candidate matches (a, b_1) and (a, b_2) are both confirmed as correct, the one achieved by a criterion with a larger priority is kept and the other is removed from the global match list.

The candidate matches in the global match list are examined when the process of schema matching ends. Each attribute in the local schema is either merged to its matched attribute or added as a new attribute in the global schema. A schema mapping is formed and sent to the context source.

An *appearance counter* is kept for each attribute in a global schema. It indicates the attribute is the aggregation of how many equivalent attributes from different local schemas.

Before integrating a local schema with the global schemas, CSM first match the schema with itself to remove redundant attributes (Line 1 in Algorithm 2). This process is the same as that in Figure 4 and we call it *self matching*.

Inter-schema Matching

We next describe the general case of our schema matching in which the local schema has not been explicitly specified to be a part of any global schema. In this case, a naïve approach is to examine matching between the local schema and every existing global schema. Such linear search is time-consuming when the number of global schemas is large.

Our CSM algorithm addresses the problem by maintaining a shared attribute dictionary among all global schemas. The motivation is that many attributes are common to different schemas. As a result, the total number of attributes in all global schemas after matching can be reasonably small. For instance, as illustrated in Figure 5, the *name*, *location* and *phone* attributes are shared by the three schemas describing context data of real-world shops, houses and persons.

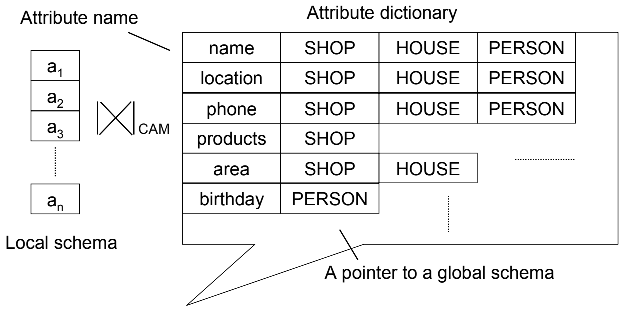

Figure 5. Match a local schema with attribute dictionary

The dictionary includes all context attributes that appear in one or more current global schemas. Each dictionary entry contains the name of an attribute, keywords extracted from the attribute descriptions (if any), and a list of pointers each of which points to a global schema that the attribute belongs to. Accordingly, a global schema contains a number of pointers to the entries in the dictionary. Figure 5 illustrates the attribute dictionary with example context attributes and their corresponding global schemas.

We ensure any two attributes in the dictionary cannot match each other via CAM. Otherwise they will be merged into a single attribute. For instance, suppose an attribute *telephone* is to be newly incorporated into the dictionary. The attribute is merged into the existing attribute *phone* (synonyms). No new entry is appended to the dictionary.

During the matching process in CSM, the new local schema performs a join with the attribute dictionary using CAM as the join predicate, as shown in Figure 5 (Line 5 in Algorithm 2). For each attribute in the local schema, a set of attributes in the dictionary each of which forms a candidate match with it are obtained after the join. Unlike the previous case of intra-schema matching, all these candidate matches will be kept for the attribute. The candidates are further filtered if human confirmation is involved.

For each existing global schema, CSM computes the total number of attributes in the local schema having at least one candidate match in this global schema (Line 6). This number is called the *match count* of the global schema in terms of the local schema, denoted as N_{mc}. Suppose the total number of attributes in the local schema is N_l, the similarity between this local and global schema pair is measured as N_{mc} / N_l.

All global schemas with a similarity score N_{mc} / N_l larger than a threshold N_d will be regarded as a candidate match to the local schema. At most one of them can be confirmed by the administrator as a correct match, or CSM automatically selects the one with the highest similarity score as the correct match (Line 7) and breaks ties randomly. The default value of N_d is 0.6 in our schema matcher.

The local schema is merged into the corresponding global schema if there is a correct match (Line 8). The appearance counters of the global schema attributes are updated. Even if an attribute in the local schema does not have any match in this global schema, the attribute is still aggregated with its correct match in another global schema if such match exists. Otherwise, the attribute is inserted as a new entry in the dictionary with a pointer to the global schema (Line 9).

