Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT
Computer Standards & Interfaces 26 (2004) 377–399
COMPUTER STANDARDS & INTERFACES
www.elsevier.com/locate/csi

Designing an XML-based context-aware transformation framework for mobile execution environments using CC/PP and XSLT

Tzu-Han Kao*, Shyan-Ming Yuan

Department of Computer and Information Science, National Chiao Tung University, 1001 Ta Hsueh Rd., Hsinchu 300, Taiwan

Received 13 July 2003; received in revised form 28 September 2003; accepted 28 September 2003

Abstract

Mobile and embedded devices provide the function of surfing the Internet anytime and anywhere. There are several kinds of mobile execution environments (MExE) built on these appliances, such as WAP, J2ME, PJava, and Microsoft CLI. It is difficult for programmers to write a program only once and then execute it on these mobile devices. The primary reason is there are a variety of devices with different runtime environments and diverse hardware/software capabilities. Therefore, in order to accomplish the following: (1) applications can be designed regardless of what kind of the target mobile device belongs to; (2) the program of an application can be automatically adapted to the target MExE environments. We propose an *X*ML-based *C*ontext-*A*ware transformation *F*ramework (X-CAF). In this framework, we design an XML-based programming model to divide programmers into two roles, user interface (UI) designer and logic programmer, so as to efficiently develop an application in separation-of-concern way. Besides, we exploit the XSLT/XPath transformation mechanism to transform documents of XML User-interface Language (XUL) and *LoGic Markup Language (LGML)* into others of the target MExE languages by means of the context information, device capabilities and user preferences. Moreover, to generate codes of the applications flexibly and efficiently, we divide the code processing of an application into that of the user interface occurring at runtime and that of the event-handling logic occurring at static time. In brief, our paper contributes an XML-based application development environment and transformation framework to the access to device independence.

Keywords: Transformation; Context-aware; Mobile execution environment; XSLT; CC/PP; XML; XUL; Programming

1. Introduction

Mobile and wireless technologies have been changing over the past few years. Through mobile and embedded devices, such as PDAs, palms, smart phones, and Java phones, people can surf the content on the Internet. Besides, they can download and install applications from a content provider's server over the Internet, like Java game download. Currently there have been four kinds of the mobile execution environments on plentiful mobile appliances. These environments include WAP [2], J2ME [4,5], PersonalJava [3], and Microsoft CLI. In the Microsoft .NET platform [25], the mobile runtime environment supported can be classified into: (1) ASP .NET Mobile Pages [27] and (2) .NET Compact Framework [28]. The former attempts to support major PDAs, cell phones, pagers,

* Corresponding author. +886-3-571-2121x59265 (Lab.), +886-3-571-2121x56602 (Office); fax: +886-3-572-1490.
E-mail addresses: gis89539@cis.nctu.edu.tw (T.-H. Kao), smyuan@cis.nctu.edu.tw (S.-M. Yuan).

0920-5489/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2003.09.003

and other devices, while the latter supports all equipments running Pocket PC 2000, Pocket PC 2002, Pocket PC Phone Edition, etc. Called *M*obile *Ex*ecution *E*nvironments (MExE) in the standard [1], these environments—class mark 1 to 4—are defined by the 3GPP working groups. They stand for WAP, J2ME, PJava, and Microsoft CLI, respectively.

Each of these mobile runtime environments has abundant resources and application interfaces (APIs) for application developing. Nevertheless, in application developers' points of view, developing applications for certain platform of the execution environments, they difficultly execute these applications on any other of the mobile execution environments. For example, a J2ME application cannot be executed on a cellular phone merely with a WAE platform (the runtime environment of WAP). Due to a wide range of the client devices with diverse hardware/software capabilities, a number of researches focus on device independence [6–9] and context-aware computing [10,11] for mobile execution environments.

In our paper, in order to achieve two objectives: (1) applications can be designed without concerning about what kind of the target mobile devices belongs to; (2) the program of an application can be automatically adapted to the target MExE environments. We propose an *X*ML-based *C*ontext-*A*ware Transformation *F*ramework (X-CAF), and design two primary techniques an *XML-based programming model* and a *context-aware transformation mechanism*.

The *XML-based programming model* divides programmers into two roles. One role is a user interface (UI) designer who focuses on devising the user interface of an application by writing an XUL [22,23] document, an XML-based user interface description language. The other role is a logic programmer who concentrates on implementing the event-handling logic of the application by writing an LGML document. *LoGic Markup Language (LGML)* is an XML-based language that we design can describe the event-handling logic of the application, explained in Section 4. XUL and LGML share the same characteristic of being all XML-based languages, the objectives of which vary, yet. XUL is capable of describing the user interfaces of the applications running on desktop computers, while LGML can generally describe the event-handling logic with computational statements: if-then-else condition-control statements, for-loop flow-control statements, etc. This division can accelerate developing applications, because each role only devotes himself to designing the user interface or the event-handling logic.

The *context-aware transformation mechanism* aims to solve the following problem. This problem is encountered during our system transform a program of an application into the codes written in the MExE languages. Some widgets on the user interfaces need to be adjusted during the transformation, or they may not be displayed on screens of some devices. These widgets are called device-sensitive widgets in this paper. An image element, for example, can be shown on the screens of Java phones if its type is PNG; if remaining PNG, it cannot be displayed on those of the WAP-capable devices. Namely, without being adapted appropriately in terms of the context information of these devices, the widgets might not be displayed on some devices; accordingly, a context-aware transformation mechanism is designed to solve it. We transform XUL and LGML documents into the others with the syntaxes of the target MExE languages by applying XSLT/XPath [12], and adapt device-sensitive widgets on user interfaces to the context profiles of the mobile devices based on the context information *user device profile* and *user preference profile*.

Also, in order to adapt and transform XUL and LGML documents, and compile their output codes (translate a source program into a binary executable code) efficiently and flexibly, we divide the entire process of the LGML and XUL documents into two sub-processes which process XUL and LGML documents separately: (1) to transform XUL documents at runtime, called *runtime XUL transformation*, and compile the transformed data subsequently, called *runtime compilation*; (2) to transform LGML documents at static time, named *static-time LGML transformation*, and compile the transformed data subsequently, named and *static-time compilation*. These operations will be illustrated in Section 5.3. Briefly, in this system, we emphasize on developing an XML-based programming model and context-aware transformation framework to approach cross-platform and device-independent.

This paper is organized as follows. Section 2 is an introduction to some related work and the background technologies. After an overall explanation of the context-aware transformation framework in

Section 3, we describe an XML-based programming model using XUL and LGML in Section 4. Section 5 presents the context-aware transformation mechanism including user interface adaptation, code transformation, and code serialization for applications. Finally, we conclude and discuss future work in Section 6.

2. Backgrounds

2.1. Context information

There are several important international working groups who endeavor to provide the related technologies for the context information and device-independence of the mobile devices, such as Device Independence Working Group (DI WG) [9], Composite Capabilities/Preferences Profile Working Group (CC/PP WG) [13–16] in W3C. In addition, they define and standardize device capabilities and user preferences of the mobile and embedded devices for authoring, adaptation and presentation of Web content and applications that can be delivered effectively through different access mechanisms.

CC/PP [13–16] derives from earlier work done within the W3C Mobile Access Interest Group and the WAP Forum's User Agent Profile working group. In the CC/PP framework, the context information of the mobile and embedded devices is collected in appearance of profiles in the XML/RDF format [17,18]. These profiles described with XML come into two advantages—validity and well-formedness. Furthermore, the context profiles can facilitate resources exchange on the Internet after using the RDF framework with three-tuples (object, type, value).

WAP UAProf [19] is a general and extensible framework from the CC/PP framework. It is standardized by the WAP Forum and proposed to specify user preferences and device capabilities. DELI [20], an open-source library developed at HP Labs, exploits Java servlets as server components to resolve HTTP request messages containing CC/PP or UAProf context information. It implements a negotiation protocol between WAP devices and servers (e.g., Wireless profiled HTTP). We use it as the context-aware service in this framework, explained in Section 3.

2.2. XML-based user interface description

There are several markup languages capable of describing user interfaces of applications: XUL [22,23], UIML [21]. XML User-interface Language (XUL) is designed as a cross-platform language to describe the graphical user interfaces of the applications on desktop computers. It works to make the user interfaces of the applications portable. Being an XML-based user interface description language as well, UIML [21] provides a user interface model with five description blocks, including description, structure, data, style, and events. Because of the generality of UIML, it can describe the user interfaces of the applications running on desktop computers, and even, small and handheld devices.

Although the two languages get developed, in our research, at the present time, we focus on transforming codes of applications to enable those to be run on the mobile execution environments. Thus, it is not necessary to apply sophisticated methods to our system design. In other words, it is meaningless to define the <toolbar>, <scrollbar>, <progressmeter> elements in the user interface description language for the applications on the small and display-limited handsets, because these control elements cannot be displayed on the screens of these devices.

Therefore, we abstract useful elements from the original XUL elements to describe the widgets on the screens of the display-limited devices. These abstracted ones are sufficient to describe the widgets of the user interfaces of the applications on the mobile or small handheld devices, such as the <textbox> element representing a Text Field. The other elements are arranged in Table 2 in Section 5.3.

