A Generic Context Interpreter for Pervasive Context-aware Systems

Been-Chian Chien and Shiang-Yi He
Department of Computer Science and Information Engineering
National University of Tainan, Tainan, Taiwan, R. O. C

ABSTRACT
Developing pervasive context-aware systems to construct smart space applications has attracted much attention from researchers in recent decade. Although many different kinds of context-aware computing paradigms were built of late years, it is still a challenge for researchers to extend an existing system to different application domains and interoperate with other service systems due to the problem of heterogeneity among systems. In this paper, we propose a generic context interpreter to overcome the dependency between context and hardware devices. The proposed generic context interpreter mainly contains two modules: the context interpreter generator and the generic interpreter. The context interpreter generator imports sensor data from sensor devices as an XML schema and produces interpretation scripts instead of interpretation widgets. Then, the generic interpreter generates the semantic context for context-aware applications. A context editor is also designed by employing schema matching algorithms for supporting context mapping between devices and context model.

Keywords: Context-aware computing, pervasive computing, generic context interpreter, XML schema matching.

INTRODUCTION
With the rapid growth of wireless sensors and mobile devices, the research of pervasive computing is becoming important and popular in recent years. The vision of pervasive computing [13] is to integrate hardware, network systems, and information technologies to provide appropriate service for our lives in a vanishing way. A context-aware system is a pervasive computing environment in which users’ preference services can be detected by making use of context including location, time, date, nearby devices and other environmental activities to adapt users’ operations and behavior [6]. All kinds of context-aware architectures and frameworks have been designed and employed for a wide spectrum of applications [3]. Since most of the systems focus on theirs specific application domains; the current context-aware systems are heterogeneous in all aspects, such as hardware, mobile resources, operating systems, application software, and platforms. The serious heterogeneous characteristics of context-aware computing are especially important and become significant drawbacks while developing or integrating context-aware applications in pervasive computing environments.

The concept of context independence was revealed in CADBA architecture [7]. Two types of context independence, the physical context independence and the logical context independence, are classified in the article. The physical context independence is to prevent misinterpreting raw
data from sensors with various specification standards; whereas the logical context independence is to allow context to be understood and applied by applications. As a result of context independence, cross-domain service applications will be able to be integrated into a unified context-aware system regardless of the heterogeneity in pervasive environment.

An OSGi-based service platform [9] based on Java VM is one of the practical solutions for accomplishing logical context independence. In this paper, a generic context interpreter is proposed to overcome the physical context dependence problem between context and sensor devices in the framework of context-aware computing. The context generic interpreter is composed of two modules: the context interpreter generator and the generic interpreter. First, the context interpreter generator imports sensor data from sensor devices as an XML schema and produces interpretation scripts instead of interpretation widgets. Then, the generic interpreter generates the semantic context for context-aware applications. An interface tool, the context editor, is also designed by employing automatic XML schema matching schemes for supporting smart context mapping between devices and context model.

The remainder of this paper is organized as follows. Section 2 introduces the foundation and summary of context-aware architectures. The architecture of proposed generic context interpreter is presented in Section 3. The detailed design of system components is described in Section 4. The performance evaluation of context mapping for proposed generic context interpreter is demonstrated and discussed in Section 5. Section 6 concludes the work and expresses the future work.

PRELIMINARIES

The term context-aware first appeared in 1994 mentioned by Schilit and Theimer [17]. Since then, various context-aware computing architectures were proposed: The Context Toolkit [8] provided context interpretation using widgets and a set of object-oriented APIs to offer the creation of service components. The Hydrogen [12] is a framework based on layered architecture in which contains the adaptor layer, the management layer and the application layer. The Gaia project [16] is a middle-ware based architecture; the system consists of Gaia kernel and application framework to support the development and execution of mobile applications. Another middle-ware system, SOCAM [9], uses a central server called context interpreter to obtain context data for building and prototyping of context-aware services. The CORTEX system [5] is also a middle-ware structure based on sentient object model which supports context-aware services in an ad-hoc mobile environment.

The above context-aware system architectures generally follow the framework presented in [1] and [3]. The five-layer model in [1] consists of the physical layer, the data layer, the semantic layer, the inference layer and the application layers. These layers in this model focus on the descriptions of functions in a context-aware system. The abstract five-layer proposed in [3] described a conceptual framework of a context-aware system containing the sensors layer, the raw data layer, the preprocessing layer, the storage/management layer, and the application layers.

