View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by P-arch

Interactions Model and Code Generation for J2ME Applications

Davide Carboni, Stefano Sanna, Sylvain Giroux*, Gavino Paddeu

CRS4, Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna, VI Strada OVEST Z.I. Macchiareddu,UTA, CA – Italy. *davide.carboni@crs4.it*

(*) Dep. of Mathematics and Computer Science, University of Sherbrooke, Sherbrooke – Canada s*ylvain.giroux@dmi.usherb.ca*

The Java2 Micro Edition (J2ME) platform can really be considered an emerging standard for new generation embedded software. The extent of business opportunities in the domain of small-device software is foreseen to become huge in few years. Though, developing new applications in J2ME is not as easy as developing for desktop computers or for the web. 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, Nokia, Siemens and RIM), the Java2 Micro Edition (J2ME) platform [MIDP] can really be considered an emerging standard for new generation embedded software. In fact, J2ME is at crossroad between Internet services and wireless networks and, as well as business opportunities in this sector will attract new investments from service providers and from appliances manufacturers, J2ME is foreseen to play an important role. Besides, J2ME can be a valid development platform for consumer appliance software like VCR displays, washing machine control software and more in general all those systems that need to provide a simple but effective interaction with users. The extent of business opportunities in those domain is foreseen to become huge in few years [GARTNER 2002]. In spite of this enthusiastic picture, 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 learning them and adapt their code.
- Testing is mainly performed on emulators that simulate network connections through Ethernet cards which do not allow taking into account wireless

network peculiarities such as limited bandwidth and fortuitous disconnections due to signal failures.

A significant percentage of code written for new applications is related to the implementation of graphical user interfaces (GUI) [SUTTON 1978]. 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. As it is case in modern IDEs, e.g. VisualAge or Jbuilder, the generated code stands as a skeleton the programmer may adapt. The developer may insert customization and personalization code that obviously cannot be produced by a automated tool. Besides, our tool provides an API that allows the communication between the generated code and the underlying system. For instance, the developer must implement such interfaces if the generated code must write or read data from a database or if data must be sent towards a remote system through the network interface.

In section (§2) we use personal agendas to highlights the means features of our system. Agendas are services available in many cellular phones. Our visual language called PLANES is then used to describe this application in terms of its interactions with the user and with the underlying system(§3). In the final part of the paper we detail the automatic generation of J2ME source code (§5) and we sketch the implementation of a personal agenda thanks to the generation tool (§6).

2. Personal agendas as test applications

A personal agenda is a service widely available in cellular phones and digital organizers. Thus, it can be considered as a valid test bed to assess the pros and cons of our prototyping system for mobile software. In this section, we present the main features of interactions with agendas we will focus on in the rest of the paper.

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 ask for a confirmation of the data and finally wait for data to be written in the database.

We are aimed at small portable mobile device such PDAs and cellular phones. These devices usually have very limited input/output resources. Thus we made two assumptions on the use of such 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. To understand the semantics of the description we need to introduce some concepts: user, data, system and context. The role of these concepts can be sharpened by means of an example: assume that the user wants to insert a new appointment and has already selected the menu option “Create New Appointment”. At this point, the application shows a form whose structure corresponds to a precise data structure composed, for instance, by description, date, time and place. The user fills the form and proceeds to the following dialog which asks the user to confirm data insertion. Once the operation is confirmed, the data are written in the database. Thus, 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.

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 structure used in procedural and object oriented programming languages. After the confirmation, the write task is performed by reading data objects from the context and sending them to another actor, the system. The system represents the runtime environment where the application runs and its interface is modeled by an appropriate protocol defined by means of a Java interface.

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

“Create a new appointment” ≡ do sequentially (fill the form, wait for user confirmation, insert data into database)

This example allows us to elicit some important considerations:

- “Create a new appointment” is an abstract task.
- “Do sequentially” is the modality of reification.
- The tasks in the sequence are concrete; some of them require the user intervention while others require the interaction with the system.

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

Abstract Task: an abstract task represents a high-level activity. It always contains a reification model.

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

Shield Task: a Shield Task is a dialog box where the user is asked to confirm his intention to proceed to the next task in a sequence. Such a type of interaction is aimed to prevent accidental execution of operations that can affect data and allows the user to move backward in a sequence of tasks or to cancel the entire sequence returning to the ancestor node in the task tree.

Read Task: a Read Task represents an interaction between the application and the system in which a data object is read from the system and copied into the current context.

Write Task: a Write Task is an operation which transfers the data resident in the current context to the system.

Delete Task: a Delete Task is an interaction which requests to remove a given data from the system.

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

1. **Sequence**: a sequence contains one or more tasks which are executed sequentially. A sequence is tightly coupled with a context in which data objects are stored and used by the tasks belonging to the sequence. A sequence has a property that defines how many iterations it needs to be executed before the sequence is considered completed. Due to the fact that any abstract task has one, and only one, reification model, the completion of a sequence causes the completion of the associated abstract task.
2. **Switch**: a switch reification model represents a choice the user has to make. For instance, the Personal Agenda starting page could show various different options like “create a new appointment” or “browse existing appointments”. Each option in a switch is indeed a task either abstract or concrete. In addition to explicit options, the user should be able to choose either returning upward in the node hierarchy or returning directly to the root of tree.