2.3. XML-based transformation

To transform an XML document into another, we can apply SAX, DOM, and XSLT/XPath. In this research, we exploit the XSLT technology [12] to convert XML documents to others with the syntaxes of the target MExE languages. We can write an XSLT style sheet to specify "How to transform an XML document into another with the syntax of the target language we want". A template, in a style sheet, is used to represent a fragment of a result tree (a transformed tree) to substitute for some parts of an input

XML document. XPath, a language for addressing parts of an XML document, is the method which provides the path expression mechanism to select the sets of the nodes of an input tree; this use of XPath is described in Ref. [34].

3. *X*ML-based *C*ontext-*A*ware *F*ramework (X-CAF)

(Fig. 1).

3.1. Device tier

This tier involves the mobile and embedded devices capable of downloading applications from the servers over the Internet. On these devices, all of the four MExE environments are capable of running applications. For example, the execution environment of Java phones is composed of MIDP, CLDC, and KVM, supports J2ME MIDlets running.

3.2. Bearer tier

This tier consists of wireless access networks (2.5G, 3G, wireless LAN, etc). It provides the functionality for these client devices to request services at the server side, and to obtain required information from the servers: via 2.5G or 3G wireless networks, a Java phone can download a J2ME game application from some content provider's server.

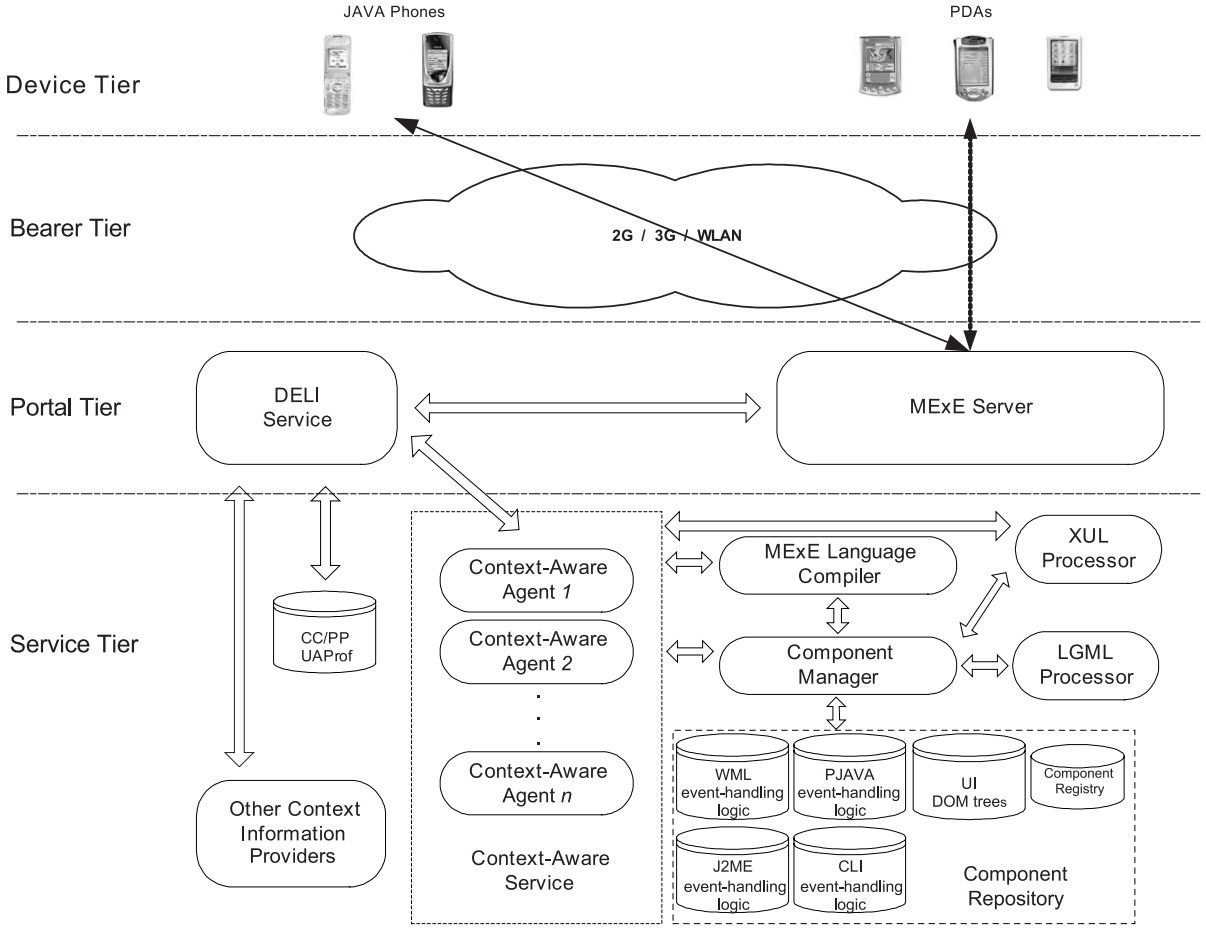

Fig. 1. The overview of our system framework inclusive of four layers: Device Tier, Bearer Tier, Portal Tier, and Service Tier.

3.3. Portal tier

In the tier, there is a portal, named *MExE Server*, responsible for listening requests sent from client devices and replying the required data to these client devices. There could be two types of the protocol to deliver the context information of the client devices. One transports profiles over Wireless Profiled HTTP (W-HTTP) and the other transports profiles over a combination of WSP and HTTP 1.1. We use DELI, which serves as the context resolver, can parse and resolve the request message containing the requesting client's context profiles. Essentially, it takes advantage of Java Servlet [24] and JENA [20]. Java Servlet technology provides a mechanism that web developers can extend the functionality of a Web server. JENA supports some application interfaces (APIs) for retrieving the context information related to client devices. Through this mechanism, this system can be aware of user preferences and device capabilities to adapt the device-sensitive widgets of the user interfaces to the context profiles of the devices.

3.4. Service tier

This tier contains several core components including: *CC/PP and UAProf storage*, *component repository*, *component manager*, *XUL and LGML processors*, *context-aware service (CAS)*, and *MExE Language Compiler (mexe-compiler)*. In order to facilitate storing profile information persistently and retrieving the information efficiently, we design a *CC/PP and UAProf* storage to store the context profiles of the devices, and a *component repository* to store the executable or interpretable codes—the codes which will be read and processed by the user agents of client devices directly—of the event-handling logics. Besides, this system offers two kinds of processors, *XUL processor* and *LGML processor*, for processing XUL and LGML documents, respectively. The following details these server components.

3.4.1. CC/PP and UAProf

It serves as a database storing the context profiles of the devices. It can keep track of the profile information of the devices beforehand. Namely, when an agent adapts an application, it can directly obtain the necessary profile from the repository without any other work. If the required profile does not exist in the database unfortunately, the agent will obtain the profile from other context providers over the Internet. In addition, the storage can cache the context profiles which have been requested by certain clients. This way can reduce the size of the messages exchange between the client devices and server. A client device has no need to send its complete profile. Instead, it sends the difference between the present hardware/software configurations and the profile stored in this storage. For example, a client boosts the memory size of his device from 32 MB to 64 MB, but other configurations of hardware utilities are not changed. In this case, the device can deliver its profile information within only the memory size in its requesting message to the server.

3.4.2. Component repository

The repository serves as a database system helping store the complied codes of the event-handling logics of the applications, and the UI DOM trees of the applications permanently. In order to retrieve the required component directly, we can store the codes and trees of an application in the repository upon programmers compete their designing of the application. As a result, it avoids processing these components when a client is requesting this application; also, it can reduce the cost of waiting for the response of application downloading.

In the repository, besides, three main tables are designed to preserve the information related to the deployed applications: (1) *application table* recording which applications have been saved, (2) *application component table* recording information related to the compiled codes of the event-handling logic segments of the applications, and (3) *application UI table* recording information related to the UI DOM trees of the applications. In brief, the storage cannot only store the deployed applications permanently, but also retrieve them efficiently.

3.4.3. XUL and LGML processor

The two processors, XUL processor and LGML processor, serve as the components which can process XUL and LGML documents, respectively. The former can validate the input XUL documents, and send the UI DOM which is generated by parsing these documents into the component repository; moreover, it has

XUL transformers to transform UI DOM trees. Similarly, the latter has the function to parse LGML DOM trees and transform these results.

The two processors are also involved in the development toolkit to support the identical functionality for logic programmers or user interface designers to process LGML and XUL documents. In this toolkit, the LGML processor can parse the input LGML documents and transform the tree generated; in addition, it can compile the result trees transformed from the transformation. However, unlike the LGML processor, the XUL processor in this toolkit has no component to transform UI DOM trees into result trees, but can validate and parse XUL documents. It is because transforming UI DOM trees is needless at the developer site. The detail will be explained in Sections 4.2 and 5.3.

3.4.4. MExE language compiler

MExE Language Compiler (mexe-compiler) can compile each of the source programs generated from the XUL transformer of the XUL processor into its specific executable code. For example, for the J2ME and PJava platforms, a Java source program (a `.java` file) generated is compiled into Java bytecodes (`.class` files). Yet in the Microsoft .NET platform, source programs a `.cs` file (C# source code) is compiled into the MSIL code (the intermediate code of .NET). The similarity of the two types of the compiled codes is that they are some kinds of the intermediate codes, which can be executed on their individual virtual machines, such as KVM in J2ME and Common Language Runtime (CLR) in the .NET platform. The difference between them lies in the format of the compiled codes.