The local schema is inserted as a new global schema in the middleware if it does not match any existing global schema. Attributes in the local schema are aggregated with existing entries or inserted as new entries in the dictionary, the same as the situation with a correct schema match.

In our schema matcher, different matching orders of local schemas from the same set of context sources can result in different syntactic evolution of the global schemas. As an example, the name of an attribute in the global schemas will be that in the first added local schema having the attribute. This syntax difference does not affect the semantic accuracy of the global schemas. However, a wrong attribute or schema match incurs incorrect semantics in the global schemas and may affect the correctness of the subsequent matches in a cascading way. This is one of the main reasons why we add human confirmation to the matcher and try best to eliminate the wrong matches. As for the missed matches, they add redundant attributes in the global schemas and have less significant semantic impact.

PERFORMANCE EVALUATION

In this section, we present an initial performance evaluation of our context schema matcher.

Schema Dataset

The schema dataset we used for our experiments contains schema information extracted from 30 real-world websites.

We used these schemas because websites are heterogeneous, autonomous data sources in the Web that can be categorized into different domains, which bears some similarity to our context sources over the semantic web. These schemas are composed by website administrators with domain-specific knowledge in a way similar to that for our context sources. Both involve the human understanding and representation of various kinds of data.

The best dataset to evaluate our context schema matcher is the actual context schemas composed by the administrators of real-world context sources. We are collecting a survey schema dataset from many researchers having experiences on sensor network deployment and management for a further evaluation of our matcher in future.

The websites in the 30-schema dataset can be categorized into three context domains: SHOP, HOUSE and PERSON. Each website is an information directory for the physical-world entities in the corresponding domain. For instance, a HOUSE website introduces a number of houses for sale or rent in a country. There are 10 websites in each domain.

A schema is composed from the HTML form template of a website that describes the entities in the domain. We copied the names of form fields directly to be the attribute names in the schema. For a field whose name has multiple words, such as "opening hours" for a shop and "first name" for a person, we used underscores to connect the compound words since the attribute names in our context schemas do not allow spaces. If there are two or more alternative names shown in the HTML form for a field, such as "first name" and "given name" for a person, the first one was used as the attribute name in the schema whereas the others were used as descriptions. If available, the descriptions of form fields were also copied directly and used as attribute descriptions.

Domain	Example attributes	Example websites
SHOP	*name*, *category*, *products*	http://www.yellowpage.com.au/ http://www.made-in-china.com/
HOUSE	*address*, *no_of_rooms*, *price*	http://www.singaporeexpats.com/ http://www.flatsdb.com/
PERSON	*date_of_birth*, *phone*, *e-mail*	http://www.myspace.com/ http://person.com/

Table 1. Example attributes of schemas in different domains from real-world websites

The number of attributes in the 30 schemas varies from 6-29. The average number of attributes per schema is 12. Table 1 lists a few example attributes and source websites belonging to each of the three domains.

The correct list of matches among the attributes in the 30 schemas of the dataset was recorded during the composition of the schemas. It serves as the ground truth in the evaluation of the accuracy achieved by our matching algorithms. In the ideal case, our schema matcher should be able to merge all corresponding schemas belonging to the same domain so that an input trace of the 30 schemas will only result in 3 global schemas.

Accuracy of Attribute Matching

We used two performance metrics, *precision* and *recall*, to evaluate the accuracy of our proposed matching algorithms. Let N_1 be the number of matches reported by the algorithm over the dataset, N_2 the number of correct matches within N_1 and N_3 the total number of correct matches in the dataset. The two metrics are defined below:

$$precison = N_2 / N_1 \quad (2)$$

$$recall = N_2 / N_3 \quad (3)$$

We first evaluated the accuracy of CAM. Figure 6 shows the precision and recall achieved by CAM when the dataset size varied from 6 to 30 schemas. For each dataset size, the number of schemas in each of the three domains was the same. For example, the dataset of size 18 had 6 schemas in each domain.

