IEICE TRANS. INF. & SYST., VOL.E94–D, NO.1 JANUARY 2011

PAPER

A Reference Programming Model for Building Context-Aware Application

Junbin ZHANG[†a)], ***Member***, **Yong QI**[†b)], **Di HOU**[†c)], ***and*** **Ming LI**[†d)], ***Nonmembers***

SUMMARY Context-aware applications are a key aspect of pervasive computing. The core issue of context-aware application development is how to make the application behave suitably according to the changing context without coupling such context dependencies in the program. Several programming paradigms and languages have been proposed to facilitate the development, but they are either lack of sufficient flexibility or somewhat complex for programming and deploying. A reference programming model is proposed in this paper to make up inadequacy of those approaches. In the model, virtual tables constructed by system and maintained by space manager connect knowledge of both developer and space manager while separating dependency between context and application logic from base program. Hierarchy and architecture of the model are presented, and implementation suggestions are also discussed. Validation and evaluation show that the programming model is lightweight and easy to be implemented and deployed. Moreover, the model brings better flexibility for developing context-aware applications.

key words: *pervasive computing, context-aware, programming model, table-driven, language extension*

1. Introduction

Pervasive computing is being integrated into and is changing people's daily lives. As an important component of pervasive computing environment, context-aware [1] application can sense the physical environment where it runs, i.e., its context, and adapt its behavior accordingly while placing minimal intervention of users, which introduces a novel set of design challenges that are not present in traditional desktop programming.

Earlier researchers proposed preference model based branching model and triggering model [2]. Some programming languages and paradigms have also been put forward in recently years to facilitate development of the context-aware applications. In COP proposed in [3], [4], program is composed of many open terms. The open terms and associated contexts are defined in advance, and will be replaced dynamically at the run time with pieces of codes (called stubs) defined in the repository of candidates by context-filling operation. The stubs are also defined beforehand by system developer. COP separates context from application logic by matching open terms with stubs, and as a result, factors of environment are decoupled from application at design time. As an extension to the Common Lisp Object System, ContextL [5] associates partial class and method definitions with layers, and activates and deactivates such layers in the control flow of a running program. When a layer is activated, the partial definitions become part of the program until this layer is deactivated. ContextL separates class definitions into distinct layers instead of factoring out the behavior code into different classes, thus causes programs to adapt to context flexibly while keeping the conceptual simplicity of object-oriented programming. Multifaceted based programming paradigm [7] facilitates the development of the context-aware pervasive services with fully defined and deterministic behavior. The multifaceted programming entity, each facet of which denotes a pair of specific context and corresponding behavior, is the core concept of the paradigm. The context associated to a facet behaves like a switch, and the adaptation of the multifaceted programming entity to the context is done by switching facets. Developers choose proper exposing strategy to control the switching process. The process exposes and hides facets of entities and makes the application follow changing of the context automatically.

Although these approaches solve some key problems of pervasive computing programming, they are either lack of sufficient flexibility or somewhat complex for programming and deploying. Our research team proposes a table-driven programming paradigm [9] for developing context-aware application to address shortcomings mentioned above. The paradigm separates context awareness and reaction to it from the application logic using a table-driven programming style. The virtual tables produced by system and maintained by space manager isolate context-related operations from application while, at the same time, connecting knowledge of both developer and space manager. The paradigm isolates programming complexity and brings better flexibility. In this paper, we induce a reference programming model by extending and perfecting the paradigm. As a lightweight reference programming model, the model can be implemented in most existing programming language. The table matching strategy and its effect on solving potential conflict problems are analyzed. Suggestions for implementing the model are also discussed. A context operation simulation platform is designed to support validating the model. Validation and evaluation shows feasibility and flexibility of the model.

The remainder of the paper is structured as follows. Section 2 describes motivations of the research. Section 3 presents details of the model. Implementation of the model

Manuscript received May 14, 2010.
Manuscript revised September 25, 2010.

[†]The authors are with the Institute of Computer Software and Theory of Xi'an Jiaotong University. Xi'an, Shaanxi, 710049, China.
a) E-mail: henry@xjtu.edu.cn
b) E-mail: qiy@xjtu.edu.cn
c) E-mail: houdi@xjtu.edu.cn
d) E-mail: lm_bai@sina.com

DOI: 10.1587/transinf.E94.D.114

is discussed in Sect. 4. Section 5 validates and evaluates the model. Related works are described in Sect. 6 and Sect. 7 concludes.

2. Motivations

Nowadays, to efficiently lower developers' concern about the context related operations and enhance efficiency of the development, researchers commonly separate these complex operations from programming logic of context-aware application by an infrastructure-centered approach to gather, manage and delivery context information to the application that requires it. Upon the context operating infrastructure, common approaches that facilitate the development of context-aware application at design time involve reconstructing of certain programming language and changing programming style. These approaches provide an isolating mechanism that hide context related operations from application, which reduces burdens of development for programmer. Also the approaches establish a linkage between environment and application, thus enable the application to behave suitably according to the changing context. Developer defines responses for every possible context value in context-aware application. When the application runs, the infrastructure extracts context information from environment and dispatches it to proper context-based response. As a consequence of that, the application behaves accordingly.

However, we are facing a complex, dynamic and ever-changing pervasive computing environment. Despite advantages of these approaches, it is unrealistic and impossible for the developer to take into account all possible changes of run-time context at design time. Researchers have provided us several solutions in their programming model. When developing with COP, we can define more open terms and corresponding stubs to cope with new emerging context. We can also create new layers associated with appropriate partial class and method definitions, and make them be capable of being activated and deactivated when programming in ContextL. Similarly, new facets, each of which denotes a pair of specific context and corresponding behavior, can be added to the multifaceted programming item by using facet acquirement operation. Unfortunately, all these modifications of application should be accomplished at design time. In most cases, any such actions will cause the program to be recompiled and the running application to be started over.

Our research team focuses exactly on this issue. To overcome the obstacle, we consider that the manager of the pervasive computing space who knows detail of application requirements and is more familiar with the environment than programmer should take part in developing and deploying the context-aware application. We need a mechanism to connect knowledge of both developer and space manager and to enable the latter to adjust behavior of application manually at run time when needed without modifying its source code.

All of our ideas will be explained based on the following example scenario:

Mr. Smith presents in an international conference. He and the other attendees that come from all over the world carry hand-held terminal (maybe PDA, smart phone or other intelligent device) with them. Mr. Smith's personal information and preference, such as name, mother language and interested session of the conference are stored in his terminal devices. A register service greets him in his mother language when he steps into the conference hall. When he roams session rooms, an introduction service sends conference materials of current session automatically to his terminal device.

The scenario represents a general context-aware application. The application includes a register service and an introduction service. The two services perceive context of the conference and response accordingly. Details of the application will be described in the next sections, and we will use these related concepts to illustrate our idea.

3. Reference Programming Model

Table-driven design [8] is an approach to software engineering which is intended to generalize and simplify applications by separating program control variables and parameters (rules) from the program code and placing them in external tables. Our table-driven programming paradigm [9] for developing context-aware application was inspired originally by the table-driven design approach. The programming model presented in this paper is based on the table-driven programming paradigm. We extend the basic principle of traditional table-driven approach to separate context related components from application, which brings lower developing complexity as well as higher usability and adaptability to context-aware application.

The essence of the programming mode is a language specification associated with a system framework. We are not intending to design a bran-new or specific language in this paper. In fact, the language specification and related framework are referential. System developer can implement the model based on her interested programming language. That is why we call the model a "reference" programming model.