Nonetheless, not all the generated source programs need to be compiled. For example, `.wml` (WML) and `.wmls` (WML Script) codes need no compilation, since they can be interpreted by user agents of client devices. From this point, if the source programs of Java and C# were generated, they would be compiled into executable codes by the mexe-compiler. Otherwise, when the programs of WML and WML Script are generated, they are not compiled. Table 1 summarizes the output codes of the compilations depending on different kinds of the target MExE languages. In this table, output codes can be classified into two types; one type is XUL and the other is LGML. With the different kinds of the inputs and the target languages, the types of the compiled codes will vary. For instance, if the target language is Java, the generated source program, a `.java` file, will be compiled into the Java bytecode. If the target language is WML, the generated source program, a `.wml` file, will not be compiled.

Table 1
The outputs generated from the MExE-compiler

Target MExE platform	Output code	
	XUL	LGML
WAP	No input	No input
J2ME	Java bytecode (conform to requirements of J2ME)	Java bytecode (conform to requirements of J2ME)
PJava	Java bytecode (conform to requirements of PersonalJava)	Java bytecode (conform to requirements of PersonalJava)
Microsoft CLI	C# (conform to the MSIL format)	C# (conform to the MSIL format)

3.4.5. Component manager

It provides the functions for context-aware agents of querying and obtaining the compiled or interpretable codes of the event-handling logics, and the UI DOM trees of the user interfaces. In its implementation, it takes the advantage of Java RMI interfaces for accepting the applications deployed from the user interface designer and logic programmer side. In other words, at the end of writing an XUL document and an LGML document, a programmer deploys his application into the server, using the development toolkit to process the XUL and LGML documents. Successfully processing these documents, executable and interpreted codes of LGML and a UI DOM of XUL will be generated and delivered to the server. Sections 4.2 and 5.2 will discuss these procedures in detail.

3.4.6. Context-aware service (CAS)

It principally maintains the *context aware agents*. These agents can be divided into two categories. One is the agents, which are active to perform the requesting clients' jobs. The other is the initiated and inactive agents, maintained in the agent pool for waiting clients' requests. The term ''inactive'' means that if

an agent is inactive, the thread of the agent will be paused for waiting certain request. In this system, an agent is composed of a thread and a state actually. The server will wake up a waiting agent to serve the requesting client, when receiving the client's request message. If the agent has returned the application that the client requests, the agent will be collected back to the pool, and be inactivated. In this manner, the server can recycle agents to avoid creating and initiating agents, while receiving clients' requests.

3.4.7. MExE server

MExE server, a web server, can receive and resolve HTTP request/reply messages for application downloading, and transmit the requesting messages to the DELI service to resolve and retrieve the context information of the requesting client.

From the explanation of the system components above, we consider the following scenario to observe the interaction among these system components. A user, for example, is using his Java phone to connect to our system. He selects the function to download an application. Then, his device transmits the request message to the MExE server. When the server receives the request message, it passes the request to the DELI service to retrieve the context profiles related to this client. Next, the context information is sent to the CAS server, which allocates an agent from the agent pool to serve him upon receiving the message. The agent will later obtain the UI DOM tree of the user interface and the Java bytecode of the event-handling logic, both of which comprise the requested application. For user interface manipulating, it transforms the UI DOM tree through the XUL transformer, and thereupon gets the stream result formed from the result tree with the syntax of Java language. It serializes the result into a source program (a `.java` file)—the term "serialize" means writing the content of the StreamResult object into a file which could be the source programs of the target MExE languages. Following that, it would compile the program into a Java bytecode (a `.class` file) via the mexe-compiler. Ultimately, the agent aggregates the `.class` files of the user interface and event-handling logic into an application unit `.jar` file, and then returns it to the requesting client.

This process needs two key measures: (1) context-awareness is used to get aware of the context information of the mobile devices, and (2) context-aware transformation is used to transform DOM trees into the result tree whose content syntactically conforming to the target MExE languages. The detail of the mechanisms will be explained and discussed in Sections 4 and 5.

4. XML-based programming model

4.1. LoGic Markup Language (LGML)

In order to achieve the objectives mentioned in Section 1, we can transform a program in some programming language into a program in another language: transforming programs written in Java into others written in C++. However, this way is complicated for us. Moreover, in this paper, we emphasize on transforming an XML document into another with the syntaxes of the MExE languages. Therefore, we design LGML capable of describing the common part of the computation logic of these MExE languages, and exploit the XSLT/XPath transformation mechanism to accomplish the language transformation.

Peripherally the features of LGML and XUL are different, but essentially the two languages have the same intention. LGML aims to describe the event-handling logic in the XML format. The event-handling logic is a section of an application, containing mathematical, comparison computations, and method invocations, etc. By composing these statements in the functions of an LGML document, each of these can be specified to deal with an event triggered by a widget of the user interface. A statement is composed of several expressions, each of which is a series of variables, operators, and method calls. Through LGML, a logic programmer can write compound expressions by combining expressions to construct the logic section of an application in the XML format. For example, a programmer layouts a button on the user interface of an application, and he can write the Increaser() method in the LGML document to handle the button pressed, shown as Listing 1 (other details of the LGML syntax are illustrated in the Appendix A).

Programmers can use LGML elements to write the event-handling logic. For instance, the <`lgml`:

method> element expresses a method declaration block. The <lgml:in> element contains some child elements, which are the arguments needed passing into this method. In LGML, there is an element <lgml:init> similar to the <lgml:in> element. Differently, it is used to declare and initiate the local variables in a flow-control element <lgml:for>...</lgml:for>, or a method declaration block <lgml:method>...</lgml:method>. Listing 2 demonstrates the Java source program transformed from the LGML description in Listing 1. Shortly speaking, the program of the event-handling logic expressed in LGML is capable of providing several advantages, listed as follows:

- Transforming a source into others in the target MExE languages easily
- Providing the characteristics of cross-platform and device-independence
- Neglecting which MExE environment to run applications when writing the programs of the applications.

Listing 1. A method declaration in LGML:

```
<lgml:object name="Increaser" version="1.0"
 xmlns:lgml="http://dcsw3.cis.nctu.edu.tw/ Project/Pervasive/LGML/">
 <lgml:declaration>
  <lgml:variable type="int" name="result" value="0"/>
 </lgml:declaration>
 <lgml:method name="incFive" return-type="int">
  <lgml:in>
   <lgml:variable type="int" name="num"/>
  </lgml:in>
  <lgml:action>
   <lgml:add result="result">
    <lgml:operand type="int" value="5"/>
    <lgml:operand select="num"/>
   </lgml:add>
   <lgml:return select="result"/>
  </lgml:action>
 </lgml:method>
</lgml:object>
```

Listing 2. The LGML expression of Listing 1 is transformed into the following code in Java:

```
public class Increaser{
 int result = 0;
 public int incFive(int num){
  result = 5 + num;
  return result;
 }
}
```

4.2. The programming model

Fig. 2 indicates the programming model of this system. In this model, the program of an application is separated into two divisions. It covers the XUL description for the user interface, and the LGML expression for the event-handling logic. Such a separation offers two merits. First, it makes programmers develop an application with separation-of-concern characteristic. A programmer can devote himself to writing the program of the user interface of an application without considering how to implement the event-handling logic of the application. On the contrary, the other programmer can center on writing the event-handling logic section without understanding how to layout the widgets of the user interface. Secondly, this separation provides flexibility for adapting user interfaces at runtime and efficiency for generating the codes of the event-handling logics at static time. We will take the proceeding two examples to explain what situations encountered if an application is not separated into two divisions, a user interface and an event-handling logic.

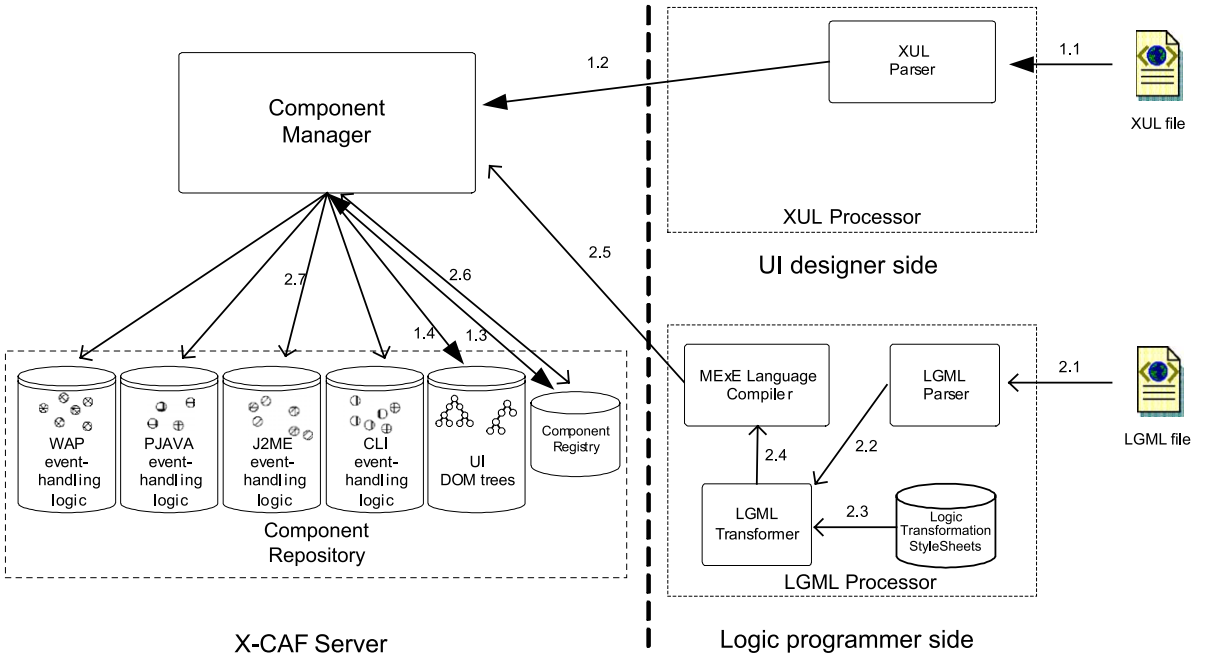

Fig. 2. The programming model and its sequence flow.

