An Intelligent Context Interpreter based on XML Schema Mapping

Been-Chian Chien  
Dept. of Computer Science and Information Engineering  
National University of Tainan,  
Tainan, Taiwan, R. O. C.  
e-mail: bcchien@mail.nutn.edu.tw

Shiang-Yi He  
Dept. of Computer Science and Information Engineering  
National University of Tainan,  
Tainan, Taiwan, R. O. C.  
e-mail: tkuchris@hotmail.com

Abstract—Context-aware computing is one of the attractive research topics in pervasive computing. Context-aware systems can react to users’ preferences according to context including location, time and other environment conditions. Context is generated by context interpreters or aggregated by context aggregators from the signals of sensors. A traditional context interpreter is usually built as an executable hard code called widget. It is difficult for the system manager to construct and maintain the large collections of context. In this paper, we propose a intelligent generic context interpreter using context scripts to overcome the hard code dependency between context and hardware devices. The generic context interpreter imports sensor data from sensor devices as an XML schema. Then, the schema matching approach is used to help system manager generating context scripts instead of widgets easily. The system was built and evaluated by different sensor schemas. The results show that the schema matching algorithm can match correct sensor types effectively and provide efficient context generation and maintenance.

Keywords—context-aware system; context interpreter; schema matching; intelligent system

I. INTRODUCTION

A context-aware system is a mobile environment in which applications can discover and make use of context information including user location, time, date, nearby devices and other environmental activities to adapt their operations and behavior [5]. A number of context-aware architectures were proposed and employed for a wide spectrum of systems and applications [3]. However, since each individual system focuses on its specific application domain, current context-aware systems are heterogeneous in all aspects, such as hardware, mobile resources, operating systems, application software, and platforms [15]. The serious heterogeneous characteristics of context-aware computing are especially important and become significant drawbacks while developing context interpreters for building context-aware services [4][14].

In general, context-aware systems use context to archive the objectivity of controlling services. Context is generated by context interpreters or aggregated by context aggregators from the signals of sensor devices in the mobile environment. A traditional context interpreter is usually built as a type of execution codes called widget [8]. Since a widget is usually designed by system programmers as a hard code to translate sensors data into a semantic representation called context, it is dependent upon the sensor devices and the application domains of context-aware systems. It is hard to maintain if some sensors were upgraded or renewed, and it is difficult for the system manager to construct new services for new devices. The extendibility of the systems thus will be restricted and lack of flexibility.

In this paper, we propose a intelligent generic context interpreter based on XML context scripts generator and XML schema matching schemes. The proposed generic context interpreter architecture consists of two modules: the context script generator and the generic script interpreter. We use context scripts to replace hard code widgets for solving the dependency problem between context and hardware devices. The context script generator imports sensor data from sensor devices as an XML schema. Then, the schema matching scheme is applied to help the system manager generating context scripts instead of widgets easily. The generic script interpreter can translate various context scripts into the corresponding contexts used in the application. The generic interpreter was implemented and evaluated by various sensors schemas. The results show that the intelligent context script generator can effectively recognize correct sensor types by the support of schema matching scheme and the context interpreter provides efficient context generation and maintenance.

This paper is organized as follows. In Section 2, we introduce the framework of context-aware systems. The generic context interpreter is described in Section 3. Section 4 presents the schema matching method for an intelligent context interpreter. The results of experiments and evaluation are shown in Section 5. Finally, conclusion is made in Section 6.

II. FRAMEWORK OF CONTEXT-AWARE SYSTEMS

The conceptual framework of context-aware systems consists of five layers as shown in Fig. 1 [4]. The contents of each layer are described as follows briefly.

![Figure 1 The five-layer conceptual framework.](image-url)
1) **The device layer**: This layer contains the operating physical devices used in the context-aware systems including sensors, identifiers, mobile devices, and actuators, etc.

2) **The interpretation layer**: This layer describes the semantic mapping between the device layer and the context layer containing context interpreter and context aggregator.
   - Context interpreter: The raw signals from sensors or mobile devices cannot work as their original format. They have to be transformed to context in context-aware system. The context interpreter is used to interpret the structures of raw data and represent the information as low-level context called sensor context.
   - Context aggregator: The context aggregator then gathers the related low-level sensor context data to form a higher-level context.

3) **The context layer**: Context processing is the core of a context-aware system. Context information is generated and managed in this layer. Context model is used to describe the interactive activities of the resource layer. Effective context extraction and efficient context management are the two main functions of managing context.