A context-aware system was utilized on its individual specific application domain of employment whatever the framework or architecture it used. Thus, the problem of heterogeneity issues was generally characterized not only on the physical devices but also on the logical context interoperability. To bridging the communication between heterogeneous devices, some
researches on adapting and integrating systems sprout recently. A OSGi-based infrastructure [9] was proposed to adapt to changing context. In [4], Bartelt et al. proposed a system that integrates devices dynamically and enable interoperability between them. Nakazawa et al. [15] presented a framework for heterogeneity handling. Schmohl and Baumgarten [18] further derived a generalized context-aware architecture for pervasive computing environments to resolve the heterogeneity issues in both context-awareness and interoperability domains.

This work is based on another generalized context-aware architecture CADBA [7]. The architecture of CADBA is also structured by a five-layer framework consisting of the device layer, the interpretation layer, the context layer, the storage layer, and the application layer, as shown in Figure 1.

The main modules and components for each layer are described as follows.

1) **The device layer:**
   This layer contains the physical equipments and devices operated and used in the context-aware systems including sensors, identifiers, mobile devices, and actuators, etc. The possible hardware and the drivers used in the domain that could be mobile devices like PDAs, smart cell phones, detecting sensors like thermometers, RFIDs, or actuators like alarms, temperature regulators.

2) **The interpretation layer:**
   A semantic mapping layer between the device layer and the context layer is needed. The main components in this layer contain context interpreters and context aggregators.

![Figure 1. The CADBA architecture for five-layer framework.](image)
• **Context interpreter**: The raw signals from sensors or mobile devices cannot work as their original format. They have to be transformed to context information in context-aware system. The context interpreter is used to interpret the structures of raw data from sensors 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. As soon as the context model is provided, context can be generated by context interpreters and context aggregators.

3) **The context layer**:
Context processing is the core of a context-aware system. A context model and efficient context management are the main two components.

• **Context model**: The *context model* is essential for a context-aware system to progress the context processing. The CADBA uses ontology to represent the context model with OWL.

• **Context management**: Context-aware computing implies the interconnection of people, devices, and environments. It makes possibility of sharing the users’ context among each other in heterogeneous systems. The issues and challenges of context management include context configuration, device fault, context conflict, context inconsistency and security [11].

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.

• **Domain knowledge**: It contains the entire resources of the context-aware computing including places, persons, devices, and objects.
  Places: All possible locations where activities will occur in the environment of applications.
  Persons: The possible people including the application users and other persons who interact in the system.
  Devices: The description for all existing devices that support the device manager to record the status of the hardware.

• **Context database**: The context database is to provide the integration of current context and the storage of historical context data. The main function of the context database is to provide an efficient context access mechanism to store and retrieve context data using context queries.

5) **The application layer**:
In this layer, application service can be built and executed by the service provider through querying the current status of context and the related historical context data.

• **Service provider**: The functions of the service provider consist of context querying, service construction, and service coordination. To satisfy the user’s need and complete the application, the service provider starts context process by context querying. Then, the returning context data from the context database are collected to determine and prepare the necessary application services. Finally, the service coordinator is used to arrange and
perform the application services. The logical context independence can be accomplished
by applying context queries to create application services.

THE ARCHITECTURE OF GENERIC CONTEXT INTERPRETER

Two types of context independence are revealed in the previous section: the physical context
independence and the logical context independence. The main goal in this paper is to achieve the
physical context independence using the proposed generic context interpreter. The physical
context independence is defined as the immunity of context from the changing or updating of
physical sensor devices. A traditional context interpreter is usually dedicated to interpreting the
context designated for a specific device in an application. Although such a tightly coupled
structure results in good performance of fast reaction for a system, its drawback is that the
application does not work and the service cannot be extended to other devices without
completely re-packing function program of the context interpreter after changing or equipping a
new sensor device. The physical context independence tries to maintain normal operations of the
context-aware system regardless of changing, reconstructing, or replacing sensor devices
anytime.