3.1 Ontology-Based Context Modeling

Ontology-based context modeling is currently a hot research topic. A general definition of context is presented in [10]. According to the authors, context is *any information that can be used to characterize the situation of an entity. An entity is a person, a place, or a physical or computational object that is considered relevant to the interaction between a user and an application, including the user and application themselves*. T. Winograd gives a special definition [11] of context: *something is context because of the way it is used in interpretation, not due to its inherent properties*. From the practical point of view, we prefer the latter definition because open-ended phrases such as "any information" and "characterize" are so broad that context covers everything

without boundary and is hard to be programmed.

Strang et al. [12] present a survey of six context modeling approaches, in which the ontology-based modeling method seems gaining unanimity in pervasive computing research communities. OWL-DL [13] is commonly used to represent ontology-based context model. We do not intend to study ontology-based context modeling technique itself, but just base our programming model on the technique. For the ease of illustration, we use the basic RDF [14] triple <*subject, predicate, value*>, as authors in [15] do, to represent context in our research field. The triple is expressed in the first order predicate logic. The subject means the owner that possesses the context. It may be a place, a person or a computing entity. The situation of subject is expressed in terms of multiple properties and their values, so we use the terms predicate and value to represent the situation of the subject with respect to a specific property.

In the example scenario shown in Sect. 2, there are several RDF triples that can be used to describe context information. For example, <*Smith, locatedIn, sessionRoom1*> indicates that a person named *Smith* is located in a place named *sessionRoom1*; <*smartPhone, hasType, O2XDAFlame*> shows that the type of the device *smart phone* is *O2 XDA Flame*. As is mentioned, using the basic RDF triple here is mainly to facilitate the description and understanding. Actually, context information is represented in the obscure and strictly grammatical OWL-DL when it is transmitted between context infrastructure and context-aware application.

3.2 Table Matching Strategy

Table matching is a very common procedure. Generally, we compare the given data with table items from the top row to the bottom, or in reverse order sometimes. In our programming model, we are considering other factors when matching table items. The table in our programming model should have the so-called "memory effect", that is, it can memorize the position of last matched row. Based the memorization, we clarify three matching strategies as follows:

- *Free mode*. This is the non-exclusive matching strategy. New incoming data will be compared with table items as usual no matter what the last matched index value is when using this mode.
- *Exclusive mode*. When using this strategy, new incoming data will not be compared with any items of the table as long as the last matched index is not a null value. The null value can be counted as a meaningless value for the table index. For instance, if the table index starts from one, we look on zero as the null value index.
- *Priority-based mode*. Each row of the table has a priority value. The row which has the higher priority will be compared earlier than those who have lower priority. Since computer always compares rows of table along a natural order, namely, the table index, we can simply equate the priority with the index value of table. If the table index value is calculated from low to hight, the row which has a lower index value has a higher priority. When adopting this strategy, new incoming data can only be compared with rows that have higher priority than that of the last matched row.

These three types of strategy are important in our table-driven programming model. We will discuss how to use them to resolve potential conflict problems later.

3.3 Table-Driven Programming Entity

Definition. A table-driven programming entity is a programming structure that can perceive changing of a specific context subject and react to it according to a virtual table which is constructed and maintained by system and configured by space manager.

To figure the definition out, let's focus on the basic structures that are used to compose a program, namely *Variable* and *Function* (we look on constant as special variable). In a typical context-aware application, programmer generally declares variables to acquire context-related information and defines functions to implement context-related operations. Different context corresponds to different variable value or function implementation. In other words, both the value of variable and implementation of function are sensitive to specific context subject. If associated with their interested context subject, both the variable and the function meet conditions to be a table-driven programming entity. Table-driven programming entity is the core part of the programming model.

There is a concept name "virtual table" in the definition. The reason why we call the table a "Virtual" one is that it is constructed by system in the light of the defining statement in the program, and it is transparency to programmer. As is mentioned, we consider that the space manager who knows detail of application requirements and is more familiar with the environment than programmer should take part in developing and deploying the context-aware application. So the virtual table connects knowledge of both the developer and the space manager. The developer defines table-driven programming entities in program. The system (actually, it is a preprocessing tool, we will give its detail later) constructs virtual tables according to the defining statements. The space manager configures the virtual tables according to actual demand when deploying the application. She can also adjust contents of the table when the application is running as long as she needs.

From the intuitive point of view, a table-driven variable (or function) is variable (or function) that is associated with its interested context subject. Conceptually, definition of a table-driven variable (hereinafter called TDV) can be represented as follows:

```
tdvDefine varType varName:cSubject[noteString]=dftValue;
```

where, *tdvDefine* indicates line-beginning of the statement

```
TDV_DEFINITION ::= "tdvDefine" <varType> <varName> ":"
                   double_quote <cSubject> double_quote "["
                   <noteString> "]" "=" <dftValue> ";"
TDF_DEFINITION ::= "tdfDefine" <retType> <funcName>
                   "(" <paraList> ")" ":" double_quote
                   <cSubject> double_quote "[" <noteString> "]"
                   <functionBodyStatement>
```

Fig. 1 Backus Naur Form of table-driven programming entity. TDV-DEFINITION is for table-driven variable, and TDF-DEFINITION is for table-driven programming function.

for defining a TDV; *varType* is the data type of the TDV and *varName* is the identifier; *cSubject* with a leading colon gives context subject to which the variable is sensitive; *noteString* with square brackets is a human readable string that prompts space manager what the variable is for; *dftValue* designates default value of the variable.

The following statement defines a table-driven function (hereinafter called TDF):

```
tdfDefine retType funcName(paraList):cSubject[noteString]{
    (Default body of the function)
}
```

Similarly, *tdfDefine* indicates line-beginning of the statement for defining a TDF; *retType* is the type of return value of the function; *funcName* is name of the function, and *paraList* denotes the parameter list of the function; *cSubject* with a leading colon gives subject of context that the function is sensitive; meaning of *noteString* is the same as the counterpart of TDV. Codes enclosed by a pair of curly brackets are the default body of the function. Effect of the default value for TDV and the default body of TDF will be explained later.

In order to describe TDV and TDF more accurately, we give partial definitions of them expressed in Backus Naur Form (BNF, [16]) respectively in Fig. 1. Meanings of the non-terminal symbols enclosed by angle brackets presented in Fig. 1 are similar to that we have just explained above. Since we are reconstructing variable and function of traditional programming language, those symbols also have the same meaning as counterparts in a concrete programming language.

Several table-driven programming entities can be found in the context-aware application of the example scenario presented above. For instance, the register service greets attendee in her mother tongue. The greeting words changes with the language setting stored in the user's handheld device. So we can define the greeting words as a TDV which is sensitive to the mother language setting. Commonly, each session room (or district) has a different session subject. Also, each session subject corresponds to different conference materials. When an attendee roams to a session room, she will get conference materials of the corresponding session subject. That is to say, the conference materials changes with the attendee's position. Therefore, the conference materials, which the system shall send to the user, can be defined as a TDV that is sensitive to the user's position. Besides, In both services, the method which the system utilizes to send messages to the user's device changes with the type of the device. Consequently, the method can be defined as a TDF that is sensitive to the type of the device.