The first example is that the whole program of an application is not transformed until a client requests for the application. Under this circumstance, the transformation and compilation occur while the system receives the client's request. It leads the requesting client to spend more time awaiting the feedback than that in the separation manner. As to the second example, the transformation and compilation of the whole program have been completed before the system receives any client request. In this case, the executable code of the application can be returned to the client's device directly. However, this example causes loss of a good opportunity to adapt the application in accordance with the client's context profiles. In other words, it loses flexibility to dynamically adapt the device-sensitive widgets of the user interfaces to a variety of the devices at runtime. For instance, a programmer can write the <image> element in his XUL document to display an image on the user interface. Owing to the transformation and compilation occurring before receiving any request, the size attribute of the <image> element has been assigned some value. Perhaps, the image cannot fit the screen of certain device. Therefore, we design static-time and runtime mechanisms to deal with the user interfaces and event-handling logics separately. Section 5.2 will discuss these details.

From the previous considerations, dividing an application into a user interface and an event-handling logic can benefit our system, hence the flexibility and the efficiency. Regarding efficiency, because the event-handling logic is device independent, the LGML DOM tree can be transformed into result trees. The stream results (explained in Section 4.2) are formed from the result trees, and then serialized into the source programs of the MExE language. The programs of J2ME, PJava, and C# need to be compiled into their corresponding executable codes, whereas those of WML and WMLS require no compilation. It is because codes of WML and WMLS can be interpreted by user agents of the WAP-capable devices. The compiled or interpretable results, in the end, are delivered to the server and stored into the repository. In this way, an agent can obtain the event-handling logic from the repository directly while serving a client. Respecting flexibility, the widgets of the user interface are adapted properly, when the application that the user interface belongs to, is requested. The adaptation (explained in Section 5.2) can be accomplished by assigning suitable

values to the attributes of the device-sensitive widgets on the basis of the client's context profiles. The following procedures illustrate how to process XUL and LGML documents (as shown in Fig. 2).

The XUL document processing:

1. (1.1): When finishing writing an XUL document, a user interface designer uses the development toolkit, inclusive of the XUL processor, to validate and parse the XUL document.
2. (1.2): Within the XUL processor, an XUL parser is capable of checking well-formedness and validity for the XUL document. With no error found in the document, the XUL parser parses it into a UI DOM tree, and then transmits the tree to the component manager on this system.
3. (1.3): When the manager receives the tree, it checks whether the tree has been deployed in the repository.
4. (1.4): If there is no such a tree found in the repository, the input tree will be stored in the component repository, and its identifier will be registered into the component registry. For duplicated-deployment handling of the applications, such a number of policies as replacement of the previous one with the new one are designed. In the case, when an application is being deployed, the new one replaces the original one in the repository.

The LGML document processing:

1. (2.1): Likewise, after a logic programmer completes writing an LGML document, he uses the development toolkit, which contains the LGML processor to validate and parse the LGML document.
2. (2.2), (2.3): When the processor receives the document, it also checks the well-formedness and validity of the document. If no error were found, the document would be parsed into an LGML DOM tree and then passed to the LGML transformer.
3. (2.4), (2.5): Next, the transformer transforms the tree into the four kinds of the result trees through each transformation style sheet for the target languages. Each of stream results (`javax.xml.transform.stream.StreamResult`), formed from a result tree, is generated; besides, the stream results are serialized into source programs. Then, the mexe-compiler is notified to compile the source programs including the languages of J2ME, PJava, and C#, except that of WAP. For instance, if the serialized source program was a `.java` file, the mexe-compiler would be notified to compile the program into a Java bytecode. Finally, these executable binary codes of J2ME, PJava, and C#, and interpretable files of WML are delivered to the component manager.
4. (2.6), (2.7): When the component manager receives these programs, it examines whether these interpretable or executable programs of the application have been deployed. If not, the codes will be stored into the component repository and be registered into the registry.

In summary, the programming model separates an application into XUL and LGML documents, for describing the user interface and event-handling logic, respectively. Besides, to adapt user interfaces flexibly, we make the adaptation and transformation of the user interfaces occur when clients are requesting. On account of efficiently returning applications to the requesting client devices, in this system, programmers can compile the codes of the event-handling logic upon finishing writing LGML documents. The details concerning the transformation and adaptation will be explained in the next section.

5. Context-aware transformation

5.1. The context information

In Section 4.2, we illustrate a simple example that the size attribute of the <image> element can be assigned some suitable values for conforming to the context profiles of the target devices. To achieve the user interface adaptation, we exploit the context information inclusive of user device profile and user preference profile based on the CC/PP and UAProf framework.

By using *user device profile*, context-aware agents can be aware of capabilities of user devices. For example, if an agent was serving a requesting client, it would retrieve the client's profiles related to his context to decide what values assigned for the attributes of the device-sensitive widgets on the

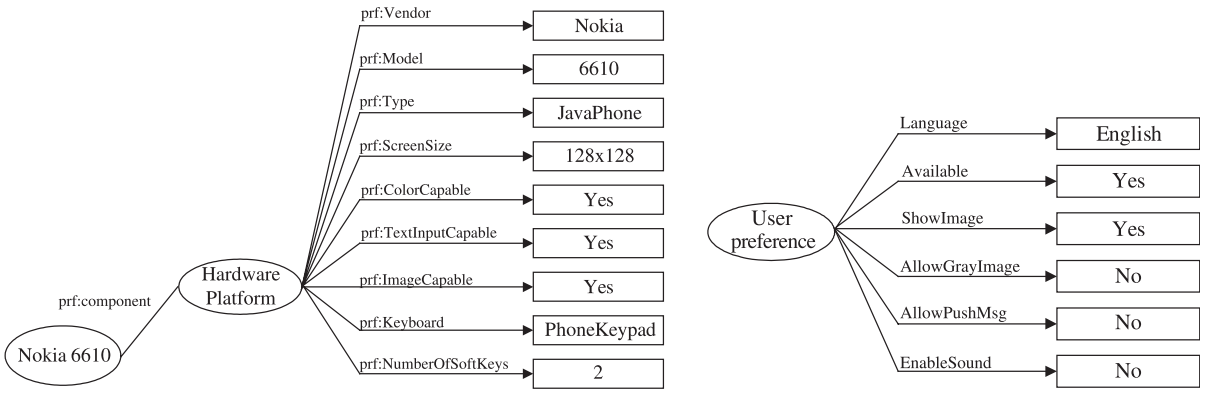

Fig. 3. On the left, an RDF graph of the user device profile, on the right the RDF graph of the user preference profile.

user interface. Fig. 3 depicts the profile of Nokia 6610 and a user preference profile in the Resource Description Framework (RDF) form. The left profile describes the several hardware capabilities of Nokia 6610: `prf:Vendor` and `prf:Model` refer its manufactory and type, respectively. Some attributes of the profile can support to adapt user interfaces, such as `ScreenSize`, `ColorCapable`, `ImageCapable` attributes. They offer the function to decide how to present an image element on the screens of the diverse devices. On the right side, it is user preference profile that describes the client's preferences. Similarly, the attributes of the profiles provide the information to aid the user interface adaptation. For example, the `Language` attribute set `English` means the language mode of the client's device is English, and the `AllowPushMsg` attribute set No stands for his unwanted interruption.