4) **The storage layer**: The storage layer stores not only the context data of the current status but also the historical context data in the context-aware system. The context data produced in the context layer are used to provide the services of applications in the application layer. To easily access context data, an effective context database is required. The context data access mechanism generally includes the storage of context schema and context query.

5) **The application layer**: In this layer, application can be defined and executed by querying the current status of context and the related historical context data from the context database in the storage layer. Since the contents of context-awareness are accessed by context queries from context databases, the various applications can be constructed under diverse applications.

III. **THE GENERIC CONTEXT INTERPRETER**

The architecture of the proposed generic context interpreter [4] is shown in Fig. 2. The main components consist of the context script generator and the generic script interpreter. The context script generator further contains three functions modules: the context mapping operators, the schema matching algorithm, and the schema mapping history. The components are explained as follows.

- **Sensor data schema**: Sensor schemas are provided by some techniques of connectivity standards, for example, UPnP and SOAP, which enable data transfer in XML-based procedure call. Each type of sensor delivers its sensor data by the predefined XML schema according to the hardware specification.
- **Context model**: The context model is built for different application environments. For mapping context schemas into sensor data schemas, ontology with XML-based representation is used for constructing the context model of the system.

![Figure 2. The architecture of generic context interpreter.](image)

IV. **THE SCHEMA MATCHING**

A. **Notation of Symbols**

In this paper, the schema matching scheme is important for finding a correct sensor type for context mapping. Without the automatic schema matching, users will miss the previous context interpretation of sensors and it further causes the problem of inconsistency on context interpretation. We apply several schema matching approaches to the system. The Similarity Yield Matcher (SYM) [7] is introduced here.
We first introduce and define the symbols used in the SYM approach.

\[ S \quad : \quad \text{Source schema}. \]
\[ s \quad : \quad \text{an element node in source schema} \, S. \]
\[ T \quad : \quad \text{Target schema}. \]
\[ t \quad : \quad \text{an element node in target schema} \, T. \]
\[ Lsim(s, t) \quad : \quad \text{the linguistic similarity between nodes} \, s \, \text{and} \, t. \]
\[ DTcom(s, t) \quad : \quad \text{the data type compatibility between} \, s \, \text{and} \, t. \]
\[ Sim(s, t) \quad : \quad \text{the structure similarity between rooted at} \, s \, \text{and} \, t. \]
\[ Wsim(s, t) \quad : \quad \text{the weighted similarity between rooted at} \, s \, \text{and} \, t. \]
\[ th_{high} \quad : \quad \text{the threshold for increasing} \, Wsim(s, t). \]
\[ th_{low} \quad : \quad \text{the threshold for decreasing} \, Wsim(s, t). \]
\[ th_{accept} \quad : \quad \text{the threshold for acceptance of valid mapping}. \]
\[ C_w \quad : \quad \text{the multiplicative factor for increasing} \, Wsim(s, t). \]
\[ C_{dec} \quad : \quad \text{the multiplicative factor for decreasing} \, Wsim(s, t). \]
\[ W_{struct} \quad : \quad \text{the weight of structure similarity for} \, Wsim(s, t). \]
\[ leaves(s) \quad : \quad \text{the set of leaves in the subtree rooted at} \, s. \]
\[ level(s) \quad : \quad \text{the depth of node} \, s, \text{the depth of root is} \, 1. \]
\[ StrongLink(s, t) \quad : \quad \text{the weight similarity} \, Wsim(s, t) \geq th_{accept}. \]

The main similarity measure of matching schemas in SYM is the weighted similarity \( Wsim \). For any two element nodes \( s \) and \( t \) belonging two schemas \( S \) and \( T \), respectively, the computation of \( Wsim(s, t) \) contains two phases: the linguistic matching and the schema structure matching. The goal of the linguistic matching phase is to find the linguistic similarity \( Lsim(s, t) \) between two element nodes \( s \) and \( t \). In the schema structure matching phase, we first compute the compatibility of data type \( DTcom(s, t) \) and the structure similarity \( Sim(s, t) \). Then, the weighted similarity \( Wsim(s, t) \) is measured by combining \( Lsim(s, t) \) with \( DTcom(s, t) \) or \( Sim(s, t) \). The detailed matching algorithm is described in the following subsections.

**B. Linguistic Matching**

The matching of element names in schemas is first step for most of the schema matching methods. A good name matcher can identify correct linguistic matching of element names effectively. An accurate name matching also helps to accomplish the element-level matching problem. However, a single name matcher with simple similarity measure cannot perform effective matching results in general. Here, we propose a linguistic matching method based on four name matchers: Levenshtein, 3-grams, Jaro-distance and WordNet.