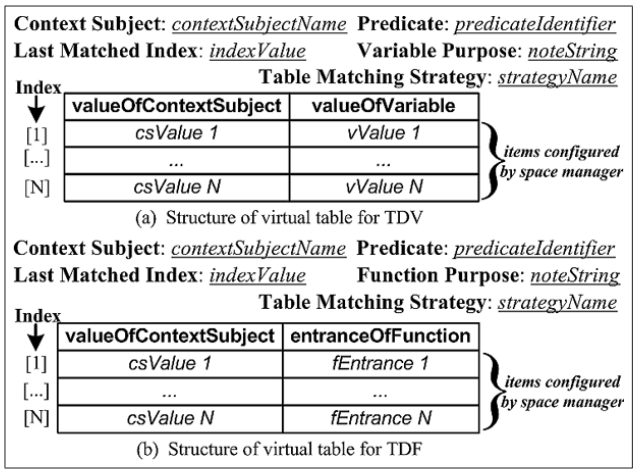

Fig. 2 Extended structure of virtual table for TDV and TDF. There are two columns, uncertain rows and several indispensable accessorial items in both tables. Rows that indexed from 1 to N will be configured by space manager.

3.4 Virtual Table

Defining statements of the table-driven programming entity will instruct system to construct virtual tables. It is the virtual table that connects knowledge of both developer and space manager. In order to combine the table matching strategy to solve potential conflict problems, we extend the structures of virtual table presented in [9]. Figure 2 shows the extended structures.

Figure 2 (a) shows structure of virtual table for TDV. There are two columns, uncertain rows and several indispensable accessorial items in the table. Bold texts in the topmost row are the table headers. The first column lists possible values of the interested context subject of the variable, while the second column lists possible target values of it. Number of rows is determined by space manager in her configuring process. Accessorial items include 1) *contextSubjectName*: interested context subject of the variable; 2) *predicateIdentifier*: identifier of the predicate in the RDF triple <*subject, predicate, value*>; 3) *indexValue*: table index of the last matched row; 4) *noteString*: human readable string which tells space manager what the variable is for; 5) *strategyName*: the name of strategy used by system to compare table rows. Its candidate choices are the aforesaid table matching strategy, that is, *free mode*, *exclusive mode* and *priority-based model*. Of all the three strategies, the *free mode*, which is the most commonly used, will be set to default one when creating virtual table. The space manager gets a clear understanding of her mission

from the *noteString* while she configures the virtual table. *PredicateIdentifier* indicated the relationship between *contextSubjectName* and *valueOfContextSubject*, which ensures that the space manager always fills meaningful items into the table.

The table matching procedure is under the guidance of the table matching strategy. Normally, values in the first column will be compared with the current context value extracted from the environment. Once they are matched, the corresponding value listed in the second column of the same row will be returned as the value of the variable. If none of the rows is matched, a null value will be returned; consequently the *dftValue* will be used by the application. When the comparing process ends, system will update value of the last matched index with the index value of currently matched row.

Figure 2 (b) shows the structure of virtual table for TDF. It looks very much like Fig. 2 (a). Actually, most of items in both tables are the same meaning, and the matching procedures are also similar. Difference between the two tables exists in the second column: the column value of virtual table for TDF is entrance of the function. The entrance has different meaning in different programming language. As is known, it would be function address or invocation name. If none of the value in the first column is matched, the default body of the function will be executed.

Although it is not mentioned above, there exists a one-to-one relationship between the table-driven programming entity and the virtual table. The virtual table acts as a linkage between the programming entity and the corresponding context subject.

General matching procedure is comparing items of table row by row, but sometimes this kind of matching will bring conflict problems. For example, a waiter service adjusts temperature in a rest room according to preference of people who comes in. A TDV named *targetTemperature*, which is sensitive to the user's comfortable temperature preference, was defined in the service to determine the temperature of the room. Several comfortable levels, each of which indicates a certain temperature, were configured into virtual table of the variable by space manager in advance. When the first people steps into the room, the service gets a target temperature value through comparing comfortable temperature preference of the people with items in the virtual table, and adjusts indoor temperature accordingly. It seems that the service works well. At that time, another people steps in, and the service adjusts temperature for the newcomer. But conflict arises if her comfortable level is different with that of the former people. The table matching strategy will help to resolve this type of conflict. Space manager can designate *exclusive mode* or *priority-based mode* while configures the virtual table. When using the *exclusive mode*, the first people who steps into the room will determine the temperature because the last matched index is not a null value as long as she stays in the room. If space manager designates *priority-based mode*, the people whose comfortable level is higher will determine the temperature whenever she steps in.

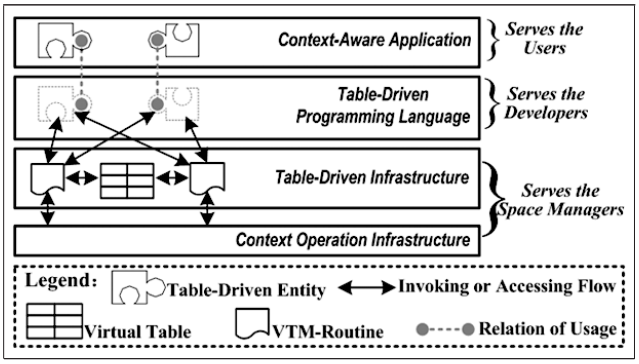

Fig. 3 Hierarchy of table-driven framework. Kernel of the framework is the table-driven programming language tier and the table-driven infrastructure tier.

3.5 Table-Driven Framework

Responsibility of table-driven framework involves two aspects: 1) Supporting table-driven programming entity at a level of programming language. 2) Supporting table-driven programming entity at a level of internal functioning mechanism. Figure 3 shows the hierarchy of table-driven framework.

The hierarchy presents relationships between context-aware application, table-driven programming language, table-driven infrastructure and context operation infrastructure. The table-driven programming language provides table-driven entities for context-aware application. The table-driven entity encapsulates context related operations and simplifies development interface, which enable the developer to concentrate on program logic without caring about how varieties of contexts are processed. Implementation of the entities depends on VTM (Virtual Table Management) routines provided by the table-driven infrastructure. Based on context repository, context operation infrastructure interacts with environment where the application runs. The infrastructure extracts context data from environment or exports values to it. Context operation infrastructure is not the focus of this study. According to [17], any changes of interested context will be reflected to the ontology-based context repository by the context operation infrastructure.

Obviously, both the table-driven programming language and the table-driven infrastructure are kernel of the table-driven framework. We will analyze how to implement the framework later.

From top of hierarchy, the application developer can see nothing but the table-driven programming language that supports table-driven programming entities. For a specific context-dependent variable or function, developer simply defines it as a table-driven programming entity and uses it in a traditional way. VTM-routines in the table-driven infrastructure assist constructing and maintaining the virtual table for the entity. When application runs, VTM-routines extract context data of current session from the context op-

eration infrastructure and complete the matching procedure. Appropriate results returned to application ensure that the latter behaves suitably.

3.6 Software Engineering Methodology

The purpose of studying the methodology is to clarify how to develop context-aware application with the programming model. Figure 4 shows the architecture of the programming model. Since context modeling and operating techniques are beyond the scope of this study, we focus mainly on top half the architecture (part that surrounded by rounded rectangle with dotted lines).

Along the sequential number marked in Fig. 4, we outline general process of developing context-aware application using the model from perspective of software engineering as follows:

Step 1: Space manager models the pervasive computing environment using context modeling tools. Modeling results are stored in the ontology-based context model repository of context operation infrastructure.

Step 2: Developer discovers interested context subject by browsing the ontology-based context model, then she writes context-aware application in table-driven programming language according to functional requirements of the space.

Step 3: The compiler of table-driven programming language preprocesses the context-aware application and creates virtual table for each table-driven programming entity defined in the program.