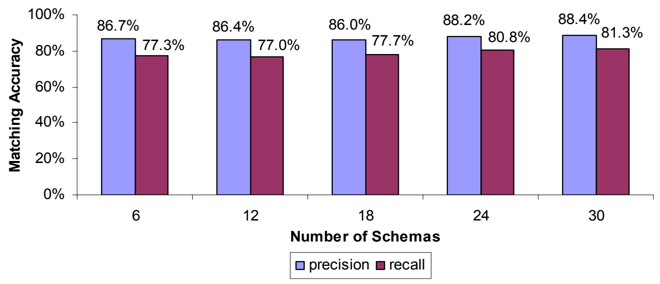

Figure 6. Accuracy of attribute matching with different dataset sizes

Each value in the figure is the average of several experimental runs. For each run of the experiment, a random subset was selected from the total of 30 schemas to avoid bias. The same setting was used for all figures shown in the following.

In Figure 6 we see that CAM achieved good accuracy and stable results over datasets of different sizes. The precision was around 86-88% while recall was between 77-81%. The stable performance of CAM is desirable since the schema matcher needs to run continuously and provide consistent performance over a long period of time.

Figure 7 illustrates the accuracy of different CAM variants with a fixed dataset size of 30 schemas. In the figure, CAM refers to the full implementation of our algorithm for context attribute matching with the six criteria. CAM-SUB is the variant of CAM that does not use substring detection in Criteria (3) and (4), CAM-SYN does not use synonym detection in Criterion (5), and CAM-SUBSYN does not use both substring and synonym detection in Criteria (3) to (5). Human confirmation is disabled in this experiment. Hence, the results in the figure only depict worst-case performance of the variants without supervision. Since there are only few attributes having descriptions in the 30 schemas, the CAM-DESCRIPTION variant was not included in our experiment.

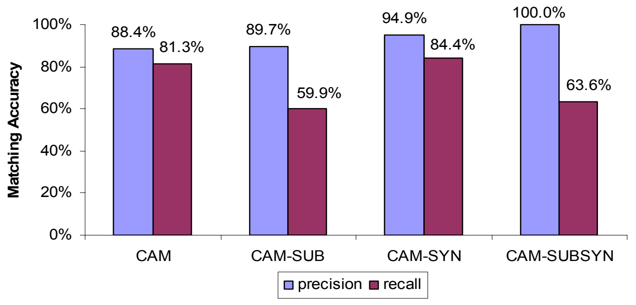

Figure 7. Accuracy of attribute matching for different CAM variants without human confirmation

As shown in Figure 7, CAM-SUBSYN achieved 100% precision in the experiment while CAM-SUB had 89.7% and CAM-SYN had 94.9%. These values are all larger than the 88.4% precision of CAM. The result suggests for our schema dataset, substring and synonym detection were the sources of wrong matches whereas equality and stemming always produced correct matches.

The figure indicates synonym detection caused more wrong matches than substring detection. One possible explanation is that the synonyms are too general or not domain-specific, thereby producing many wrong matches upon the ground truth of a particular domain. Also, there is a high probability for related attributes to share a common substring, because of the tendency of people to use attribute names based on common root words.

As for recall, the figure shows that substring detection was very helpful to improve this metric. Without these criteria, the recall of CAM-SUB was only around 60%. Compared to CAM which had 81.3% recall, CAM-SUB suffered more than 20% decrease in performance. The main reason is the nature of the dataset. It is characterized by the prevalence of multi-word attributes that can only be matched by substring detection (e.g., *company_name* vs. *name* and *no_of_rooms* vs. *number_of_rooms*).

Interestingly, Figure 7 reveals that synonym detection had no contribution to the improvement of recall. The criterion even decreased the recall slightly in our experiment. CAM-SYN had a recall of 84.4%, which is larger than the 81.3% one of CAM. This was because for our schema dataset, synonym detection not only provided a few correct matches, but also made many wrong matches such as *state* and *land*. A previous wrong match caused several succeeding correct matches to go undetected, which decreased both precision and recall. Furthermore, the number of wrong synonym matches was larger than that of the correct synonym matches in the experiment. This caused a negative impact on the overall performance of precision and recall.