5.2. User interface adaptation

The section details how to adapt the user interfaces to the context of mobile and embedded appliances. Conceptually, assigning different values for some attributes of the device-sensitive widgets can affect different displays. For example, the `type` attribute of the `image` element could be `PNG` for J2ME MIDP, or `WBMP` for WAP-capable devices. The `length` attribute of the `TextLabel` element can be used to adjust the length of this label widget on user interfaces. Thus, by assigning suitable values by means of the context profiles, the device-sensitive widgets can be adapted to the specifications of the target devices. This adaptation occurs before transforming a UI DOM tree into a result tree. The following sequence shows this procedure:

1. (1): The Java phone transmits the request message with a header containing an URI of its profile and the profile-diff information, which is the different part from the original profile, to the MExE server.
2. (2): When receiving the client's request, the MExE server dispatches the request to the DELI service to resolve the request with a CC/PP header.
3. (3): Next, the request solver parses the request, and then transmits the result to the CAS server.
4. (4), (5), (6), (7): When receiving the request, the CAS server assigns an agent to perform the job that client requests. For example, context-aware agent 1, shown in Fig. 4, is allocated to serve this client who is using the Java phone for application downloading. The agent invokes the interfaces of the DELI service to obtain this client's context information. As a result, a Vector object (a Java abstract collection object Ref. [26]) consisting of the information of the requested profile will be created and returned to the agent.
5. (8), (9): The agent, after obtaining the context information, retrieves the executable code (the Java bytecode in this case) of the event-handling logic from the component repository depending on the device's class mark information. Thus, `.class` files comprising the event-handling logic of the application are retrieved as well as the UI DOM tree.
6. (10), (11): The agent passes the UI DOM tree to the XUL transformer. Then, the transformer transforms the tree into a result tree with the syntax of J2ME

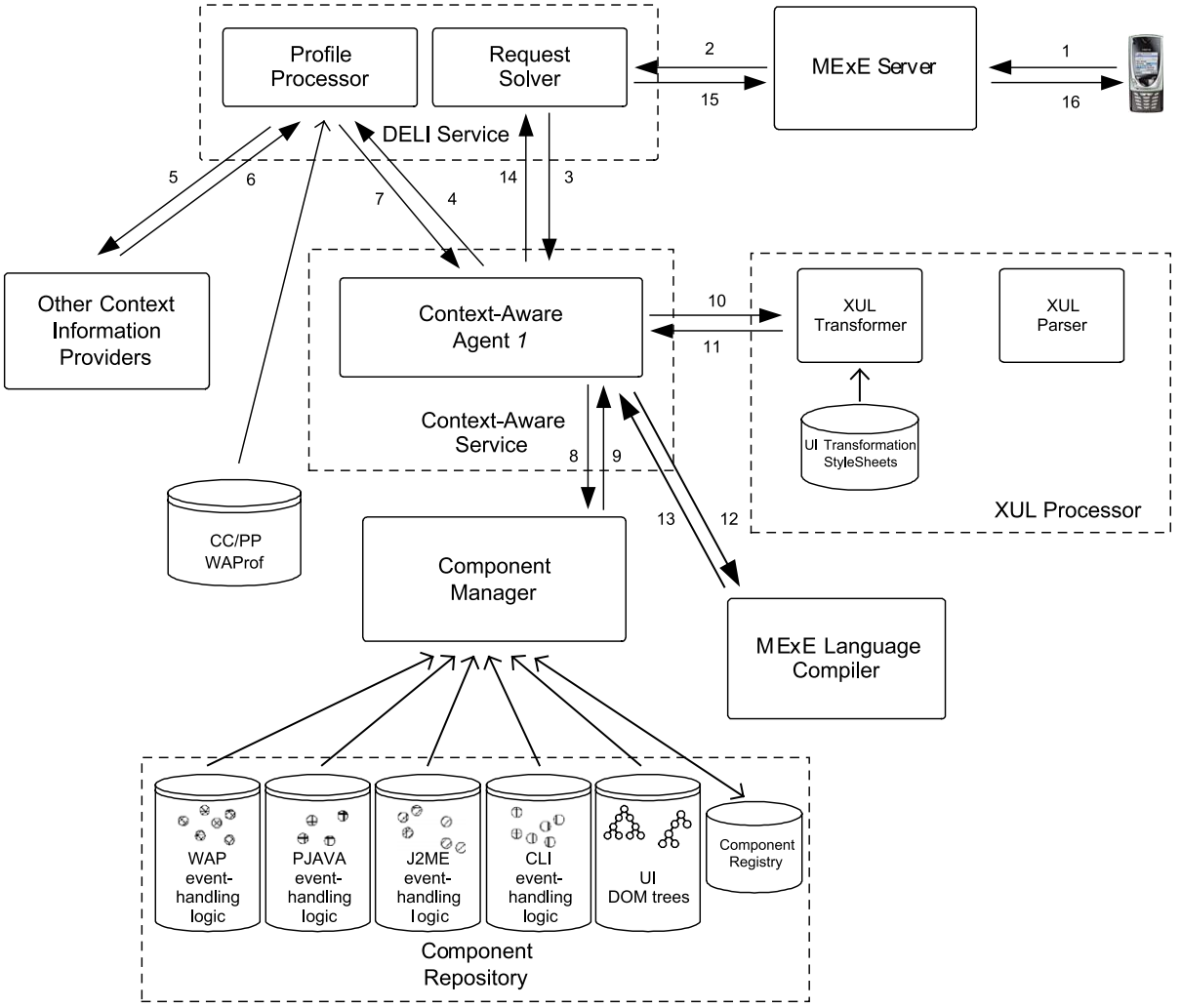

Fig. 4. Interaction diagram of the user interface adaptation.

MIDP, and assigns the suitable values to the attributes of the device-sensitive widgets by means of the client's context profiles. For example, on the basis of the `ScreenSize`, `ColorCapable`, and `ImageCapable` attributes in its profile, the value of the size attribute in an <`image`> element can be decided properly. Because the form of the user interface is a DOM tree before the agent transforms it, to assign values for the attributes is expanding the nodes which are selected through the XUL-to-J2ME transformation style sheet. Completing that, a result tree with the syntax of J2ME MIDP is generated; subsequently, source program is formed from this tree.

7. (12), (13): The agent notifies the mexe-compiler to compile the program `.java` file into a Java bytecode. A `.class` file of the user interface is generated and packed with the `.class` files of the logic of the requested application into a compressed `.jar` file.
8. (14), (15), (16): Finally, the agent delivers the executable code of the application back to the client's device.

In this procedure shown in Fig. 4, steps 10–13 are displayed in Listing 3. There are four objects, `CASAgent`, `XULProcessor`, `XULTransformer`, and `MexeCompiler`. The `getApp()`

method of the CASAgent class provide function to obtain the application which a client is requesting for. When the getApp() method is invoked, the method of the XUL processor to transform an input UI DOM tree into the result trees would be invoked. Moreover, the result is serialized into a program file, such as the .wml file for WAP, and the .java file for J2ME MIDP or PJava, etc. If the getApp() method is invoked, the process() method of the XULProcessor object will be invoked to transform the input UI DOM tree.

Within the XULTransformer class, we exploit the APIs of Xalan-Java 2.5.1 [35], which serves as the XSLT transformation engine in our system, for transforming documents. These APIs are listed as follows. The javax.xml.transform.Source interface is implemented by the javax.xml.transform.dom.DOMSource (used in Lines 19 and 31) class for passing the input DOM trees. The javax.xml.transform.Result interface is implemented by the javax.xml.transform.stream.StreamResult class (used in Lines 22 and 32) for serializing the stream result into source programs.

The XULTransformer class provides the transformToClassLang() method for transforming the input tree into the result trees with the syntax of target language. Within the method, the javax.xml.transform.Transformer (Line 34) class is used for transforming. It can transform the input tree and then generate its corresponding source program. Line 34 shows that an array Transformer[] keeps track of four types of the transformers, each of which can transform the input DOM tree into a result tree with the syntax of the specified MExE language. For example, the transformation style sheet referred by the Transformer[0] object is an XUL-to-J2ME style sheet, which has the function to transform the input DOM tree into the result tree with the syntax of J2ME MIDP.

The MexeCompiler class (mexe-compiler shown in Fig. 1) is responsible for compiling source files into executable codes. The compileTo() method of the MexeCompiler class capable of compiling programs of Java or C#. However, if the results generated by the transformer were .wml or .wmls files, compilation of the files would be unnecessary—methods of the MexeCompiler class are not invoked for compilation. In brief, if being the source program of J2ME, PJava, and C#, the generated code will require compilation.

In the implementation of compilation (Lines 45 and 50), we invoke the runtime.exec(commandLine) method to invoke external compiling process. The Java_commandLine object is an array composed of five java.lang.String objects as its arguments. Each of them sequentially corresponds to an argument of a command line in the console mode. For instance, we want execute the command c:\>javac in.java, we can set javac (java compiler) as the first argument and in.java (a file name of source program) as the second argument in the Java_commandLine array. This equals to executing c:\>java in.java under the console. In current version, our system compile .java files via javac-bootclasspath c:\mobileclass\classes.zip-target 1.1 source.java.

Some real-time issues respecting the compilation are considered as follows. Compiling the program of an application while the application is requesting provides flexibility to adapt the user interface of the application, though it costs more time to wait for the requesting client. To improve the responsiveness of returning applications to the requesting clients, we can consider the following measures: (1) preserve the executable codes or interpretable files of the applications that have been requested, in the system repository; (2) replace invoking external process with invoking some APIs.

In the first approach, when receiving the request for downloading the same application, the system determines whether or not the client device's hardware/software capabilities constraints the running of this application. If the capabilities satisfied the requirements of the application, this application would be delivered to the requesting client's device. Thus, the executable or interpretable files can be retrieved directly without any redundant compilation. The second approach reduces the time that the operating system creates and executes a new process. The reason for applying this method is that invoking external process presumably wastes more time in context-switch of two processes. Therefore, to solve it, we can compile source program through some APIs to diminish time spent in the compilation process.