Step 4: Space manager configures the virtual tables with the assistance of routines in table-driven infrastructure according to actual circumstances of the environment.

Step 5: When context-aware application runs, the table-driven infrastructure extracts target context value of current session from the context operation infrastructure.

Step 6: Table-driven infrastructure compares the context value with items in the virtual table and returns appropriate value (variable value or entrance of function) to the context-aware application.

Step 7: The application continues to run and influences the environment (e.g. controlling devices or informing users, etc.).

From the process described above, we know that duties of roles who involve in development and deployment of context-aware application are clear. The developer concentrates her efforts on developing context-aware application without caring about context-related operations, while the space manager models the pervasive computing environment and configures virtual tables. The table-driven infrastructure supports running of the context-aware application, and the context operation infrastructure, which is based on the ontology-based context model repository, interacts with environments where application runs.

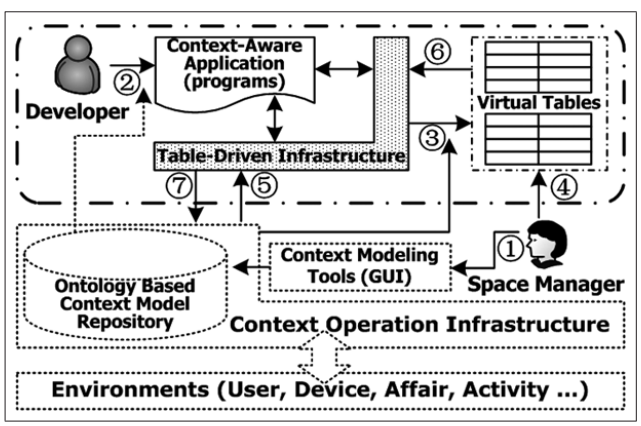

Fig. 4 Architecture of the reference programming model. Sequential number marked in the figure illustrates general development process using the programming model.

4. Table-Driven Framework Implementation

As mentioned above, the table-driven framework is composed of the table-driven programming language and the table-driven infrastructure. The two components support the virtual table at different level. In this section, we come to analyze how to implement the framework.

4.1 XML-Based Template for Virtual Table

Virtual table, the core concept of the framework, is used to exchange data between context-aware application and the table-driven infrastructure. To facilitate the data exchanging, we suggest to use XML [18] to represent virtual table. Since a one-to-one correspondence exists between table-driven programming entity and virtual table, and all the entities of the same type (TDV or TDF) have the same structure, an XML-based template could be used to create and manage virtual tables.

Figure 5 shows excerpt of an XML-based template for virtual table of TDV. The template file is encoded in UTF-8 [19]. Each element of the table has its corresponding item in defining statement of the table-driven programming entity. The "VTID" in line 2 is an unique identification for each table-driven programming entity. It acts as a link between the virtual table and the context-aware application.

```
1 <?xml version="1.0" encoding="utf-8" standalone="yes"?>
2 <VTDATA Version="1.0" Type="variable" VTID="">
3   <CONTEXTSUBJECT Value="" />
4   <PREDICATE Value="" />
5   <NOTESTRING Value="" />
6   <LASTMATCHEDINDEX Value="" />
7   <STRATEGYNAME Value="" />
8   <METADATA>
9     <FIELDS>
10      <FIELD AttrName="valueOfCS" FieldType="String" />
11      <FIELD AttrName="valueOfVariable" FieldType="String" />
12    </FIELDS>
13  </METADATA>
14  <ROWDATA>
15    <ROW valueOfCS="-" valueOfVariable="" />
16  </ROWDATA>
17 </VTDATA>
```

Fig. 5 Excerpt of XML-based template for virtual table of TDV. The property "Type" of element "VTDATA" indicates the type of virtual table. Property "VTID" indicates identification number of the table.

Elements in line 3 to 7 correspond to the indispensable accessorial items shown in Fig. 2. Elements "FIELD" in line 10 to 11 are table headers, of which the property "AttrName" indicates name of the column. The "valueOfCS" means value of the context subject, and the "valueOfVariable" gives value of the variable. Element "ROWDATA" embraces all row data of the virtual table. Line 15 gives an example row, its two attributes correspond to the two elements indicated in element "FIELDS".

Template for virtual table of TDF is similar to template shown in Fig. 5. Difference between the two templates exists in the table headers (elements named "FIELDS") and table rows (elements named "ROWDATA"), as it can be seen in Fig. 2. System uses these templates to create virtual tables for table-driven programming entities when preprocessing program codes of context-aware application. We will explain how the preprocessing tool utilize these templates to create virtual tables later.

4.2 Table-Driven Programming Language

The main task of table-driven programming language is to provide support for table-driven programming entity. According to the definition of the table-driven programming entities, a set of declarations and annotations that enable system to perform automatically "table-driven" actions must be added to a programming language. This is the so-called Language Extension. That is to say, the table-driven programming language is a language extension on a host language. The language extension offers a well-defined semantics and minimizes the workload of the developer while enabling maximal reuse and extensibility. The implementation of the language extension involves two aspects:

- Modifying syntax of the host language by adding grammatical structures as shown in Fig. 1.
- Implementing a preprocessing program. The programm processes source codes written in extension language and translates extended grammatical items into items in the host language.

Actually, these two aspects are inseparable, and substantive work lies in implementing the preprocessing program. Here we give the algorithm of the preprocessing program. Input of the algorithm is source codes in extension language, and the outputs of it are virtual tables corresponding to the table-driven programming entities and source codes in host language. To facilitate the description, we adopt the definition format of table-driven programming entity illustrated in Fig. 1.

Step 1: Searches for the first line that begins with *tdvDefine* or *tdfDefine* from the input source codes.

Step 2: If nothing is found, the preprocessing process stops; otherwise, checks syntax of the line found by utilizing the Backus Naur Form expressions illustrated in Fig. 1.

Step 3: If syntax of the line is wrong, shows prompt message and stops; if not, generates a serial number (named *vtID*) as the identification number of the table-driven programming entity.

Step 4: If the line found begins with *tdvDefine*, goes to Step 5; otherwise goes to Step 6.

Step 5: Replaces definition of TDV with its general format in the host language and changes all references of the TDV to expression *getVariable(vtID, varName)*; goes to Step 7.

Step 6: Replaces definition of TDF with its general format in the host language and changes all invocations of the TDF to expression *execFunction(vtID, funcName, paraList)*; goes to step 7;

Step 7: Creates virtual table with information such as *vtID*, *noteString*, *contextSubjectName* for the current entity (based on the template we have just discussed).

Step 8: Searches for the next occurrence of line that begins with *tdvDefine* or *tdfDefine* from the input source codes, then goes to Step 2.

Meaning of the *vtID* in step 3 is the same as the "VTID" in the template for virtual table. It is an internal ID and is used to refer to corresponding virtual table for TDV or TDF in the final program. The *getVariable* in step 5 and the *execFunction* in step 6 are fundamental VTM-routines provided by the table-driven infrastructure. The infrastructure also provides routines that are used to create and manage the virtual tables.