The same problem existed for substring detection. However, substring detection achieved much more correct matches than wrong matches in the experiment. This caused positive improvement of the performance as a whole.

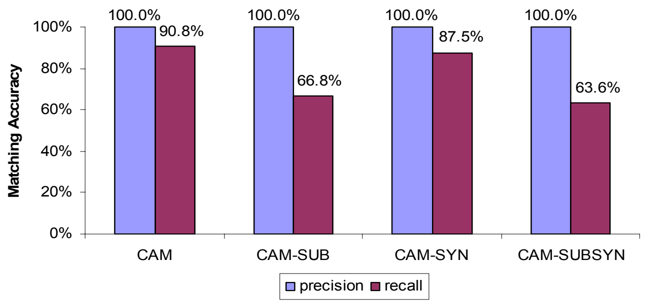

Figure 8. Accuracy of attribute matching for different CAM variants with human confirmation

As a counterpart of Figure 7, Figure 8 shows the accuracy of different CAM variants with human confirmation of the ground truth. Compared to Figure 7, results in this figure described the best-case accuracy achievable by the variants. The dataset size was still 30 schemas in this experiment.

In Figure 8, the precision of all variants was 100% since human confirmation filtered out wrong matches. There was also a significant increase in the recall performance of all variants because more correct matches were detected by both substring and synonym detection without side effects of the wrong matches. Figure 8 complements the results of Figure 7 and reiterates our prior observation regarding the superiority of substring detection over synonym detection with respect to the current schema dataset. This conclusion is indicated by the superior recall performance of CAM-SYN compared to CAM-SUB in both figures.

To summarize, results of Figures 7 and 8 suggest that real-world schemas from the Web tend to use the same names or names with the same stem for equivalent attributes. CAM-SUBSYN achieved as high as 100% precision and 64% recall upon our dataset. The schemas also often use names with common substrings for equivalent attributes, which makes substring detection an important component for any matcher. On the other hand, the success of using synonym detection depends on choosing synonym sources that have the proper context of the schemas being processed in order to truly identify the equivalent attributes.

Accuracy of Schema Matching

To evaluate the accuracy of CSM, we first created three global schemas in the schema matcher corresponding to the SHOP, HOUSE and PERSON domains. Each global schema contains the union of all attributes appeared in the schemas of the domain in our dataset. For equivalent attributes, only a randomly-chosen one was added into the global schema.

After running several trials of the experiment using all the 30 schemas, our schema matcher had 100% precision and recall in all trials. CSM successfully matched every schema in the dataset to the corresponding global schema of the domain. The result demonstrated the effectiveness of CSM.

Computation Overhead

We implemented our schema matcher in Java and ran the experiments on a PC with Intel Pentium 2.0GHz CPU, 1GB main memory and Windows XP OS. The computation cost of matching a schema in the dataset in our experiments was between 0-200ms, which is tiny and negligible. The result indicated the efficiency of our matching algorithms.

We have performed trial experiments with different initial orders of the matching criteria in CAM and found that the criteria can be adaptively re-ordered based on characteristics of our 30-schema dataset. As a typical example, the criteria of substring detection were always put ahead of synonym detection after an experimental run, because there are more substring matches than synonym matches in the dataset. We are crawling hundreds of schemas from various websites to form a larger experimental dataset for an ongoing extensive evaluation on the adaptivity of our matcher.

Our current results revealed many interesting observations on real-world schemas that guide us to the enhancement of our matching algorithms. One possible research direction is the incorporation of a Latent Semantic Analysis (LSA) [8] criterion in CAM by mining web documents to build domain-specific context dictionaries, which may help to lessen the number of wrong matches caused by synonym detection.

RELATED WORK

Existing approaches to database schema matching can be categorized into the following taxonomy [12,17]:

(1) Schema-level vs. Instance-level – whether to consider the contents of data or just the schema information.