Listing 3. Code sections of CAS agent, XUL processor, XUL transformer, and mexe-compiler

```
1  import java.lang.*;
2  import javax.xml.transform.*;
3  …
4  class CASAgent extends Thread{
5    …
6    public void getApp(String app_id){
7      …
8      xulProcessor.process(i, UIDOMTree, codeFileName);
9      if ( i == CLASS2 || i == CLASS3 || i == CLASS4){
10        mexeCompiler.compileTo(i, codeFileName);
11      }
12      …
13   }
14   …
15 }
16 …
17 class XULProcessor{
18   …
19   public void process(int classNum, DOMSource UIDOMTree, String
20    outputFileName){
21      …
22      result = new StreamResult(new FileOutputStream(outputFileName));
23      xulTransformer.transformToClassLang(classNum, source, result);
24      …
25   }
26   …
27 }
28 …
29 class XULTransformer{
30   …
31   public void transformToClassLang(int classNum, DOMSource DOMTree,
32    StreamResult outputSourceCode){
33      …
34      transformer[classNum].transform(DOMTree, outputSourceCode);
35      …
36   }
37   …
38 }
39 …
40 class MexeCompiler{
41   …
42   public void compileTo(int classNum, String sourceFileName){
43     …
44     if(classNum == CLASS2 || classNum == CLASS3){
45       javac_commandLine[4] = sourceFileName;
46       runtime.exec(javac_commandLine);
47     }
48     if(classNum == CLASS4){
49       cs_commandLine[4] = sourceFileName;
50       runtime.exec(cs_commandLine);
51     }
52     …
53  }
54  …
55 }
```

5.3. Code transformation

We have mentioned how to transform XUL and LGML DOM trees in the last section; however, when and where the two processes occur differs. Thus, for the transformation and compilation we design runtime and static-time mechanisms, which happen at different intervals. It can improve the responsiveness when a client is requesting for application downloading, shown as in Fig. 5. There are two types of transformations in the XUL processing flow and LGML processing flow, called *runtime XUL transformation* and *static-time LGML transformation*, respectively.

Runtime XUL transformation means that an agent transforms UI DOM trees into the result trees with the syntaxes of the four MExE languages, while receiving a request for the application which contains this user interface. It is demonstrated in Fig. 4, and its flow is shown in Fig. 5. Some source files formed from the result tree need to be compiled into executable binary codes subsequently. This phase is called *run-time compilation*. Conceptually, the scenario of adapting user interfaces is mentioned below.

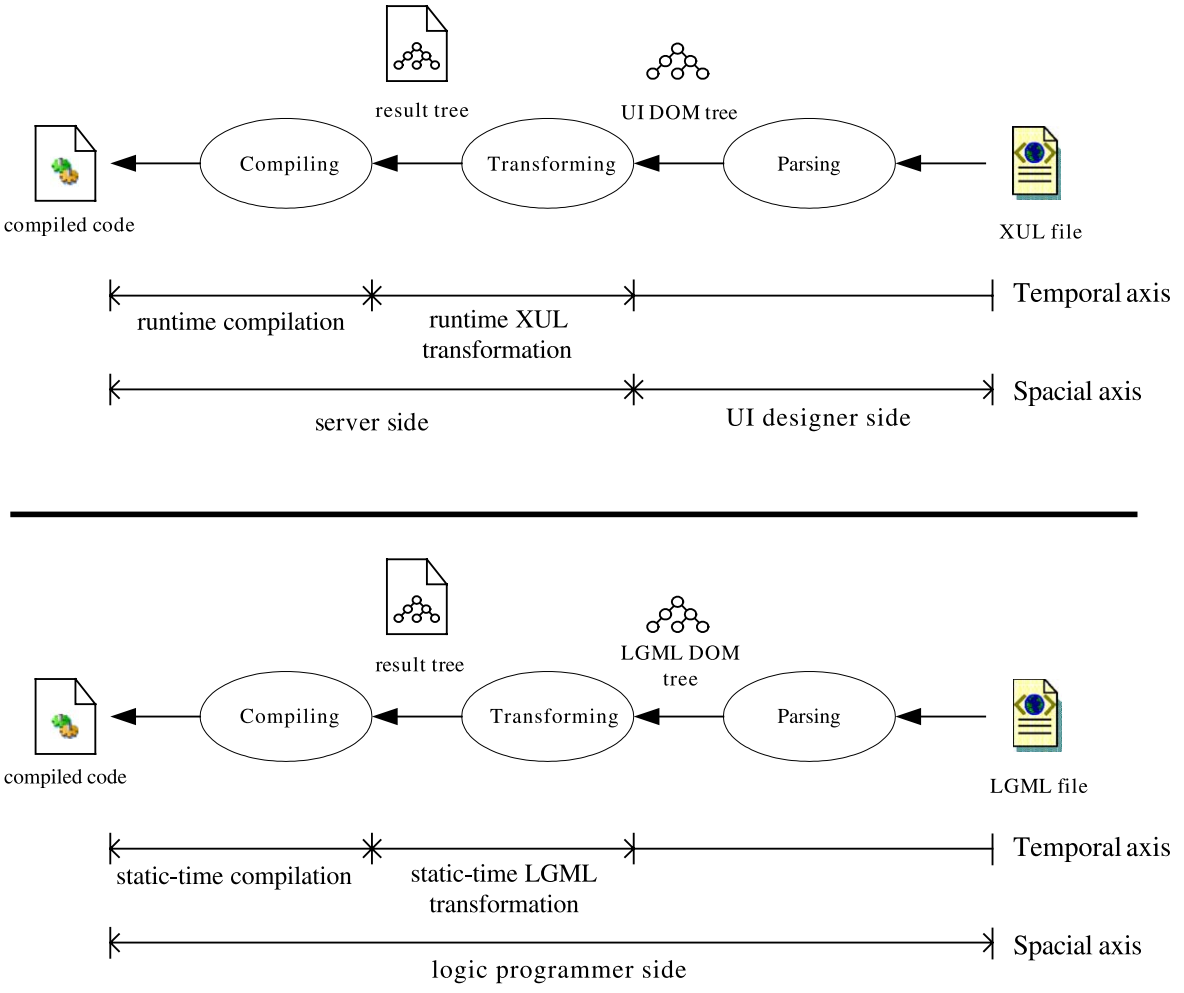

Fig. 5. The processing flows of XUL and LGML.

When receiving a request message for downloading a J2ME application, the MExE server allocates an agent performing the following job. The agent retrieves the UI DOM tree belonging to the application from the component repository; it adapts this tree and transforms it into a result tree. The steam result of the result tree will be serialized into a source program .java file next. Besides, the program is compiled into a Java bytecode. In the process, to transform the document, we apply the XSLT/XPath technique. Table 2 lists the mappings of the elements of the user interface from the XUL tags to its related J2ME MIDP expressions, which is used in the XUL-to-J2ME transformation style sheet.

Respecting LGML (Fig. 5), upon completing writing the program of the logic, a programmer can transform the LGML trees and compile the source codes generated locally, named *static-time LGML transformation*. In other words, after a logic programmer writes an LGML document, he can use the LGML parser of the development toolkit to validate and parse this document to get an LGML DOM tree. Following that, he uses the LGML transformer to transform the tree into the result trees with the syntaxes of the MExE languages. Then, the stream results of the result trees will serialized them into the source programs. Compiling the program subsequently is named *static-time compilation*. Unlike the process flow of the user interfaces, adapting the logic is not necessary in that of the logic. It is because the event-handling logic has device-independence property. Hence, the workload of transforming and compiling can be distributed on every programmer's working computer when the applications are being developed.

5.4. Code aggregation and serialization

Programming the user interface and the event-handling logic is divided. The codes of two segments need to be aggregated into a complete application finally. Namely, the related functions of the event-handling logic to execute the required actions must be performed when user interface manipulating trigger those functions. We consider the following scenario to see how to accomplish notifying the event-handling logic if the widgets of the user interfaces are triggered.

A client, for example, is manipulating his device to download a J2ME application from our system. The application manager, on the client device, transmits a request message to the MExE server. The MExE server passes this message to the DELI service after receiving. Then, the DELI service will retrieve the context profiles of the client, and forward the request to the CAS server. Upon receiving the request, the CAS server allocates an agent to perform the transformation, serialization, and compilation for the user interface of the requested application. The agent will insert information into the program of the user interface during XUL transformation. This information contains a *package name* for aggregating the user interface and event-handling logic, and *import statements* to import other packages.

A package is a named collection of classes. Packages are capable of grouping related classes and define a namespace for the classes they contain.

Table 2
The mappings from XUL tags into MIDP expressions in the XUL-to-J2ME transformation style sheet

	XUL tag	MIDP expression
Text string	<label>	javax.microedition.lcdui.StringItem
Text field	<textbox>	javax.microedition.lcdui.TextField
Item list	<listbox>	javax.microedition.lcdui.ChoiceGroup
Image display	<image>	javax.microedition.lcdui.ImageItem
Action trigger	<button>	javax.microedition.lcdui.Command
Container item	<box>	javax.microedition.lcdui.Form

For example, java.lang contains two more specialized packages, named java.lang.reflect and java.langref. The package namespace of the event-handling logic, and the package name are inserted to the program of the user interface when a DOM tree is transformed. Actually, to transform a source tree into another result tree means expanding the specified nodes of the source tree through the selection mechanism of XPath. For example, the text (shown in Fig. 4) related to the root node, are inserted when the root node of the tree is expanded at the beginning of transformation. Likewise, similar declarations (shown in Listing 5) are also inserted into the event-handling logic; nevertheless, the insertion takes place in the static-time LGML transformation. At the end of the runtime XUL transformation and compilation, the code of the user interface is packed into a .jar file with the code of the event-handling logic. Finally, application_name.jar will be generated and delivered to the requesting client device.