4.3 Table-Driven Infrastructure

According to the hierarchy shown in Fig. 3, the table-driven infrastructure provides the following VTM-routines:

- *CreateVirtualTable*. The routine will be invoked by preprocessing program of language extension. It firstly queries the ontology-base context model repository to find out the *predicateIdentifier* of specific context subject. As is mentioned above, default value for the last matched index is 0, and default matching strategy is *free mode*. When all these things are available, the routine creates virtual table based on corresponding XML-based table template.
- *A group of routines that assists space manager in configuring the virtual tables*. The configuration involves adding, editing and deleting items of the table. Besides, the routines assist space manager in designating matching strategy and adjusting sequence of rows when using priority-based strategy. At the end of each operation, consistency and integrity of virtual tables will be validated.
- *GetCurrentCSValue*. The routine works at the boundary between the table-driven infrastructure and the context operation infrastructure (see the number "5" in Fig. 4). It retrieves value of current context subject that belongs to current session from the context-operation infrastructure. To simplify operations, certain stringizing operator may be used to convert values.
- *GetVariable*. The preprocessing program replaces all reference to the TDV with this routine when processing

codes written in extension language. At run time, the routine compares value of the context subject in current session with counterpart item in virtual table for the variable based on the table matching strategy, and returns value of the current TDV.

- *ExecFunction*. The preprocessing program replaces all reference to the TDF with the routine when processing codes written in extension language. At run time, it compares value of the context subject in current session with counterpart item in virtual table for the function, and completes invocation of function according to the returned entrance of the function.

To sum up, the table-driven infrastructure provides support for the table-driven programming model at both design-time and run-time. At design-time, it supports the preprocessing program to create virtual tables. It also assists space manager to manage and configure virtual tables when deploying context-aware application. At run time, it enables the application to be driven by tables and to adjust its behaviors accordingly.

Based on the design ideas in this section, we implemented a table-driven framework on J2SE 6.0 [20]. we added language extension to Java language [20] with the help of JavaCC [21]. JavaCC is a popular parser generator, it reads a grammar specification and converts it to a Java program that can recognize matches to the grammar. The extension is named Table-Driven Java (hereinafter called **Tdava**). The framework provides a preprocessing tool named Table-Driven Java Compiler (hereinafter called **TDJC**). The TDJC reads source codes in Tdava, and produces source codes in Java and virtual tables described in XML. Figure 6 shows the sketch map of preprocessing procedure.

The table-driven infrastructure is provided in the form of a set of API (Application Programming Interface). The infrastructure uses SPARQL [28] Query Language for RDF to interact with the context-operation infrastructure. The Ontology-Based Context Model Repository and the APIs of Context-Operation Infrastructure are surrounded by surrounded by rectangle with dotted lines in Fig. 6. Actually, they are not part of the table-driven framework. We present them here just for illustrative purposes.

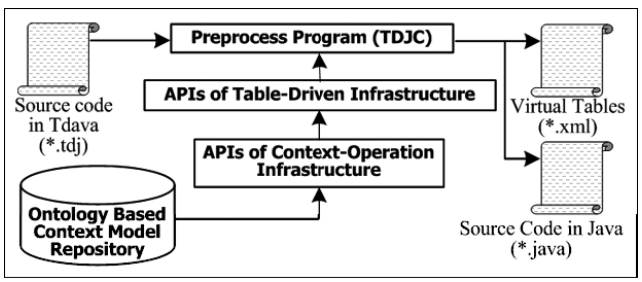

Fig. 6 Sketch map of preprocessing procedure of the TDJC. The TDJC reads source codes in Tdava, and produces source codes in Java and virtual tables described in XML.

5. Validation and Evaluation

In this section, we will validate the feasibility of the programming model by realizing the example scenario represented previously within the framework we have implemented. According to the architecture shown in Fig. 4, we need a context-operation infrastructure to support the table-driven framework.

5.1 Context-Operation Infrastructure Simulation

Generally, the context-operation infrastructure provides supporting for context modeling, interacts with environment and completes the invocation of context related operation. According to the hierarchy shown in Fig. 3, the table-driven infrastructure need not to interact directly with the physical environment. Therefore, we designed a context-operation infrastructure simulation platform to support the table-driven infrastructure. Our idea of simulation was inspired by simulation model described in [22]. We constructed a visualized context-operation infrastructure over OSGi [23] framework by encapsulating the VantagePoint [24]. VantagePoint is a software tool that allows the semantic modeling and interactive simulation of physical real-world environments. It creates semantic models of environments through graphical editing operations, visualizes semantic context information using Jena [25] Semantic Web framework, and allows developers to edit context information and trigger context events through graphical user interface. The resultant models are described by OWL [13] files that define rooms, devices, services and persons contained in the environment [26], [27].

The infrastructure simulation platform provides an event publication and subscription service for upper tier. It also provides a programming interface of context retrieving. Figure 7 shows relationship among the context-operation infrastructure simulation platform, the table-driven framework

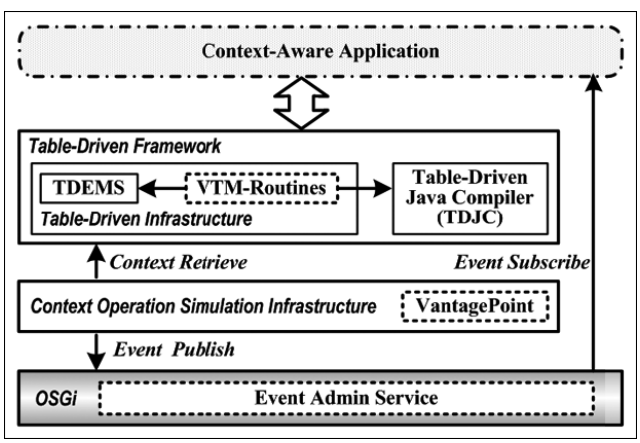

Fig. 7 Relationship among the context-operation infrastructure simulation platform, the table-driven framework and the context-aware application.

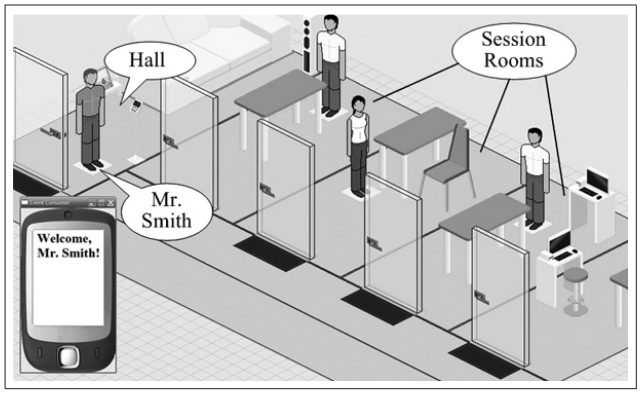

Fig. 8 Example scenario of an international conference. There are one hall and several session rooms in the space. The small window located in the left bottom corner, in which a mobile phone displays, simulates the hand-held terminal of Mr. Smith.

and the context-aware application.

The VantagePoint encapsulated in the infrastructure simulates real environment and acts as a context repository. The infrastructure accepts context retrieve request from the table-driven infrastructure, transforms it into query request which is described in ontology query languages SPARQL and sends it to VantagePoint. Query results from VantagePoint will be transformed and returned to the table-driven infrastructure. VTM-routines in the table-driven infrastructure support both the Table-Driven Entity Management System (TDEMS) and the TDJC, and interact with the context-aware application at run time.

5.2 Context-Aware Application for the Scenario

The example scenario, namely the international conference which is described in Sect. 2, can be modeled using the VantagePoint encapsulated in the context-operation infrastructure simulation platform. The resultant visualized model is shown in Fig. 8.

We aimed at the introduction service which sends conference materials of current session automatically to the attendees's terminal device when they roams the session rooms. As is analyzed above, the conference materials can be defined as a TDV which is sensitive to the user's position, and the send method can be define as a TDF that is sensitive to the type of the attendees's terminal device. Figure 9 (a) shows excerpt from source codes of the introduction service written in Tdava.