The computation of linguistic similarity for Levenshtein, 3-grams, Jaro-distance are listed in [1]. For two strings \( s \) and \( t \), the similarity values between \( s \) and \( t \) are denoted as \( sim_{lev}(s, t) \), \( sim_{3-gram}(s, t) \), and \( sim_{jaro}(s, t) \), respectively.

WordNet was developed by Miller et al.[18]. The relations of words like, hyponyms, synonym and antonym are also computed by the levels of semantic hierarchy in the groups of words called synsets. Two words \( s \) and \( t \) are first stemmed. Then, the similarity of two words is computed by the depth of the different Lexicon’s hierarchies [13], as follows:

\[
\text{Distance} = \frac{\text{depth}_1 - \text{depth}_2 + \text{depth}_t - \text{depth}_s}{2}, \quad (1)
\]

\[
\text{sim}_{wordnet}(s, t) = 1 - \text{Distance}^2, \quad (2)
\]

where \( \text{depth} \) is the common parental depth, \( \text{depth}_s \) is the depth of \( s \) and \( \text{depth}_t \) is the depth of \( t \) from the root of the hierarchy.

Although we just introduce the four different name matchers, they are usually not used at the same time while computing the linguistic similarity \( Lsim(s, t) \) of element names \( s \) and \( t \). One of the reasons not using all name matchers is that these name matchers include implicit similarity hierarchy of each other. For instance, the 3-grams matcher has a high similarity when the Levenshtein matcher gets high similarity. Another reason is to reduce the computation time. Since the more matchers are operated, more computation cost is needed. Especially, WordNet spends a lot of time on searching dictionary for computing the similarity in their synsets. It is not a great idea to often execute WordNet. Hence, a decision tree of combining name matchers [10] is designed. The decision steps are listed as follows:

**Step 1.** For two input element names \( s \) and \( t \), we check whether the two strings are the same or not. If they are identical, the linguistic similarity is set to be one and the processing is halt; otherwise, the above name matchers are used to compute the linguistic similarity as the next step.

**Step 2.** The \( \text{sim}_{lev}(s, t) \) are first computed and tested. Then, one of the following three cases will happen:

1) If the value \( \text{sim}_{lev}(s, t) \) is larger than 0.55, the linguistic similarity is set to the value \( \text{sim}_{lev}(s, t) \).

2) If the value \( \text{sim}_{lev}(s, t) \) is less than 0.25, \( Wsim \) is used and the linguistic similarity is set to the value \( \text{sim}_{wordnet}(s, t) \).

3) If the value \( \text{sim}_{lev}(s, t) \) is between 0.25 and 0.5, the 3-gram and Jaro-distance matchers are used as Step 3.

**Step 3.** The average value of \( \text{sim}_{lev}(s, t) \) and \( \text{sim}_{wordnet}(s, t) \) are computed. If the average of the two matchers is larger than 0.15, the linguistic similarity is set to the average; otherwise, WordNet is finally used and the linguistic similarity is set to the value \( \text{sim}_{wordnet}(s, t) \).

Since WordNet matcher is time-consuming, it will not be started until no proper linguistic similarity is produced by the other matchers.

**C. Schema Structure Matching**

The goal of the schema structure matching phase is to find the weighted similarity \( Wsim(s, t) \) between two nodes \( s \) and \( t \) belonging to schema \( S \) and \( T \), respectively. The weighted similarity is calculated by a combination of the linguistic similarity \( Lsim \) and the structure similarity \( Sim \).

Without loss generality, we suppose that a node in a schema would be an internal node or a leaf node. Hence, the structure matching in schemas may perform on two leaf nodes, two internal nodes and one leaf node vs. one internal node. We regard matching two leaf nodes as the leaf-structure similarity matching. By contrast, the nonleaf-structure matching includes the cases of matching two internal nodes and matching one leaf with one internal node.
The leaf-structure similarity matching
Since the nodes \( s \) and \( t \) are leaves (element names) in this case, there is no tree structure on \( s \) and \( t \). The structure similarity, \( S_{sim}(s, t) \), considers the data types of the element names as the measure of their structures.

