Interactions Model and Code Generation for J2ME Applications

Davide Carboni[1], Stefano Sanna[1], Sylvain Giroux[2], and Gavino Paddeu[1]

[1] CRS4, Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna,
VI Strada OVEST Z.I. Macchiareddu,UTA, CA – Italy.
{dadaista,gerda,gavino}@crs4.it
[2] Dep. of Mathematics and Computer Science, University of Sherbrooke,
Sherbrooke – Canada.
sylvain.giroux@dmi.usherb.ca

Abstract. The Java2 Micro Edition platform can really be considered an emerging standard for new generation embedded software. This article introduces a practical methodology aimed to automatically generate a software prototype starting from an abstract description which defines the dialogue between the user and the application by means of a device independent and abstract description. We will show how an agenda application for cellular phones can be described by means of a visual language called PLANES and present how the personal agenda prototype is implemented by an appropriate generation tool.

1 Introduction

With an army of 2.5 million programmers all around the world, ready to conceive and develop new intelligent applications for mobile devices, and the commitment of some sector giants (such as Motorola and Nokia), the Java2 Micro Edition (J2ME) platform [3] can really be considered an emerging standard for new generation embedded software. Developing new applications in J2ME is not as easy as developing for desktop computers or for the web. Some difficulties to overcome are:

- Integrated development environments (IDE) for J2ME provide less functionality than their counterparts for Java2 Standard Edition (J2SE).
- Application Programming Interfaces (API) for J2ME differ from their corresponding API for J2SE. Programmers need to learn them and adapt their code.

This paper describes a method and the implemented related tools for automatic code generation of the GUI and of the code that manages the sequences of forms and dialogs on the screen for small mobile devices as personal digital assistants, cellular phones etc. The dialogue between a user and the application is specified with a visual language. A device independent description is then produced. This description is used for code generation on the J2ME platform.

F. Paternò (Ed.): Mobile HCI 2002, LNCS 2411, pp. 286–290, 2002.

2 Personal Agendas as Test Applications

The main task in a personal agenda is managing a set of appointments. From the viewpoint of GUI, this description is too abstract and needs to be refined towards more detailed and concrete steps. Concretely the agenda will initially provide a choice between two options:

- Create and insert a new appointment.
- Consult the whole list of appointments.

These choices then lead the user to other choices or sequences of more concrete tasks. For instance, the creation of a new appointment prompts the user with a form, then asks for a confirmation of the data and finally waits for data to be written in the database. We made two assumptions on the use of target devices:

1. Due to screen restrictions, it is not possible to work with more than one window at a time, thus the dialog is constrained in well defined pathways along a tree of choices and sequences of tasks.
2. The user should have the possibility to escape the current task at any time and return to the main menu by means of hot keys bound with physical buttons in the device.

With respect to these assumptions, the next section describes a practical approach to model the interaction between the user, the application, and the runtime environment.

3 Task Modeling with PLANES

Task modeling with PLANES involves a high-level description of choices and sequences of activities. Such a description is formalized by means of a visual language whose structure is basically a tree. The lexical elements of such a language are grouped into three classes: abstract tasks, concrete tasks and reification models. In the agenda example, creating a new appointment is a sequence of three tasks:

1. Form-filling: the user creates a new data object with a given structure; this object is copied into the current context.
2. Confirmation, the user is asked to provide his authorization to proceed to the next task. We coin such tasks as “shield” tasks.
3. Write operation, the data object is copied from the context to the system.

Tasks represent activities at different levels of abstraction. For instance, a task representing the insertion of a new appointment into the personal agenda must be considered an abstract task which will be reified by one or more concrete tasks.

The user is indeed an actor able either to create brand new data objects or modify existing objects by form-filling. A form is a tool which allows the user to create a data object with a given structure and write temporarily this object into the current context in order to make it available for the next task. The context acts as a container which stores the data during the interaction. Data objects have a structure composed by a number of primitive data types such as strings, numbers, booleans, enumerations and lists in a way similar to data structures used in procedural and object oriented programming languages.

To describe tasks and structure the dialogue, PLANES contains the following language elements:

Abstract Task: represents a high-level activity; for instance, "Inserting a new appointment" is an abstract task which must be reified by a sequence of concrete tasks.

Application Task: an application task represents an interaction with the user by means of a form. A task is completed once the user fills the form with required input.

Shield Task: a Shield Task represents an interaction in which the user must confirm his intention to proceed to the next task in a sequence. Typically, this task prevents the user from making accidental changes in the database.

Data Task: they represent an interaction between the application and the system in which a data object is read, written or deleted from/to the system and copied into/from the current context.

Reification Models: each abstract task must be reified by a set of detailed tasks. A Reification Model collects those tasks and defines the control flow between tasks. We have identified two types of reification models: Sequence and Choice.

4 The Visual Editor and J2ME Code Generator

PLANES is provided with a visual editor which allows the designer to build a new model and save it serialized in XML. In addition to these essential functionalities, the editor can perform a simulation of the application's behavior by means of dialogs and menus that follow the structure defined by the model. The generation process is depicted in figure below:

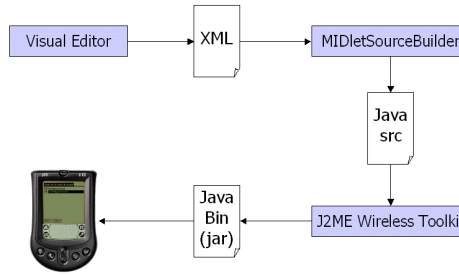

Fig. 1. The creation process of the prototype from the abstract model to the application.