(2) Element-level vs. Structure-level – whether to consider the schema elements individually or complex combinations of the elements.

(3) Language-based vs. Constraint-based – whether to consider the matching based on the texts (names) of the elements or based on certain constraints.

(4) Hybrid vs. Composite – whether the matcher employs a combination of dependent matching criteria or independent ones that are aggregated using certain weights.

According to this taxonomy, the context schema matcher we propose is schema-level, element-level, language-based and hybrid. Instance or constraint-based matching is not quite appropriate to our problem as described in the Introduction. Structure-level or composite matching is still applicable and we are investigating how to incorporate them to improve our matching algorithms. We have not employed structure-level matching in our current algorithms since the computation cost of checking complex schema structures is considerable. Moreover, human confirmation can help to reduce certain wrong matches caused by ignoring structures, e.g., wrongly matching two schemas in different domains with a common subset. Composite matching brings the problem of adaptive tuning of the criteria weights.

LSD [3] uses machine learning to match XML data sources to a pre-defined global schema. Both schema and instance-level matching are performed in LSD. It uses a composite model and applies supervised learning to adjust the criteria weights based on user-supplied training data. In comparison, our matcher integrates schemas from context sources based on an attribute-value relational model. Like unsupervised learning approaches our matcher does not reply on a training dataset that needs substantial user effort to prepare. It can work reasonably well even without human effort of match confirmation. Our matching algorithms are designed to be lightweight and adaptive to the real-time patterns of schema inputs from context sources in different domains.

Cupid [12] employs a hybrid model of three stages. The first stage uses language-based matching to categorize elements based on names, types and domains. It computes element similarities using substring matching and other methods. The second stage converts the original schema into tree structure and applies bottom-up matching to find structural similarity between pairs of elements. The last stage uses the calculated weighted mean from previous two stages to decide mapping among elements and structures. We apply substring detection in our matcher for context attributes and demonstrate that this criterion is very useful in practice.

SemInt [9] utilizes clustering and neural networks to find attribute matches in different database schemas based on the signatures generated by multiple matching criteria. The authors observed that clustering performs well in matching nearly identical attributes whereas the neural network with training is better in less similar attributes. We use a context schema matching algorithm similar to clustering that groups the local schemas into global ones.

Our matcher is aimed at effectively and efficiently handling the matching of various schemas from a large number of context sources that are registered to the middleware in real time. To our best knowledge, all prior approaches to schema matching in traditional databases have not been presented or evaluated in an online manner using tens of schemas. Some of them involve processing that is unnecessary or inapplicable to our scenario of context-aware computing. Therefore, their performance in our scenario is unknown. We are identifying a few traditional database matchers that may be applicable in our middleware. An extensive performance comparison between these matchers and our matcher is an important direction of our future work.

There have been many publications on ontology matching in the research community [14,15]. Using our current context schema matcher as a starting point, we are studying how to extend our work to develop a more generic and complex ontology matcher in context-aware computing. The goal is to design a real-time ontology matcher that ensures both good accuracy and small computation cost despite the more versatile information encapsulated in the ontology model. Our CAM algorithm can be used as a sub-component of the ontology matcher to check the name similarities of classes, instances, properties and relationships in two ontologies. The shared attribute dictionary used in our CSM can be extended to represent other types of ontology elements.

CONCLUSION

We propose a name-based schema matcher for context-aware computing in this paper. We designed a CAM algorithm to match individual attributes in a local schema with those in a set of global schemas. The algorithm employs multiple matching criteria sorted and examined by a decreasing order of their priorities. The priorities and parameters of the criteria are dynamically adjusted based on the recent matching accuracy, which makes the algorithm adaptive to the current patterns of schema inputs. We further designed a CSM algorithm that uses the CAM outputs to integrate a local schema into one of the global schemas based on the largest common subset of matched attributes. The algorithm uses a shared attribute dictionary in the matcher that contains all attributes in the global schemas.