Before constructing and performing context interpreter, the context model is essential in a
context-aware system and the representation of context must be determined. For designing a
generic context interpreter, ontology with OWL representation is considered as the context
model for the proposed architecture. The techniques of ontology fusion can be applied to
integrate heterogeneous context models and the representation, OWL, is XML based scripts. It is
formatted, unified, and standardized. While importing data from sensor devices, the XML format
is also easy to convey the information extracted by data acquisition interface to the context
interpreter. The context used in the CADBA architecture is divided into three levels by means of
XML-based context model:

1) **Sensor context (Context\textsubscript{sensor}):**
   This type of context is the essential raw information triggered by sensors and interpreted by
context interpreters directly. The definition is as follows:
   \[
   \text{Context}_{\text{sensor}} = \text{Interpret}\{\text{sensor} \mid \text{sensor} \in D_j \text{ and } D_j \subseteq \text{Dev}\},
   \]
   where \text{sensor} is a kind of sensors in the type of devices \text{Dj}, and \text{Dev} is the set of all types of
devices in the context aware system.

2) **Event context (Context\textsubscript{event}):**
   This type of context is aggregated by different sensor context. We define the behavior of
   aggregating various actions of sensors as an event context.
   \[
   \text{Context}_{\text{event}} = \text{Aggregate}\left\{\bigcup_{j=1}^{m} \text{Context}_{\text{sensor}}\right\}.
   \]

3) **Scenario Context (Context\textsubscript{scenario}):**
   The scenario context is another high-lever context containing not only sensor context but also
   at least one event context. The scenario context is defined as follows:
\[ \text{Context}_{\text{scenario}} = \text{Aggregate}\{ \bigcup_{i=1}^{n} \text{Context}_{\text{event}} \} \cup \{ \bigcup_{i=1}^{m} \text{Context}_{\text{sensor}} \}, \]

where \( 1 \leq |\text{Context}_{\text{event}}| \leq n \) and \( 1 \leq |\text{Context}_{\text{sensor}}| \leq m \), or \( 2 \leq |\text{Context}_{\text{event}}| \leq n \) and \( 0 \leq |\text{Context}_{\text{sensor}}| \leq m \).

The architecture of the proposed generic context interpreter is shown in Figure 2. The main components are the context interpreter generator and the generic interpreter. The context interpreter generator further contains three functions: the mapping operators, the context editor and the schema matching algorithm. In this model, context interpreters generated by the interpreter generator are interpretation scripts instead of interpretation functions or programs. The interpretation script draws linking relationships and transforming methods between sensor data and the semantics in the context model. The generic interpreter then translates sensor raw data into context by the corresponding interpretation scripts while the context interpretation process being proceeded. We depict each component block as follows and the detailed design of function blocks will described in Section 4.

- **Sensor data model:**
  Sensor models can be 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. Such a schema will be sent to the context editor for linking relationship rules with the context model.

- **Context model:**
  The context model is built for different application environment. For mapping context schema into sensor data schema, ontology with XML-based representation is used here.

*Figure 2. The architecture of generic context interpreter.*
• **Context interpreter generator:**
  Interpreter generator produces interpretation scripts which contain the mapping relationships between context schema and sensor data schema. The mapping interface tool is the context editor. The context editor supports a graphical user interface to assist users to link the relationship between context schema and sensor data schema. The mapping job can be finished manually or automatically by applying schema matching schemes.

• **Interpretation script:**
  The interpretation script uses XSL (eXtensible Stylesheet Language) to describe the schema mapping. The content of the script is the rules of transforming source data schema (sensor data) into target data schema (context).

• **Generic interpreter:**
  Since interpretation scripts are stated in XSL, the XSLT (XSL Transformation) Processor can be used to be the generic interpreter directly. The XSLT-Processor will read the sensor raw data with XML tag and interpret the context according to the corresponding interpretation script.

  The interpretation flow consists two phrases: interpretation scripts generation and context interpretation. In the interpretation scripts generation phrase, mobile sensors prepared circumstances data schema in XML format and delivered them to the context interpreter generator. After sensor data schema was received by the context interpreter generator, the context editor is used to build the relationship between sensor data and sensor context according to the definitions in context model. An XSL script will be generated for interpreting the sensor context if the confirmation is done. Then, in the context interpretation phrase, the XSLT-Processor is used to translate original XML sensor raw data into XML sensor context defined in the context model. After completing the definitions of sensor context, event context and scenario context can be aggregated from sensor context by the context editor as the same way.