The introduction service is driven by event. In Fig. 9 (a), line 6 defines the topic of the event which may be triggered when an object in the VantagePoint is moved. Line 7 defines a TDV named **sDocs**, and it's interested context subject is denoted by a string expression, namely the "User.Position". As is known, the ontology-based context model is not a simple hierarchy structure. It is more a network structure. For example, the conception **User**, **Device** and **Position** are subclass of the conception **Entity**. Meanwhile, both the **User** and the **Device** has the property **Position**. The **User** has the property **Device**. In order to clearly specify the context subject that we want to describe, we use the expression which is separated by dots. The context subject "User.Position" here means the position of an user (Not position of other **Entity**). Line 8 defines a TDF named **sendDocs**, and it's interested context subject is "User.Device.Type". The context subject means the type of device that belongs to an user. Line 9 to 10 are default body of the **sendDocs**. According to the matching procedure of virtual table analyzed previously, if none of the rows in the virtual table for **sendDocs** is matched, the default body of it will be executed. Consequently, the expression "return false;" in line 9 leads to a failure sending operation. In practice, many reasons will lead to the result that no row is matched. For instance, when the application is just initialized, the space manager has not had the time to configure the virtual tables. Or when a new type of device emerges, there is no corresponding information in the virtual table. etc. Line 11 to 14 are the event handle. When the desired event is triggered, the TDF **sendDocs** is called with the TDV **sDocs** as it's parameter.

Figure 9 (b) shows the corresponding Java source codes of the introduction service. It is generated from source codes

```
1  package org.henrystudio.eventconsumer;
2  import org.osgi.service.event.*;
3  ...
4  public class eventHandler implements EventHandler, Runnable {
5   ...
6   public final static String topics = "org/henrystudio/cmss/vpevent/move";
7   tdvdefine String sDocs:"User.Position"["session docs!"] = "";
8   tdfdefine boolean sendDocs(String cDocs):"User.Device.Type"["send docs!"]{
9    return false;
10  }
11  public void handleEvent(final Event event) {
12   if (topics.equals(event.getTopic())) {
13    sendDocs(sDocs);
14   }
15  }
16  ...
17 }
```

(a) Source codes in Tdava

```
1  package org.henrystudio.eventconsumer;
2  import org.osgi.service.event.*;
3  import org.henrystudio.TDI.*;
4  ...
5  public class eventHandler implements EventHandler, Runnable {
6   ...
7   public final static String topics = "org/henrystudio/cmss/vpevent/move";
8   String sDocs = "";
9   boolean sendDocs(String cDocs){
10   return false;
11  }
12  public void handleEvent(final Event event) {
13   if (topics.equals(event.getTopic())) {
14    private Routines tdrs = new Routines();
15    tdrs.currentSubjectValue = event.getProperty("URI");
16    tdrs.EXEC(2, this.className, "sendDocs", tdrs.GETV(1, sDocs));
17   }
18  }
19  ...
20 }
```

(b) Corresponding Java source codes produced by TDJC

Fig. 9 Excerpt source codes of the service.

illustrated in Fig. 9 (a) by the TDJC. The TDJC works according to the algorithm presented in Sect. 4.2. Line 3 is used to import packages of the table-driven infrastructure. The class **Routines** referenced in line 14 is from the infrastructure, and it's instance **tdrs** is used to represent executing environment of the framework. Member of the class, namely the **currentSubjectValue** referenced in line 15, denotes value of the subject on which the event occurred. Line 3, 14, 15 are added by the TDJC. Since line 7 in the Fig. 9 (a) begins with "tdvdefine", the TDJC extracts context subject from it and creates corresponding virtual table. It also replaces the definition of the TDV with it's general format in the host language (line 8 in Fig. 9 (b)), and changes all references of the TDV to expression **tdrs.GETV(1, sDocs)** (line 16 in Fig. 9 (b)). The number "1" in the parameter list is the ID of the TDV, and the **sDocs** maintains its original meaning as a traditional variable. Similarly, the TDF defined in line 8 to 10 in Fig. 9 (a) are transformed into it's general format in the host language (line 9 to 11 in Fig. 9 (b)). In addition to the virtual table being created, all invocations of the TDF will be changed to expression **tdrs.EXEC(2, this.className,"sendDocs", ...)**. The number "2" in the parameter list is the ID of the TDF. Parameter **this.classname** and **"sendDocs"** ensure that the default body of the function will be executed when the matching of virtual table is failed.

Finally, we compiled these Java source codes using Javac into executable Java Bytecode [20]. Under the premise of appropriate values being configured into virtual tables of these table-driven programming entities, the introduction service works well. When we "let" Mr. Smith roam about session rooms, documents of current session are sent to his terminal device automatically.

From the excerpt shown in Fig. 9 (a), we can see that the codes written in Ddava is compact, succinct, and is easy to understand and maintain. Moreover, we need not to know details of concrete context information before development. Those information is left to the space manager to configure when deploying the application. For programmers, the most important thing is to recognize which variable (or function) could be defined as a TDV (or TDF), and to which context subject the TDV (or TDF) is sensitive.

The table-driven programming entities isolate context-related operations from base programming logic. What will happen if there is no supporting of table-driven infrastructure to develop the introduction service? Suppose the programmer intends to develop the service using plain Java in traditional way. First of all, she needs to consult the space manager to learn information about the sessions. The information involves number and names of the rooms, conference materials corresponding to each of the sessions, and possible device type of hand-held terminal that registered in the attendee's receipt. In the program, she has to write codes to retrieve the position information of the current user from the context-operation infrastructure, and complete the matching between the position and the conference materials of session room using the switch-case statement of Java language. She also has to retrieve device type of the current user and choose proper function package to finish the sending procedure. So it's no doubt that the codes of the final program is inevitably diffuse and difficult to understand. In addition, switch-case statements bring about hard-coded codes to program and make the application difficult to maintain.

Table 1 Comparison between the traditional programming in Plain Java and the table-driven programming in Tdava.

Aspects Compared	Developing in Plain Java	Developing in Tdava
Time Consumed in Consulting (days)	2	0.5
Time Consumed in Programming (days)	4	0.5
Time Consumed in Deploying (days)	0.5	2
Number of Lines of Effective Codes (lines)	340	45

Based on the same context-operation infrastructure simulation platform, we have tried to develop the introduction service using plain Java, and found that the two methods were different in development cycle time and number of lines of effective codes. The development cycle time involves 1) the time consumed by the programmer in consulting the space manager what the service requirements are; 2) the time consumed by the programmer in programming; 3) the time consumed by the space manager in deploying the service. A comparison between the two methods is shown in Table 1. The table shows that it takes less total time to develop the introduction service using the table-driven programming model than that using the traditional method. Besides, number of lines of effective codes is far less than the latter when using the former approach. Overall, the table-driven programming model is excelled than the traditional method in developing context-aware application. Although the application of the introduction service is simple and is not universal, we can get a rough idea of the potential of the model.

5.3 Advantages of the Model

As an abstraction of context-related element, the table-driven programming entity isolates complex context-operations from program logic of context-aware application. Virtual table corresponding to the entity connects knowledge of both developer and space manager. By this way, the application developers can concentrate on upper-layer logic of program without concerning about how those contexts being processed, which reduces lots of programming efforts from them. In this sense, the model provides better isolation for development from the perspective of software engineering.