We have conducted an initial performance evaluation of our matcher using schemas obtained from real-world websites. The results demonstrate the good accuracy of our proposed CAM and CSM algorithms. Specifically, we have shown in our experiments that attribute matching based on equality, stemming and substring detection over their names are quite accurate in practice. Synonym detection has both strengths and weaknesses and must be put into the particular domain context of the schemas for a more subtle consideration.

Our ongoing and future work includes collecting large-scale datasets to further evaluate the performance of our schema matcher and compare it with traditional database matchers, extending our matcher to ontology matching and exploring the effects of using domain-specific LSA dictionaries rather than a general word directory for synonym detection.

ACKNOWLEDGMENTS

This work is funded by the Science and Engineering Research Council of Singapore under the research grant SERC 0521210083. We thank our shepherd, Tim Kindberg, for his guidance and helpful suggestions on improving this paper. We also thank the anonymous reviewers for their constructive comments.

REFERENCES

1. Chen, G. and Kotz, D. A Survey of Context-Aware Mobile Computing Research. *Technical Report TR2000-381*, Dartmouth College, 2000.
2. Dey, A.K. Understanding and Using Context. *Personal Ubiquitous Computing 5*, 1 (2001), 4-7.
3. Doan, A.H., Domingos, P. and Halevy, A. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In *Proc. SIGMOD 2001*, ACM Press (2001), 509-520.
4. Gu, T., Pung, H.K. and Zhang, D.Q. A Service-Oriented Middleware for Building Context-Aware Services. *Journal of Network Computer and Applications 28*, 1 (2005), 1-18.
5. Haghighi, P.D., Zaslavsky, A. and Krishnaswamy, S. An Evaluation of Query Languages for Context-Aware Computing. In *Proc. DEXA 2006*, Springer-Verlag (2006), 455-462.
6. Judd, G. and Steenkiste, P. Providing Contextual Information to Pervasive Computing Applications. In *Proc. Percom 2003*, IEEE Computer Society (2003), 133.
7. Kjær, K.E. A Survey of Context-Aware Middleware. In *Proc. Software Engineering 2007*, ACTA Press (2007), 148-155.
8. Landauer, T., Foltz, P.W. and Laham, D. Introduction to Latent Semantic Analysis. *Discourse Processes 25* (1998), 259-284.
9. Li, W.S. and Clifton, C. SemInt: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Network. *Data Knowledge Engineering 33*, 1 (2000), 49-84.
10. Longest Common Substring Problem. http://en.wikipedia.org/wiki/Longest_common_substring_problem.
11. Madden, S., Franklin, M.J., Hellerstein, J.M. and Hong, W. TinyDB: an Acquisitional Query Processing System for Sensor Networks. *ACM Transactions on Database Systems 30*, 1 (2005), 122-173.
12. Madhavan, J., Bernstein, P.A. and Rahm, E. Generic Schema Matching with Cupid. In *Proc. VLDB 2001*, VLDB Proceedings (2001), 49-58.
13. Ng, W.S., Ooi, B.C., Tan, K.L. and Zhou, A. PeerDB: A P2P-based System for Distributed Data Sharing. In *Proc. SIGMOD 2003*, ACM Press (2003), 659.
14. Noy, N.F. Semantic Integration: A Survey of Ontology-Based Approaches. *SIGMOD Record 33*, 4 (2004), 65-70.
15. Ontology Matching. http://www.ontologymatching.org/index.html.
16. Porter Stemming Algorithm. http://tartarus.org/~martin/PorterStemmer/.
17. Rahm, E. and Bernstein, P.A. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal 10*, 4 (2001), 334-350.
18. Ramakrishnan R. and Gehrke J. *Database Management Systems*. McGraw-Hill, Columbus, OH, USA, 2002.
19. Stop Word List. http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words.
20. WordNet. http://wordnet.princeton.edu/.
21. Yao, Y. and Gehrke, J. The Cougar Approach to In-Network Query Processing in Sensor Networks. *SIGMOD Record 31*, 3 (2002), 9-18.