**FUNCTIONAL DESIGN OF CONTEXT INTERPRETER GENERATOR**

**MAPPING OPERATORS:**

The function of mapping operators is to transfer the sensor raw data into semantic context. Four types of mapping operators are defined in the system. The operators are designed by XSL descriptions. These operators thus can be easily reused and modified to a new one.

**Operator 1: concat**

The concatenation operator is used to merge two separated data into one. For instance: date-time or firstname-lastname. The rule of concatenation operator in XSL is shown as the following.

```
<AllName>
    <xsl:value-of select="concat(concat(string(LastName), ' '), string(FirstName))"/>
</AllName>
```

**Operator 2: value-map**
This operator maps the raw data with a item value into the semantic context described in context model. It states that the item value is equal to the context in the system. For example, if “RFID-tag:0X8001” represents the context “Person: John”. The definition of the operator in XSL description is shown as follows.

```xml
<xsl:template name="Value_map:Person">
    <xsl:param name="input" select="../.">
    <xsl:choose>
        <xsl:when test="$input='0X8001'">
            <xsl:value-of select="'John'"/>
        </xsl:when>
        <xsl:when test="$input='0X8002'">
            <xsl:value-of select="'Peter'"/>
        </xsl:when>
        <xsl:when test="$input='0X8003'">
            <xsl:value-of select="'George'"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:value-of select="'none'"/>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```

**Operator 3: Value-Semantic**

The purpose of this operator is to process numerical values from sensor raw data. For example, if temperature is lower than 16°C, the context “cold” will be interpreted and used in the system instead of the original numerical value. The XSL definition of the above context can be described as follows.

```xml
<xsl:template name="Value_Semantic:Temperature">
    <xsl:param name="input_value" select="../."/>
    <xsl:param name="lessThan" select="../."/>
    <xsl:param name="greaterThan" select="../."/>
    <xsl:param name="High_Semantic" select="../."/>
    <xsl:param name="Medium_Semantic" select="../."/>
    <xsl:param name="Low_Semantic" select="../."/>
    <xsl:choose>
        <xsl:when test="string($input_value &lt; $lessThan) != 'false'">
            <xsl:value-of select="$Low_Semantic"/>
        </xsl:when>
        <xsl:otherwise>
            <xsl:choose>
                <xsl:when test="string($input_value &gt; $greaterThan) != 'false'">
                    <xsl:value-of select="$High_Semantic"/>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:value-of select="$Medium_Semantic"/>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:otherwise>
    </xsl:choose>
</xsl:template>
```
Operator 4: Conversion-Function
This operation allows users to define formula for some specific requests. A practical example is
the transform between degree centigrade and Fahrenheit scale. The transform formula is usually
used to convert 20°C into 68°F.