The model also brings better flexibility. It enables both developer and space manager to cooperate in a loosely-coupled fashion. Preprocessing program produces virtual table according to defining statement written by the developer, and the space manager can configure the virtual ta-

ble at any time to adapt the context-aware application to the changing environment. For example, in the introduction service of the example scenario, developer defines session document variable which is sensitive to session rooms. Space manager preliminarily configures virtual table for the variable in light of the session rooms arrangement when deploying the service. When the conference is in progress, she can reconfigure items of the table whenever necessary to adapt the service to latest rearrangement of session rooms, meanwhile, the introduction service needs neither being reprogrammed nor being interrupted and started over.

Besides, the model is lightweight and easy to be utilized. For one thing, the framework of the model is lightweight and is easy to be realized in the light of the implementation suggestion proposed above; for another, no matter what language the model is realized in, development is in substance within the frame of the host language, and developer needs not to learn much more about a new language. Compared with the approaches discussed foregoing, the programming model is easier to program and deploy, and the source codes of application is succinct and is easy to read and maintain.

Actually, the table-driven programming model can be embedded into another programming model because of its lightweight, provided both the models are based on the ontology-based context modeling technology. When our programming model is embedded into another model, it's better to process the source codes with the preprocessing program of our model firstly. According to the algorithm given in Sect. 4.2, the preprocessing program only processes lines that begin with *tdvDefine* or *tdfDefine*, and the rest codes will be left to another compiling tools. The embedded programming model has no impact on the host model. Instead, it enhances flexibility of the latter.

5.4 Limitation and Future Work

According to the definition given in Sect. 3.3, the table-driven programming entity is sensitive to a specific context subject. In general, we can find a specific interested context subject for variable or function which is changed with the environment, but it was not always the case. Sometimes, in complex context environment, variables and functions are sensitive to multiple context subjects. That's to say, they are determined by a combination of several subjects. Currently, our programming model is insufficient to deal with this kind of combination. As a lightweight programming model, it is suitable to describe the situation that programming entity is the one-to-one relationship with context subject, but its expression power is limited when the latter has a many-to-one relationship with the former.

In fact, in the context-aware application which is based the ontology-based context model, the "combination" problem can be partially resolved. For instance, in an intelligent family health care system, a patient aiding service monitors pathological status of the patient and makes some response. when the pulse frequency, the blood pressure and the body temperature of the patient reaches a certain level, the patient's condition may have reached a critical state and the doctor must be sent for. Generally speaking, the function which is used to send for the doctor is sensitive to the pulse frequency, the blood pressure and the body temperature. It seems that we face the combination of context subjects when developing the service. In fact, this situation can be avoided by the context reasoning mechanism [15] in a typical context-operation infrastructure. We can add a context subject named "patientConditionLevel" to context subject "patient" when modeling the system. The context subject "patientConditionLevel" denotes the level of the patient's condition. The context reasoning mechanism will reason out a certain level value of the subject using the value of context subjects such as the pulse frequency, the blood pressure and the body temperature. The patient aiding service can use the value of the "patientConditionLevel" to determine what action should be taken. Thus, the function which is used to send for the doctor can be defined as a TDF which is sensitive to the context subject "patientConditionLevel". It is actually a one-to-one relationship. So as is mentioned above, it is very important for developer to recognize which variable (or function) could be defined as a TDV (or TDF), and to which context subject the TDV (or TDF) is sensitive.

However, the context reasoning mechanism can not fundamentally solve the "combination" problems. Making the model support for complex context environment is one of our research direction in future. Furthermore, the procedure that the space manager configures numbers of virtual tables for big scale application is somewhat burdensome. We are also exploring different ways to configure the tables automatically.

6. Related Works

In COP [3], [4], lots of stubs are defined in the repository of candidates. Each of the stubs corresponds with a specific context. The programmer embeds open terms in the program when developing. The open terms are gaps which will be replaced with selected stubs by the context-filling operation at run time. Context-filling operation involves two steps. First, it selects appropriate stub according to value of the current context and description of the open term. Second, it binds value of variables in the program to parameters of the stub. COP separates context related operation from application logic by matching open terms with stubs, and as a result, factors of environment are decoupled from application at design time. However, the developer has to know details of the environment and how to react to them before she writes program, otherwise she can not define stubs and use open terms.

Layer is an important concept in ContextL [5]. Partial class and method definitions can be associated with layers in program logic. The layers can be activated or deactivated according to the context of running environment. When a layer is activated, the partial definitions become part of the program until this layer is deactivated. A novel reflec-

Table 2 Qualitative comparison of capability between the table-driven programming model and previously approaches.

Comparison Indexes	COP	ContextL	MFP	TDP
Q1: level of codes simplifying	Low	Normal	Medium	High
Q2: language preprocessing involved	No	Yes	Yes	Yes
Q3: approach can be embedded into other programming model	No	No	Yes	Yes
Q4: approach is NOT limited to a specific language platform	Yes	No	Yes	Yes
Q5: details of context need NOT to be known by developer beforehand	No	No	No	Yes
Q6: approach can be applied in complex context environment	No	Yes	Partly	Partly
Q7: behavior of application can be adjusted while system keep running when new situation emerges	No	No	Yes	Yes

Note: COP = Context-Oriented Programming, MFP = Multifaceted Programming, TDP = Table-driven Programming (our reference programming model in this paper). Q1: the higher the level is, the easier the code of application to be read and maintained. Q2: if yes, partial burden of developer can be reduced by preprocessing program. Q3: if yes, the approach is sufficiently lightweight and can be used to enhance flexibility of other programming model. Q4: if yes, the approach can be applied widely, that is, theoretically developer can choose from most of current language she likes to program. Q5: if yes, developer can concentrate her efforts on logic of application without caring about how contexts are. Q6: if yes, the approach can be applied in complex context environment; if partly, the approach can be applied partially or be conditionally applied. Q7: if yes, it is convenient to adjust behavior of application temporarily when new situation emerges, with no need to stop the running system.

tive interface controls the activation and deactivation of layers [6]. As is analyzed, ContextL makes programs to adapt to context flexibly while keeping the conceptual simplicity of object-oriented programming. Nevertheless, the programmer has to define layer meta-classes, decide which layers are instances of which layer meta-classes, define methods correctly and understand the interactions between different methods, so development in ContextL is somewhat complex.

Context-dependent behaviors can be abstracted as multifaceted programming entities when developing with the multifaceted based programming paradigm [7]. Each facet of the multifaceted programming entity denotes a pair of specific context and corresponding behavior. Also, each facet can be exposed or hidden by a switching process according to the current context. The switching process is under control of the exposing strategy. By switching facets of the entity, the application behaviors suitably in the light of changing of the environment. The multifaceted programming entity can hold any number of facets, but all these facets must be designated at design time and can not be modified easily at run time to adapt to changing of the environment, which makes the paradigm be of less flexibility. Besides, it is inefficient to modify program instantaneously when the behavior adjustment of context-aware application cannot adapt to the changing of environment any longer.

Table 2 shows qualitative comparison between the approaches discussed and the reference programming model presented in this paper. To facilitate the description, we abbreviate the reference programming model as TDP (combination of the first letter of "table-driven programming"). The comparison is based on index from Q1 to Q7. From the comparison, we can see that the TDP is lightweight and has better flexibility when applying in general environment. As is analyzed in Sect. 5.4, the TDP can be partially applied in complicate context environment on the premise of the "combination" problems being resolved by context reasoning.