We make use of the type definitions on W3C Schema [15] and the data type conversion of W3C XQuery [16] to construct a data type compatibility table for every data type. In the convertibility table, the data type of the node \( s \) cannot be converted into the data type of the node \( t \), the data type compatibility between \( s \) and \( t \), \( DTCom(s, t) \), is set to be 0.1. If the data types of the nodes \( s \) and \( t \) can be converted each other, the value of \( DTCom(s, t) \) is 0.5. If the data type conversion of the nodes \( s \) and \( t \) depends on the source values, \( DTCom(s, t) \) is set to be 0.3.

In this case, since the structure similarity \( S_{sim}(s, t) \) is measured by the data type compatibility \( DTCom(s, t) \), the weighted similarity of the leaf-structure similarity matching is defined as

\[
W_{sim}(s, t) = 0.5 \times DTCom(s, t) + 0.5 \times L_{sim}(s, t). \tag{3}
\]

The nonleaf-structure similarity matching
If one of the matching nodes \( s \) and \( t \) is an internal node or both of them are internal nodes, the tree structures rooted at \( s \) and \( t \) need to be further considered. The structure similarity of the two nodes \( s \) and \( t \) is measured by the weighted similarity of nodes in the sets of leaves(\( s \)) and leaves(\( t \)) rooted at \( s \) and \( t \), respectively. The structure similarity, \( S_{sim}(s, t) \), in the nonleaf-structure similarity matching is defined as

\[
S_{sim}(s, t) = \frac{|\text{leaves}(s) \cup \text{leaves}(t)|}{\sum_{i=1}^{n}\text{leaves}(s)} \sum_{i=1}^{m}\text{leaves}(t) W_{sim}(x_i, y_j) + \sum_{j=1}^{n}\text{leaves}(t) W_{sim}(y'_j, x'_i), \tag{4}
\]

where

\[
(x_i, y_j) \in \{(x_i, y_j) \mid \text{for } x_i \in \text{leaves}(s), \exists y_j \in \text{leaves}(t) \}\text{satisfying max(StrongLink}(x_i, y_j))
\]

and \( \text{StrongLink}(s, t) \) is the matching pair that the weighted similarity \( W_{sim}(s, t) \) is larger than the threshold \( th_{accept} \).

The weighted similarity of nonleaf-structure similarity matching is defined to be the combination of the structure similarity \( S_{sim}(s, t) \) and the linguistic similarity \( L_{sim}(s, t) \), as follows:

\[
W_{sim}(s, t) = W_{struct} \times S_{sim}(s, t) + (1 - W_{struct}) \times L_{sim}(s, t), \tag{5}
\]

where \( 0 \leq W_{struct} \leq 1 \).

The algorithm of matching two schema structures starts tree matching process from the element names (leaf nodes) of the schemas. The value of \( W_{sim}(s, t) \) in the leaf-structure similarity matching is first computed by equation (3) for all leaf nodes of the source schema and the target schema. Then, the values of \( W_{sim}(s, t) \) for the nonleaf nodes at the upper level are computed by \( \text{StrongLink}(\text{leaves}(s), \text{leaves}(t)) \) of the leaves rooted at \( s \) and \( t \). The larger value of \( W_{sim}(s, t) \) is performed, the stronger structure similarity of the nodes \( s \) and \( t \) are resulted. On the contrary, if the value of \( W_{sim}(s, t) \) is small, the structure similarity of the nodes \( s \) and \( t \) is weak. Since the values of \( W_{sim}(s, t) \) for the nodes at upper levels of schemas are still computed by the leaves of subtrees rooted at \( s \) and \( t \), we have to make use of adjusting the values of \( W_{sim} \) on the leaf nodes to reflect the current structure similarity of nonleaf nodes.

Two criteria, \( th_{high} \) and \( th_{low} \), are used to strengthen and weaken, respectively, the values of weighted similarity on leaf nodes. If the value of \( W_{sim}(s, t) \) for two nonleaf nodes is larger than \( th_{high} \), all of the values of weighted similarity \( W_{sim}(\text{leaves}(s)), \text{leaves}(t)) \) are increased by multiplying the multiplicative factor \( C_{inc} \). On the other hand, when the value of \( W_{sim}(s, t) \) for two nonleaf nodes is less than \( th_{low} \), all of the values of weighted similarity \( W_{sim}(\text{leaves}(s)), \text{leaves}(t)) \) are decreased by multiplying the multiplicative factor \( C_{dec} \). The multiplicative factors \( C_{inc} \) and \( C_{dec} \) are set as follows:

\[
C_{inc} = 1 + \frac{1}{3 \times [\text{level}(s) + \text{level}(t)]}, \quad s, t \in T; \tag{6}
\]

\[
C_{dec} = 1 - \frac{1}{2 \times [\text{level}(s) + \text{level}(t)]}, \quad s, t \in T. \tag{7}
\]

The detailed schema tree matching algorithm is listed as in Fig. 3. The postorder sequence is used for matching nodes from the leaves to the upper levels.