```
<xsl:template name="Conversion:CelsiusToFahrenheit">
  <xsl:param name="CelsiusValue" select="/.."/>
  <xsl:value-of select="((($CelsiusValue * 9) div 5) + 32)"/>
</xsl:template>
```

SCHEMA MATCHING:
The main function of schema matching algorithm is to assist users to retrieve related context
from the historical mapping repository. Since the number of context is usually large in a context-
aware application, the management of context is an important work. One of the issues is to find
similar or even the same context mapping to develop context interpretation scripts. The
advantage of using XML to represent context here is that XML schema matching method can be
applied to search the sensor schema with high similarity. Making use of reusing the similar
schema of context mapping, the new context mapping will be built quickly. A good XML
schema matching algorithm can help users to construct intelligent context interpreter generator.

The XML schema and ontology matching problem is one of the research issues in data
manipulation. There are many researches and discussion on such a topic. Two famous schema
matching algorithm were applied in our work. The first is Cupid [14] proposed by Madhavan et
al. The other is COMA++ [2] by Aumuller. We will discuss the effectiveness and efficiency of
the two methods after applying to the context interpreter generator in Section 5.

CONTEXT EDITOR:
The context editor is designed for defining mapping operators, managing context mapping
between sensor data schema and context model, and generating context interpretation scripts. A
graphical user interface is provided to help users to finish the sensor binding efficiently and
easily. The context mapping operation in a context editor is demonstrated as Figure 3. First, the
editor reads the XML sensor data or others context from the context model, as shown in Figure
3(a). Second, the corresponding context in the context-aware system is selected from the context
model, as Figure 3(b) shows. The next step shown in Figure 3(c) is to arrange the interpreting
operations. Some predefined transformation operators and user defined operators can be added
into the system for transferring data values to meaningful context. At last, the finished context
mapping, as Figure 3(d), can be stored in XSL script to interpret the context in the context-aware
system. Owing to the context interpreter is done by interpretation scripts, the context interpreter
only builds a new context interpretation script instead of full-function program while mobile
devices are updated or changed. The goal of the physical context independence can be enforced.
SYSTEM CONSTRUCTION AND EVALUATION

The generic context interpreter proposed in this paper used Mapforce API to develop the context mapping in the context editor. The context editor provides both manual mapping and automatic mapping tools. 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 1. 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 2. Then the schema matching algorithms, Cupid and COMA++, are tested on each schema. The matching results of similarity are evaluated and ranked, as shown in Table 3. To evaluate the performance of ranking, we refer to $R_{norm}$ [10] values as the criterion of effectiveness. The estimation of running time is also shown in Table 4.
### Table 1. The test schema sets.

<table>
<thead>
<tr>
<th>Schema</th>
<th>number of leaves</th>
<th>number of nodes</th>
<th>tree depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$: GPSData (GPS)</td>
<td>5</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>$S_2$: HumidityData (Humid)</td>
<td>5</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>$S_3$: IRData (IR)</td>
<td>5</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>$S_4$: LightData (Light)</td>
<td>5</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>$S_5$: RFIDData (RFID)</td>
<td>5</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>$S_6$: SensorData (Sensor)</td>
<td>6</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>$S_7$: Temp2Data (Temp2)</td>
<td>4</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

### Table 2. The ranking results of Experts for the test sets.

<table>
<thead>
<tr>
<th>Experts ranking</th>
<th>$S_1$ GPS</th>
<th>$S_2$ Humid</th>
<th>$S_3$ IR</th>
<th>$S_4$ Light</th>
<th>$S_5$ RFID</th>
<th>$S_6$ Sensor</th>
<th>$S_7$ Temp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$S_2$ Humid</td>
<td>$S_3$ IR</td>
<td>$S_4$ Light</td>
<td>$S_5$ RFID</td>
<td>$S_6$ Sensor</td>
<td>$S_7$ Temp2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$S_1$ GPS</td>
<td>$S_2$ Humid</td>
<td>$S_3$ IR</td>
<td>$S_4$ Light</td>
<td>$S_5$ RFID</td>
<td>$S_6$ Sensor</td>
<td>$S_7$ Temp2</td>
</tr>
<tr>
<td>4</td>
<td>$S_1$ GPS</td>
<td>$S_2$ Humid</td>
<td>$S_3$ IR</td>
<td>$S_4$ Light</td>
<td>$S_5$ RFID</td>