7. Conclusion

Approaches proposed by researchers today to facilitate development of context-aware application solve some key problems of pervasive computing programming, but they are either lack sufficient flexibility or somewhat complex for programming and deploying. In this paper, a novel lightweight reference programming model is proposed to make up inadequacy of those approaches to some extent. The model isolates context-dependent behaviors from the application logic using virtual table that connects knowledge of both developer and space manager, which consequently enables both the roles to cooperate in the development process in a loosely-coupled fashion. Hierarchy, architecture and suggestions for implementing the model are presented. We validate and evaluate the model by implementing an example scenario, and compare the model with previously related works. The experiment shows that the model is lightweight and brings better isolation and flexibility.

As a lightweight programming model, the proposed model provides limited supporting for developing context-aware application in complicate context environment. Besides, the procedure that the space manager configures the virtual tables for big scale application is somewhat burdensome. We will address these issues in future.

Acknowledgements

The work is supported by National Natural Science Foundation of China under Grant No.(60933003), the National High Technology Research and Development Program (863 Program) of China (2006AA01Z101), and IBM Joint Project (JLP200906008-1).

References

[1] A. Dey and G. Abowd, "Towards a better understanding of context and context-awareness," Workshop on the What, Who, Where, When and How of Context-Awareness at CHI 2000, April 2000.

[2] K. Henricksen and J. Indulska, "A software engineering framework for context-aware pervasive computing," Proc. Second IEEE Annual Conference on Pervasive Computing and Communications'04, 2004.

[3] R. Keays and A. Rakotonirainy, "Context-oriented programming," Proc. 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access, pp.9–16, San Diego, CA, USA, 2003.
[4] A. Rakotonirainy, "Context-oriented programming for pervasive space," Proc. ACM Dynamic Languages Symposium, 2006.
[5] P. Costanza and R. Hirschfeld, "Language constructs for context-oriented programming: An overview of contextL," Dynamic Languages Symposium (OOPSLA'05), San Diego, USA, Oct. 2005.
[6] P. Costanza and R. Hirschfeld, "Reflective layer activation in contextL," Symposium on Applied Computing, Proc. 2007 ACM Symposium, March 2007.
[7] A. Rarau, K. Iulian Benta, and M. Cremene, "Multifaceted based language for pervasive services with deterministic and fully defined behavior," Pervasive Services, IEEE International Conference, July 2007.
[8] W. Cunneyworth, "Table driven sesign: A development strategy for minial maintenance information systems," Data Kinetics, June 1994.
[9] J.B. Zhang, Y. Qi, D. Hou, and M. Li, "A table-driven programming paradigm for context-aware application development," SAINT'09, 2009 Ninth Annual International Symposium on Applications and the Internet, Seattle, USA, July 2009.
[10] A.K. Dey, D. Salber, and G.D. Abowd, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," Human-Computer Interaction Journal, vol.16, no.2, pp.97–166, Dec. 2001.
[11] T. Winograd, "Architectures for context," Human Computer Interaction, vol.16, no.2-4, pp.401–419, 2001.
[12] T. Strang and C.L. Popien, "A context modeling survey," Workshop on Advanced Context Modeling, Reasoning and Management as Part of UbiComp 2004, The 6th International Conference on Ubiquitous Computing, pp.33–40, Sept. 2004.
[13] W3C OWL Working Group, "OWL 2 web ontology language document overview," World Wide Web Consortium (W3C), http://www.w3.org/TR/owl2-overview/, accessed Aug. 2009.
[14] P. Hayes and B. McBride, "RDF semantics," World Wide Web Consortium (W3C), http://www.w3.org/TR/rdf-mt/, accessed Aug. 23, 2009.
[15] D. Ejigu, M. Scuturici, and L. Brunie, "An ontology-based approach to context modeling and reasoning in pervasive computing," Proc. Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07), 2007.
[16] D.E. Knuth, "Backus normal form vs. backus naur form," Commun. ACM, vol.7, no.12, pp.735–736, Dec. 1964.
[17] D. Preuveneers, Y. Vandewoude, P. Rigole, D. Ayed, and Y. Berbers, "Context-aware adaptation for component-based pervasive computing systems," Adjunct Proc. Pervasive 2006, Dublin, May 2006.
[18] XML Core Working Group, "Extensible markup language (XML)," World Wide Web Consortium (W3C), http://www.w3.org/XML/, accessed Aug. 23, 2009.
[19] The Unicode Consortium, "Unicode transformation format-8," Unicode, Inc., http://unicode.org/resources/utf8.html, accessed Aug. 23, 2009.
[20] Oracle Technology Network, "Java SE overview — At a glance," Oracle, http://www.oracle.com/technetwork/java/javase/overview/index.html, accessed Sept. 1, 2010.
[21] S. Viswanadha and S. Sankar, "Java compiler compiler (JavaCC) - The java parser generator," Java.net, https://javacc.dev.java.net/, accessed Aug. 23, 2009.
[22] M.C. Huebscher and J.A. McCann, "Simulation model for self-adaptive applications in pervasive computing," DEXA, Database and Expert Systems Applications, 15th International Workshop on (DEXA'04), pp.694–698, 2004.
[23] OSGi Community, "OSGi - The dynamic module system for java," OSGi(TM) Alliance, http://www.osgi.org/, accessed Aug. 23, 2009.
[24] I. Niskanen, "VantagePoint," http://www.vtt.fi/proj/vantagepoint/index.jsp, VTT Technical Research Centre of Finland, accessed Aug. 23, 2009.
[25] HP Labs Semantic Web Programme, "Jena — A semantic web framework for java," Hewlett-Packard Development Company, http://www.openjena.org/, accessed Aug. 23, 2009.
[26] I. Niskanen, J. Kalaoja, J. Kantorovitch, and T. Piirainen, "An interactive ontology visualization approach for the networked home environment," International Journal of Computer and Information Science and Engineering, vol.1, no.6, pp.370–375, 2007.
[27] T. Piirainen, I. Niskanen, J. Kantorovitch, and J. Kalaoja, "Context simulation and support for context-aware application development," Proc. 2nd IEEE European Conference on Smart Sensing and Context, EuroSSC 2007, Kendal, UK, Oct. 2007.
[28] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," World Wide Web Consortium (W3C), http://www.w3.org/TR/rdf-sparql-query/, accessed Aug. 23, 2009.

Junbin Zhang received his Master of Engineering degree from Dept. of Computer Science and Technology in Xi'an Jiaotong University, China, in 2004. He is currently a Ph.D. candidate in Dept. of Computer Science and Technology of Xi'an Jiaotong University. His interests include software engineering, pervasive computing and work flow related technologies.

Yong Qi received his Doctor of Engineering degree from Dept. of Computer Science and technology in Xi'an Jiaotong University, China, in 2001. He work as a professor and a doctoral supervisor in Dept. of Computer Science and Technology of Xi'an Jiaotong University. His interests include Distributed Computing System, Pervasive Computing, Mobile Computing, Object-oriented Technology and Middle-ware technology. He is the syndic of China Computer Federation (CCF), the administrative syndic of China Computer Federation Software Committee.

Di Hou work as an associate professor of Dept. of Computer Science and Technology in Xi'an Jiaotong University. His interests include software system architecture and database technology.

Ming Li received her Master of Engineering degree from Department of Information Management, Xi'an University of Technology, 2003. She works for Department of Information Management, Xi'an University of Technology, and is currently a Ph.D. candidate in Dept. of Computer Science and Technology of Xi'an Jiaotong University. Her interests include Software Engineering, Pervasive Computing and E-Commerce.