1. Sub Schema Tree Match(SourceTree S, TargetTree T)
2. \( S' = \text{post-order}(S), T' = \text{post-order}(T) \)
3. for each \( s \) in \( S' \)
4. for each \( t \) in \( T' \)
5. if (\( s, t \) are leaves) then
6. set \( DTCom(s, t) = \text{Datatype-Compatibility}(s, t) \)
7. \( W_{sim}(s, t) = 0.5 \times DTCom(s, t) + 0.5 \times L_{sim}(s, t) \)
8. end if
9. if (\( s, t \) are non-leaf nodes) then
10. compute \( S_{sim}(s, t) = \text{structural-similarity}(s, t) \)
11. \( W_{sim}(s, t) = W_{struct} \times S_{sim}(s, t) + (1 - W_{struct}) \times L_{sim}(s, t) \)
12. end if
13. if (\( W_{sim}(s, t) \geq th_{high} \)) then
14. increase-weighted-similarity(\text{leaves}(s), \text{leaves}(t), C_{inc})
15. end if
16. if (\( W_{sim}(s, t) \leq th_{low} \)) then
17. decrease-weighted-similarity(\text{leaves}(s), \text{leaves}(t), C_{dec})
18. end if
19. end for
20. end for
21. End Sub

Figure 3. The schema tree matching algorithm.
D. Schema Similarity

After the computation of all values of $W_{sim}(s, t)$ for the source schema $S$ and the target schema $T$, the match pairs of the two schemas are generated by the $W_{sim}(s, t)$ of the leaf nodes $s$ in $S$ and $t$ in $T$. The match pairs generating algorithm is described as follows:

Step 1. Consider the matching from $S$ to $T$. For each leaf node $s$ in $S$, the matched node $t$ in $T$ must satisfy the conditions of maximizing $W_{sim}(s, t)$ and $W_{sim}(s, t) \geq t_{\text{accept}}$. Let the set of selected match pairs be $S_T$.

Step 2. Consider the matching from $T$ to $S$. For each leaf node $t$ in $T$, the matched node $s$ in $S$ must also satisfy the conditions of maximizing $W_{sim}(s, t)$ and $W_{sim}(s, t) \geq t_{\text{accept}}$. Let the set of selected match pairs be $T_S$.

Step 3. The final match results are the match pairs in the set $S_T \cap T_S$.

After generating the matching pairs, the schema similarity between two schemas $S$ and $T$ is defined as:

$$S_{S}S_{S}(S, T) = \frac{\sum_{(s, t) \in \text{match}} W_{sim}(s, t)}{|\text{leaves}(S)|}.$$  \hspace{1cm} (8)

V. EXPERIMENTAL RESULTS

The generic context interpreter proposed in this paper used Mapforce API to develop the context mapping in the context mapping editor. The initial blank system needs build context mapping manually. Once more mapping datasets were accumulated in the mapping history, Users will be able to refer to the existing schema mapping cases and a similar sensor schema mapping was selected to modify as a new context mapping.

The test schema sets include seven different sensor schemas listed in Table I. The depths of schema structures are four levels. The number of leaves is between the range four and six. The number of nodes is in the range of seven to ten. We first ranked the similarity degree of each schema by experts as shown in Table II. Then the proposed schema matching algorithms, Cupid [11] and COMA++ [2][8] are tested on each schema. To evaluate the schema matching performance of ranking, we refer to $R_{\text{norm}}$ [11] values as the criterion of effectiveness. The matching results of similarity are evaluated and ranked, as shown in Table III.

<table>
<thead>
<tr>
<th>TABLE I. THE STRUCTURE INFORMATION OF SENSOR SCHEMAS.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>GPSData (GPS)</td>
</tr>
<tr>
<td>HumidityData (Humid)</td>
</tr>
<tr>
<td>IRData (IR)</td>
</tr>
<tr>
<td>LightData (Light)</td>
</tr>
<tr>
<td>RFIDData (RFID)</td>
</tr>
<tr>
<td>SensorData (Sensor)</td>
</tr>
<tr>
<td>Temp2Data (Temp2)</td>
</tr>
</tbody>
</table>

The two methods SYM and SYM-Dict represents the proposed schema matching algorithms. The difference between them is that SYM-Dict used the full algorithm of the four linguistic matchers, whereas SYM sets the $L_{sim}(s, t)$ to be 0.4 instead of using WordNet searching and similarity computing.