Listing 4. Lines are inserted to a Java source program of the user interface.

```
package org.XCAF.application_name;
…
import org.XCAF.application_name.event_handling_logic_interface;
…
```

Listing 5. Lines have been inserted in a transformed and serialized Java code of the event-handling logic.

```
package org.XCAF.application_name;
…
import  org.XCAF.application_name.user_interface_interface;
…
```

6. Conclusions and future work

To summarize, in this paper we aim at the development of a context-aware transformation framework, making two main contributions, device-independence and application adaptation. For device-independence, this framework enables developers to use the XML-based language to write applications regardless of the target MExE environments. Besides, the programming model in this paper separates the behaviors of developing applications into to two divisions, including the design of the user interface by a user interface designer and the event-handling logic programming by a logic programmer. This separation comes into the benefit so that programmers can develop applica-

tions rapidly by writing XUL and LGML documents in separation-of-concern way. In order to adapt applications efficiently and flexibly, we adapt the device-sensitive widgets at runtime by means of the context information, and generate the program of the event-handling logic at static time.

Currently, there have been several XML-based user interface description languages, such as XUL [22,23], UIML [21], AUIML [29], XIML [30], etc., as well as JavaML [31], an XML-based logic description language. So far, however, no one has integrated them to provide a full XML-based programming environment comprising XML-based description languages for the user interface and computational logic. We found out that this programming model can be extended to cope with the increasing and changing mobile execution environments through adding XSLT transformation style sheets easily. Programming using XML documents causes some related issues, yet, discussed below.

The size of the program are larger than that of programming using some general programming languages, like Java, C+, C, etc. To improve that, we will develop a toolkit, which can provide a GUI making programmers layout the user interfaces by a drag-and-draw measure readily.

As to LGML, constructing a common standard library can enhance its computational capabilities further. The common standard library can be abstracted from the intersection of the functions of these target MExE languages. In other words, the functions existing in the MExE languages have the same purpose; these functions can be retrieved and given a name in LGML. For example, to find the minimal one between two integer number `a` and `b`, we use `Lang.min(a,b)` in WML Script, but become to use `java.lang.Math.max(a,b)` in Java. Although the representation of them is different, the function remains the same. Thus, we can abstract these functions in four MExE languages to define a common standard library, which enables programmers use useful APIs in the event-handling logic. Our system can render the names of the functions to the corresponding ones of the target language specifications when transforming LGML documents. Moreover, we will apply the service invocation mechanism of XML Web Services [32] to those applications to make the applications invoke the computational functions of the services over the Internet. The enhancement can be explained in the following two reasons. First, the target languages have begun to support XML Web Services, such as Sun J2ME (J2ME supporting Web Services in Ref. [33]) and Microsoft Compact Framework [28]. The other reason is that, to add more abundant functions enriches the event-handling logic for using LGML conveniently.

Some real-time issues concerning transforming and compiling codes at runtime will be discussed in the future. We will profile the requesting flow from receiving client request to returning the required application, to find out critical performance obstructions. Also, to enhance transforming codes, we will attempt different XSLT engines and refine transformation style sheets to measure their influences on transformation. To improve compiling codes, we will experiment on several methods, such as caching mechanisms, or attempt applying the APIs of the compilation, etc., as mentioned in Section 5.2.

Furthermore, we will investigate using self-adaptation in our framework to adapt the device-sensitive widgets of the user interface by changing the types of these widgets with the various capabilities of the devices. For example, a high-quality image widget that can be shown on the screen of some device, but cannot be displayed on resource-restricted ones. Instead, the image can be replaced by a lower quality one or translated into a text string. Therefore, to approach combining the transformation and the adaptation functionalities, we will try integrating various adaptation mechanisms to use the context-aware transformation framework best.

Acknowledgements

This work was supported by Computer and Communication Research Laboratory's Program of Research and Development of Intelligent Self-adaptive Technology for Pervasive Computing under Grant T1-92016-5 in Industrial Technology Research Institute.

Appendix A. The LGML syntax and its corresponding MIDP expression

	LGML Syntax	Illustration (MIDP)

Variable declaration	`<lgml:variable name="…" type="…" value="…" />`
	`<lgml:variable name="…" type="…" >`
	`…`
	`</lgml:variable>`	

Variable declaration block	`<lgml:declaration>`
	`…`
	`</lgml:declaration>`
		LGML expression:
		`<lgml:declaration>`
		`<lgml:variable`
		`type="int" name="t" value="7" />`
		`</lgml:variable>`
		`</lgml:declaration>`
		
		Java expression:
		`int t=7;`

Variable usage	`<lgml:operand type="…" value="…" />` `<lgml:operand select="…" />`	

Assignment	`<lgml:assign arg1="…" arg2="…" />`
	`<lgml:assign arg1="…">`
	`…`
	`</lgml:assign>`
		LGML expression:
		`<lgml:assign arg1="a" arg2="b" />`
		
		Java expression:
		`a = b;`

Mathematical operators		*LGML expression:*
Addition	`<lgml:add result="…"> … </lgml:add>`	`<lgml:add result="w">`
Subtraction	`<lgml:sub result="…"> … </lgml:sub>`	`<lgml:mult>`
Multiplication	`<lgml:mult result="…">`	`<lgml:operand select="a"/>`
	`…`	`<lgml:operand select="b"/>`
	`</lgml:mult>`	`</lgml:mult>`
Division	`<lgml:div result="…"> … </lgml:div>`	`<lgml:sub>`
Remainder	`<lgml:mod result="…"> … </lgml:mod>`	`<lgml:operand select="c"/>`
		`<lgml:operand select="d"/>`
		`</lgml:sub>`
		`</lgml:add>`
		Java expression: `w = (a x b) + (c - d)`
Relational operators		*LGML expression:*
Greater-Than	`<lgml:greater-than>`	`<lgml:greater-than>`
	`…`	`<lgml:operand select="a"/>`
	`</lgml:greater-than>`	`<lgml:operand`
Lesser-Than	`<lgml:lesser-than>`	`type="int" value="5"/>`
	`…`	`</lgml:greater-than>`
	`</lgml:lesser-than>`	*Java expression:*
Equal-To	`<lgml:equal-to>`	`a > 5`
	`…`	
	`</lgml:equal-to>`	

(continued on next page)

Appendix A (*continued*)

	LGML Syntax	Illustration (MIDP)
Logic operators		
And	`<lgml:and> … </lgml:and>`	
Or	`<lgml:or> … </lgml:or>`	
Not	`<lgml:not> … </lgml:not>`	

Condition-control	`<lgml:if>`
	`<lgml:init>`
	`…`
	`</lgml:init>`
	`<lgml:test>`
	`…`
	`</lgml:test>`
	`<lgml:action>`
	`…`
	`</lgml:action>`
	`<lgml:else>`
	`…`
	`</lgml:else>`
	`</lgml:if>`
		LGML expression:
		`<lgml:if>`
		`<lgml:test>`
		`<lgml:equal>`
		`<lgml:operand select="a"/>`
		`<lgml:operand type="boolean"`
		`value="true">`
		`</lgml:equal>`
		`</lgml:test>`
		`<lgml:action> … </lgml:action>`
		`</lgml:if>`
		
		Java expression:
		`If(a == true) { … }`

Flow-control	`<lgml:for>`
	`<lgml:init>`
	`…`
	`</lgml:init>`
	`<lgml:test>`
	`…`
	`</lgml:test>`
	`<lgml:step>`
	`…`
	`</lgml:step>`
	`<lgml:action>`
	`…`
	`</lgml:action>`
	`</lgml:for>`
		LGML expression:
		`<lgml:for>`
		`<lgml:init>`
		`<lgml:variable type="int"`
		`name="i" value="0"/>`
		`</lgml:init>`
		`<lgml:test>`
		`<lgml:lesser-than>`
		`<lgml:operand select="i"/>`
		`<lgml:operand type="int"`
		`value="10"/>`
		`</lgml:lesser-than>`
		`</lgml:test>`
		`<lgml:step>`
		`<lgml:add result="i">`
		`<lgml:operand select="i"/>`
		`<lgml:operand type="int"`
		`value="1">`
		`</lgml:add>`
		`</lgml:step>`
		`<lgml:action> … </lgml:action>`
		`</lgml:for>`
		
		Java expression:
		`for(int i=0; i<10; i=i+1){ … }`

Method declaration	`<lgml:method name="…"`
	`return-type="…">`
	`…`
	`<lgml:init>`
	`…`
	`</lgml:init>`
		LGML expression:
		`<lgml:method name="incFive"`
		`return-type="int">`
		`<lgml:in>`
		`<lgml:variable type="int"`
		`name="num"/>`

Appendix A (*continued*)

	LGML Syntax	Illustration (MIDP)
	<lgml:action> … <lgml:return> … </lgml:return> </lgml:action> </lgml:method>	</lgml:in> <lgml:action> … </lgml:action> </lgml:method>
Method invocation	<lgml:call-method object="…" name="…"/> <lgml:call-method object="…" name="…"> … </lgml:call-method>	*Java Expression:* int public incFive(int num){…} *LGML Expression:* <lgml:assign arg1="a"> <lgml:call-method name="findA"> <lgml:operand select="x"/> </lgml:call-method> </lgml:assign> *Java Expression:* a = findA(x);

Object class declaration	<lgml:object name="…" version="…" xmlns:lgml="…">
	…
	</lgml:object>
		LGML Expression:
		<lgml:object name="Increaser" version="1.0" xmlns:lgml="http://dcsw3.cis.nctu.edu.tw/Project/Pervasive/LGML/">
		<lgml:method name="incFive" return-type="int">
		<lgml:in>
		<lgml:variable type="int" name="num"/>
		</lgml:in>
		<lgml:action> … </lgml:action>
		</lgml:method>
		</lgml:object>
		
		Java Expression:
		public class Increaser{
		public int incFive(int num){…}
		}

References

[1] 3GPP TS 22.057 V5.4.0. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Mobile Execution Environment (MExE); Service description, Stage 1 (Release 5), 2002, http://www.3gpp.org.