The MIDletSourceBuilder reads the XML file and, starting form the root-level abstract task, it performs a depth-first search and generates for each node a corresponding Java source file. The main benefit of using an automatic tool for J2ME application development is for providing a complete composition and chaining of the GUI pieces. The tool described in this paper differs from commercial tools for application development which assist the programmer with visual composition of graphics widgets and project management. Visual Editor and MIDletSourceBuilder, instead, focus on the structure of interaction with end user.

5 Results for the Personal Agenda

This section depicts how we modeled the personal agenda in the visual editor. Figure 2 shows some screen shots (1) to (4) captured from the emulator and representing, in succession, the graphical components that lead the user along the creation and insertion of a new appointment in the personal agenda. Starting from the root node, the application shows a selection menu (1), from which the user selects the creation of a new appointment. Next form (2) contains the fields where the user can insert the details (3) of new appointment. When data have been inserted, the application asks the user (4) to confirm data writing on system database. The "write task" is the only concrete task the developer has to implement. In Figure 3, an excerpt of the XML code is shown.

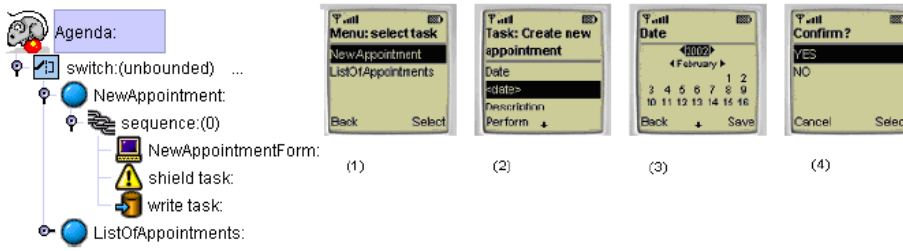

Fig. 2. Agenda model in the Visual Editor and screenshots of the prototype running on the J2ME emulator.

```
<?xml version="1.0"?>
<Interaction>

    ...snip...

    <AbstractTask ID="0_M_0" ancestorID="0_M" name="NewAppointment" type="abstract">
        <Model ID="0_M_0_M" ancestorID="0_M_0" type="sequence" description="A sequence model" repeat="0">
            <ModelItem taskID="0_M_0_M_0"/>
            <ModelItem taskID="0_M_0_M_1"/>
            <ModelItem taskID="0_M_0_M_2"/>
        </Model>
    </AbstractTask>
    <ConcreteTask ID="0_M_0_M_0" ancestorID="0_M_0_M" name="NewAppointmentForm" type="application" />
    <ConcreteTask ID="0_M_0_M_1" ancestorID="0_M_0_M" name="shield task" type="shield" />
    <ConcreteTask ID="0_M_0_M_2" ancestorID="0_M_0_M" name="write task" type="write" />
...snip....
</Interaction>
```

Fig. 3. An excerpt of XML code describing the sequence of tasks for the "Insertion of a new appointment".

6 Conclusion and Future Work

In this paper, we have sketched a visual language (PLANES) and an automatic code generator for prototyping new J2ME-MIDP applications. This process allows

programmers to get a working prototype without the burden of writing all code lines for event handling and user interface creation and let them spend their efforts to customize the prototype and transform it in a real application. We have tested PLANES with a personal agenda application, which is a service largely available in many cellular phones. Application prototyping with a visual tool has demonstrated its efficacy both for showing and discussing application's functionalities with the customer, and for the quality of the generated source code which results untangled, easily readable, and customizable.

We believe that the adoption of model-based techniques is a valid support for software prototyping and application development not only for small devices but for web and desktop applications as well.

7 Related Works

PLANES stands at the crossroad between automation methodologies for embedded software prototyping and model based user interface design. Pros and cons of such methodologies are discussed in [2], [5], [6]. The tool which most inspired our work is ConcurTaskTree [4]. Another tool, MASTERMIND [1], [7], defines a technique based upon the definition of three distinct models, one for the functionalities available by means of the user interface, another defines the presentation structure in terms of widgets and a last model defines the dialog in terms of end-user interaction and how these affect the presentation and the application.

PLANES is not in competition with those methodologies, that represent milestones in HCI research, but rather, it is an attempt to experiment some of model-based design concepts and ascertain how they can be applied to the rapid prototyping of software for embedded systems and mobile phones.

References

1. T. P. Browne et al. Using declarative descriptions to model user interfaces with MASTERMIND. In F. Paterno and P. Palanque, editors, Formal Methods in Human Computer Interaction.
2. Janssen C., Weisbecker A., Ziegler J.: Generating user interfaces from data models and dialogue net specifications. In: Ashlund S., et al. (eds.): Bridges between Worlds. Proceedings InterCHI'93 (Amsterdam, April 1993). New York: ACM Press, 1993, 418-423.
3. Sun Microsystems, MIDP Specification (Final V1.0), 2001 (available at http://java.sun.com)
4. Paterno, F., Model-Based Design and Evaluation of Interactive Applications. Springer Verlag, ISBN 1-85233-155-0, 1999.
5. Puerta A.R. 1993. The Study of Models of Intelligent Interfaces. In Proceedings of the 1993 International Workshop on Intelligent User Interfaces. Orlando, Florida, January 1993, pp. 71–80.
6. Schlungbaum E., Elwert T.: Automatic user interface generation from declarative models. In: Vanderdonckt J. (ed.): Computer-Aided Design of User Interfaces. Namur: Presses Universitaires de Namur, 1996, 3-18.
7. Szekely, P. et.al. Declarative interface models for user interface construction tools: the MASTERMIND approach. In Proc. EHCI'95 (Grand Targhee Resort, August 1995).