The experimental results show that the SYM leads in four schemas and COMA++ is generally superior to others in three schemas. The $R_{\text{norm}}$ values of COMA++ are better than Cupid for 5 schemas except $S_7$:RFID and $S_5$:Sensor. The reason is that the type of value(xs:decimal) in $S_7$:Temp matched the type of value(xs:string) in $S_5$ and $S_5$. This mistake causes the higher rank of $S_7$:Temp. It shows that COMA++ is relatively weak in the matching of types on leaves. The SYM-Dict is surprisingly the worst in the four methods. The main reason is that the matching of WordNet did not perform proper linguistic similarity matching results. The WordNet may give relative high similarity for two different terms. On the contrary, the SYM directly sets the values as 0.4 is a good similarity since the linguistic similarity is generally less than 0.5 if the three linguistic matchers are not used. The average $R_{\text{norm}}$ values show that the SYM has the best ranking. The COMA++ combines different matchers and gains the second place. Generally speaking, SYM-Dict and Cupid is not recommended to be used in this application.

VI. CONCLUSION

The main contribution of this paper is to propose an intelligent context interpreter to get context independence. A context interpretation script is proposed to replace hard code-based context interpreter. We design a generic context interpreter consisting of the context script generator and the generic script interpreter. We also design a context editing tool for support the context mapping operation and devices maintenance. By employing schema matching approaches, the generic context interpreter performs a more intelligent operating interface for users. The heterogeneity in pervasive context-aware computing will gain a graceful solution.

The proposed schema matching methods also show their effectiveness on the matching rank. The SYM method is superior to SYM-Dict, Cupid and COMA++.

The heterogeneity is a series problem while developing and extending context-aware applications in an environment of pervasive computing. This work is intended as a starting point of further investigating on context-aware computing. The problems of context management for context-aware computing will be paid more attention in the future.

ACKNOWLEDGMENT

This research was supported in part by the National Science Council of Taiwan, R.O.C. under contract NSC 98-2221-E-024-012.

REFERENCES


---

**TABLE II. THE RANKING RESULT OF EXPERTS.**

<table>
<thead>
<tr>
<th>Experts</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rank</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Humid</td>
<td>GPS</td>
<td>Light</td>
<td>Humid</td>
<td>GPS</td>
<td>GPS</td>
<td>Light</td>
</tr>
<tr>
<td>3</td>
<td>Light</td>
<td>S3</td>
<td>S3</td>
<td>S3</td>
<td>Humid</td>
<td>S3</td>
<td>S3</td>
</tr>
</tbody>
</table>
| 4       | IR | S3 | IR | S3 | GPS | S3 | S3 | S3 | Light | Temp2 | S7 | Humid
| 5       | Sensor | Sensor | Temp2 | S7 | Temp2 | S7 | IR | S3 | Light | S3 | IR |
| 6       | Temp2 | S3 | Temp2 | S3 | RFID | S3 | RFID | S3 | Sensor | IR | S3 | Sensor |
| 7       | RFID | S3 | RFID | S3 | Sensor | S3 | Temp2 | S3 | RFID | S3 | Temp2 | S3 | RFID |

**TABLE III. THE PERFORMANCE COMPARISON OF SCHEMA MATCHING.**

<table>
<thead>
<tr>
<th>Methods</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>Temp2</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYM</td>
<td>0.905</td>
<td>0.905</td>
<td>1.000</td>
<td>0.952</td>
<td>1.000</td>
<td>1000</td>
<td>0.857</td>
<td>0.946</td>
<td></td>
</tr>
<tr>
<td>SYM-Dict</td>
<td>0.714</td>
<td>0.857</td>
<td>0.857</td>
<td>0.762</td>
<td>0.905</td>
<td>0.875</td>
<td>0.762</td>
<td>0.819</td>
<td></td>
</tr>
<tr>
<td>Cupid</td>
<td>0.667</td>
<td>0.905</td>
<td>0.801</td>
<td>0.801</td>
<td>0.905</td>
<td>0.952</td>
<td>0.762</td>
<td>0.828</td>
<td></td>
</tr>
<tr>
<td>COMA++</td>
<td>1.000</td>
<td>1.000</td>
<td>0.857</td>
<td>0.857</td>
<td>0.81</td>
<td>0.905</td>
<td>0.952</td>
<td>0.912</td>
<td></td>
</tr>
</tbody>
</table>