[2] WAP, http://www.wapforum.org/.

[3] PersonalJava, http://java.sun.com/products/personaljava/.

[4] Sun Microsystems. Java 2Platform Micro Edition Technology for Creating Mobile Device, Sun Microsystems, Inc., 4150 Network Circle Santa Clara, CA 95054, 2000, http://www.sum.com.

[5] JSR 118 Expert Group. JSR-000118 Mobile Information Device Profile 2.0 (Final Release), Sun Microsystems, Inc., 4150 Network Circle Santa Clara, CA 95054, 2002 May, http://www.sum.com, http://jcp.org/aboutJava/communityprocess/final/jsr118/index.html.

[6] M.H. Bulter, Current Technologies for Device Independence, Hewlett-Packard Company, Technical Publications Department HP Labs Research Library, 1501 Page Mill Road Palo Alto, CA 94304-1126, 2001 March, http://www.hpl.hp.com/techreports/, http://www-uk.hpl.hp.com/people/marbut/currTechDevInd.htm.

[7] M. Bulter, F. Giannetti, R. Gimson, T. Wiley, Device Independence and the Web, IEEE Internet Computing, IEEE Computer Society, 10662 Los Vaqueros Circle P.O. Box 3014 Los Alamitos, CA 90720-1314, 2002 October, http://www.computer.org/.

[8] R. Gimson, S.R. Finkelstein, S. Maes, L. Suryanarayana, Device Independence Principles, W3C Working Draft 18, W3C (MIT, ERCIM, Keio), in the United States, in Europe, and in Japan, 2001 September, http://www.w3.org/.

[9] W3C, Device independence working group charter, W3C (MIT, ERCIM, Keio), in the United States, in Europe, and in Japan, HP Labs Research Library, 1501 Page Mill Road Palo Alto, CA 94304-1126, http://www.hpl.hp.com/techreports/, http://www.w3.org/2002/06/w3c-di-wg-charter-20020612.html.

[10] J. Indulska, R. Robinson, A. Rakotonirainy, K. Henricksen, Experiences in using CC/PP in context-aware systems, in: M.-S. Chen, P.K. Chrysanthis, M. Sloman, A.B. Zaslavsky (Eds.), Proceedings of Mobile Data Management, 4th International Conference, MDM 2003 (Lecture Notes in Computer Science, vol. 2574), Springer, Melbourne, Australia, 2003, pp. 239–245.

[11] B.N. Schilit, N. Adams, R. Want, Context-aware computing applications, Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, IEEE Computer Society, 10662 Los Vaqueros Circle P.O. Box 3014 Los Alamitos, CA 90720-1314, 1994, pp. 85–90, http://www.computer.org/.

[12] H. Maruyama, K. Tamura, N. Uramoto, M. Murata, A. Clark, et al., XML and Java Second Edition: Developing Web Applications, Addison-Wesley, 75 Arlington Street, Suite 300 Boston, MA 02116, 2002, http://www.awprofessional.com/.

[13] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M.H. Butler, L. Tran, Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies. W3C Working Draft, W3C (MIT, ERCIM, Keio), in the United States, in Europe, and in Japan, 2003 March, http://www.w3.org/, http://www.w3.org/TR/CCPP-struct-vocab/.

[14] L. Suryanarayana, J. Hjelm, CC/PP for content negotiation and contextualization, Lecture Notes in Computer Science, vol. 1987, Springer Verlag, Heidelberg, 2001.

[15] M.H. Butler, Implementing Content Negotiation using CC/PP and WAP UAProf. External Technical Report HPL-2001-190, 2001, http://www.hpl.hp.com/techreports/2001/HPL-2001-190.html.

[16] H. Ohto, J. Hjelm, CC/PP Exchange Protocol Based on HTTP Extension Framework. W3C Note, W3C (MIT, ERCIM, Keio), in the United States, in Europe, and in Japan, 1999 June, http://www.w3.org/, http://www.w3.org/TR/NOTE-CCPPexchange.

[17] D. Brickley, R.V. Guha, B. McBride, RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, W3C (MIT, ERCIM, Keio), in the United States, in Europe, and in Japan, http://www.w3.org/, http://www.w3.org/TR/rdf-schema/.

[18] O. Lassila, R.R. Swick, Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, W3C (MIT, ERCIM, Keio), in the United States, in Europe, and in Japan, 1999 February, http://www.w3.org/, http://www.w3.org/TR/REC-rdf-syntax.

[19] WAP Forum, User Agent Profiling Specification, Open mobile Alliance, Ltd., Management Office 2570 West El Camino Real, Suite 304 Mountain View, CA 94040-1313 USA, 2001 October, http://www1.wapforum.org, http://www1.wapforum.org/tech/terms.asp?doc=WAP-248-20011020-a.pdf.

[20] M.H. Butler, DELI: A DElivery context LIbrary for CC/PP and UAProf. External Technical Report HPL-2001-260, HP Labs, 2002, http://delicon.sourceforge.net/.

[21] M. Abrams, C. Phanouriou, A.L. Batongbacal, S.M. Williams, J.E. Shuster, UIML: An Appliance-Independent XML User Interface Language, http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html.

[22] N. Deakin, XUL Tutorial, The Mozilla Organization, 2002 July, http://www.mozilla.org/mozorg.html, http://www.xulplanet.com/tutorials/xultu/.

[23] XUL Language Specification, http://www.mozilla.org/xpfe/languageSpec.html.

[24] Sun Microsystems, Java Servlet technology, http://java.sun.com/products/servlet/.

[25] Microsoft, Microsoft .NET, Microsoft Corporation, Microsoft Chicago and Microsoft Technology Center 77 W. Wacker Suite 2300 Chicago, IL 60601, 2002, http://www.microsoft.com/, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmapnet/html/mapintronet.asp.

[26] J. Zukowski, Java collections, Apress, 2560 Ninth St., Ste. 219 Berkeley, CA 94710, 2001, http://www.apress.com/.

[27] Microsoft Mobile Web Forms, http://samples.gotdotnet.com/mobilequickstart/(mgk4rd2jnyo1zm55tgnot02p)/Default.aspx.

[28] NET Compact Framework, http://samples.gotdotnet.com/quickstart/compactframework/.

[29] R.A. Merrick, et al., AUIML: An XML Vocabulary for Describing User Interfaces, IBM United Kingdom Limited, UK Head Office PO Box 41, North Harbour Portsmouth Hampshire, PO6 3AU, 2001 May, www.uk.ibm.com, http://www.belchi.be/download/merrick.pdf.

[30] A. Puerta, J. Eisenstein, XIML: A Common Representation for Interaction Data, Proceedings of the Sixth Intelligent User Interfaces Conference (IUI 2002), San Francisco, California, USA, 2002 January.

[31] G.J. Badros, JavaML: A Markup Language for Java Source Code, Proceedings of Ninth International World Wide Web Conference, Amsterdam, 2000 May.

[32] V. Chopra, Z. Zoran, G. Damschen, et al., Professional XML Web Services, Wrox. Press, Wrox, Customer Care Center 10475 Crosspoint Blvd. Indianapolis, IN 46256, 2001 September, http://www.wrox.com/.

[33] J. Ellis, J2ME™ Web Services Specification, Sun Microsystems, Inc., 4150 Network Circle Santa Clara, CA 95054, 2003 July, http://www.sum.com.

[34] J. Clark, XML Path Language, W3C Recommendation, W3C (MIT, ERCIM, Keio), in the United States, in Europe, and in Japan, 1999 November, http://www.w3.org/, http://www.w3.org/TR/xpath.

[35] Xalan-Java 2.5.1, http://xml.apache.org/xalan-j/.

Tzu-Han Kao was born on December 20, 1976 in Taichung, Taiwan, Republic of China. He received his BS degree in Computer Science and Information Engineering from Chung Hua University, Taiwan, in 2000. He is now a PhD candidate in Department of Computer and Information Science of National Chiao Tung University. His current research interests include Context-aware, Ubiquitous, and Pervasive Computing, Internet Technologies, Data Mining, and Machine Learning.

Shyan-Ming Yuan was born on July 11, 1959 in Mauli, Taiwan, Republic of China. He received his BSEE degree from National Taiwan University in 1981, his MS degree in Computer Science from University of Maryland, Baltimore County in 1985, and his PhD degree in Computer Science from the University of Maryland College Park in 1989. Dr. Yuan joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in October 1989. Since September 1990, he has been an Associate Professor at the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. He became a Professor in June 1995. His current research interests include Distributed Objects, Internet Technologies, and Software System Integration. Dr. Yuan is a member of ACM and IEEE.