<td>$S_6$ Sensor</td>
<td>$S_7$ Temp2</td>
</tr>
<tr>
<td>5</td>
<td>$S_1$ Sensor</td>
<td>$S_2$ Temp2</td>
<td>$S_3$ Temp2</td>
<td>$S_4$ Light</td>
<td>$S_5$ RFID</td>
<td>$S_6$ Sensor</td>
<td>$S_7$ Temp2</td>
</tr>
<tr>
<td>6</td>
<td>$S_1$ Sensor</td>
<td>$S_2$ Temp2</td>
<td>$S_3$ Temp2</td>
<td>$S_4$ RFID</td>
<td>$S_5$ Sensor</td>
<td>$S_7$ Temp2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>$S_1$ Sensor</td>
<td>$S_2$ Temp2</td>
<td>$S_3$ Temp2</td>
<td>$S_4$ RFID</td>
<td>$S_5$ Sensor</td>
<td>$S_7$ Temp2</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3. The evaluation results of the of schema matching algorithms.

<table>
<thead>
<tr>
<th>Schema ranking</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
<th>$S_6$</th>
<th>$S_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$S_6$</td>
<td>$S_4$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>3</td>
<td>$S_3$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
<td>$S_1$</td>
</tr>
<tr>
<td>4</td>
<td>$S_7$</td>
<td>$S_6$</td>
<td>$S_2$</td>
<td>$S_6$</td>
<td>$S_2$</td>
<td>$S_2$</td>
<td>$S_2$</td>
<td>$S_6$</td>
<td>$S_2$</td>
<td>$S_2$</td>
<td>$S_2$</td>
<td>$S_2$</td>
<td>$S_2$</td>
<td>$S_2$</td>
</tr>
<tr>
<td>5</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_6$</td>
<td>$S_3$</td>
<td>$S_4$</td>
<td>$S_4$</td>
<td>$S_4$</td>
<td>$S_6$</td>
<td>$S_3$</td>
<td>$S_4$</td>
<td>$S_4$</td>
<td>$S_4$</td>
<td>$S_4$</td>
<td>$S_4$</td>
</tr>
<tr>
<td>6</td>
<td>$S_4$</td>
<td>$S_7$</td>
<td>$S_7$</td>
<td>$S_6$</td>
<td>$S_3$</td>
<td>$S_3$</td>
<td>$S_3$</td>
<td>$S_7$</td>
<td>$S_6$</td>
<td>$S_3$</td>
<td>$S_3$</td>
<td>$S_3$</td>
<td>$S_3$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>7</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
<td>$S_5$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R_{norm}$</th>
<th>0.667</th>
<th>0.905</th>
<th>0.801</th>
<th>0.801</th>
<th>0.905</th>
<th>0.952</th>
<th>0.762</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.828</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.912</td>
</tr>
</tbody>
</table>

### Table 4. The estimated running time for Cupid and COMA++.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$ GPS</td>
<td>27.892</td>
</tr>
<tr>
<td>$S_2$ Humid</td>
<td>27.402</td>
</tr>
<tr>
<td>$S_3$ IR</td>
<td>36.010</td>
</tr>
<tr>
<td>$S_4$ Light</td>
<td>25.954</td>
</tr>
<tr>
<td>$S_5$ RFID</td>
<td>32.339</td>
</tr>
<tr>
<td>$S_6$ Sensor</td>
<td>33.072</td>
</tr>
<tr>
<td>$S_7$ Temp2</td>
<td>21.029</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>$S_1$ GPS</th>
<th>$S_2$ Humid</th>
<th>$S_3$ IR</th>
<th>$S_4$ Light</th>
<th>$S_5$ RFID</th>
<th>$S_6$ Sensor</th>
<th>$S_7$ Temp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cupid</td>
<td>27.892</td>
<td>27.402</td>
<td>36.010</td>
<td>25.954</td>
<td>32.339</td>
<td>33.072</td>
<td>21.029</td>
</tr>
<tr>
<td>COMA++</td>
<td>2.844</td>
<td>2.865</td>
<td>1.998</td>
<td>2.632</td>
<td>2.559</td>
<td>2.545</td>
<td>2.538</td>
</tr>
</tbody>
</table>
The experimental results show that COMA++ is generally superior to Cupid in both effectiveness and efficiency. The \( R_{\text{norm}} \) values of COMA++ are better than Cupid for 5 schemas except \( S_5: \text{RFID} \) and \( S_6: \text{Sensor} \). The reason is that the type of value(xs:decimal) in \( S_7: \text{Temp} \) matched the type of value(xs:string) in \( S_5 \) and \( S_6 \). This mistake causes the higher rank of \( S_7: \text{Temp} \). It shows that COMA++ is relatively weak in the matching of types on leaves.

**CONCLUSION**

The main contribution of this paper is to propose a generic context interpreter to accomplish physical context independence. This work is based on the context-aware architecture, CADBA. Ontology based context model is used in this architecture. We design a generic context interpreter including context interpreter generator and a generic interpreter. A context interpretation script is proposed to replace function-based context interpreter. As we known, this is the originality of context provider or interpretation in context-aware computing. We also design a context editing tool for support the context mapping operation and devices maintenance. By introducing mapping operators and schema matching schemes, 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 problem of heterogeneity is a bottleneck while developing and extending context-aware systems in pervasive computing environment. The enforcement of context independence resolves dependency of devices and improves interoperability of applications. This work is intended as a starting point of future research on context generation. The problems of context management for context-aware computing will be paid more attention in the future.

**ACKNOWLEDGMENTS**

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**