An abstract task reified by means of a switch reification model reach its completion as soon as one of the tasks collected in the switch reaches its completion. A switch can be repeated several times. For instance, the iterations of the Personal Agenda initial

menu are unbounded and it always re-proposes the choice unless the user decides to quit the application.

4. The visual editor

PLANES is provided with a visual editor which allows the designer to build a new model and save it serialized in XML. The XML serialization is also useful to export the model to the code generator. The visual editor is a Java stand-alone application (Figure 1) with a Swing graphical user interface. It provides a palette of tools to insert, cut, copy and paste tasks, switches and sequences.

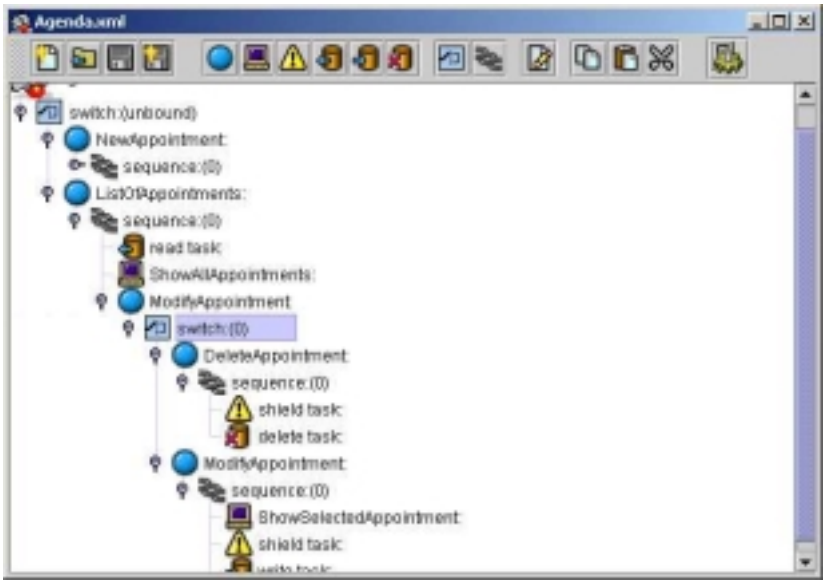

Figure 1: PLANES Visual Editor

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. Once the tasks are specified, the J2ME code can be generated and then customized if necessary.

5. The J2ME code generator

This section describes the code generation process which translates a XML description of a PLANES model into Java source files specifically tailored for the J2ME platform. All generated Java files are compiled and executed in the Sun Microsystems Wireless Toolkit 1.0.3 [J2MEWT]. The generation process is depicted in (Figure 2).

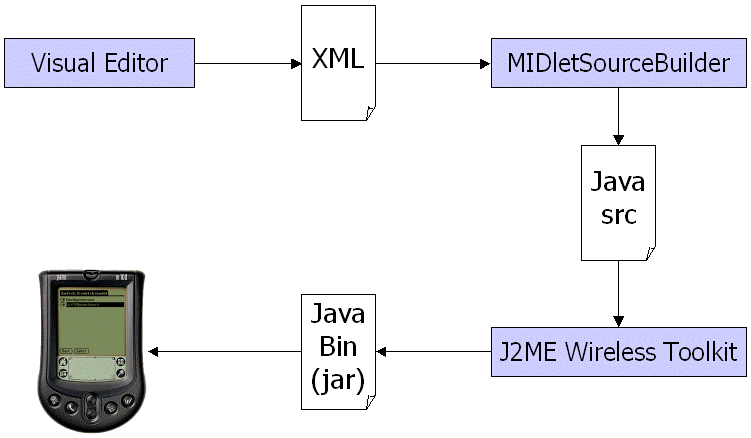

Figure 2: The creation process of the prototype, from the abstract model to the application.

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. Indeed, coding difficulties arise in the development of MIDP applications [AYERS 2001]:

- Graphic context is available only to the running MIDlet[1]. Therefore, application designer should provide a way to propagate the display context to graphical components;
- MIDP API provides a poorer event-handling mechanism than AWT/Swing. Moreover, there are several limitations regarding widgets composition;
- MIDP API specification gives no guarantee about how a widget will be rendered on a specific device. For instance, depending on the actual device, Commands[2] could be displayed in form of soft-button functions or icons on the display.

These issues can be addressed through well-defined patterns in the application design. Our work takes advantage of these patterns in the automatic code generation of prototypes. The MIDletSourceBuilder hides all details of event-handling, form chaining and GUI creation, and let the programmer focus on the customization of single “nodes” of the model.

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. This approach results in abstracting and decoupling from the actual device and untangled source code easy to maintain and personalize.

[1] MIDlets are the basic executable objects in the J2ME application manager.
[2] Commands are action objects triggered by user interaction.

6. Results for the personal agenda

This section depicts how we modeled the personal agenda in the visual editor (Fig. 3). The model is serialized and stored in a XML file and then imported by the code generator that performs the generation of Java source code. Java files are compiled and executed in the emulator. Figure 3shows 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. This is the only concrete task the developer has to implement.

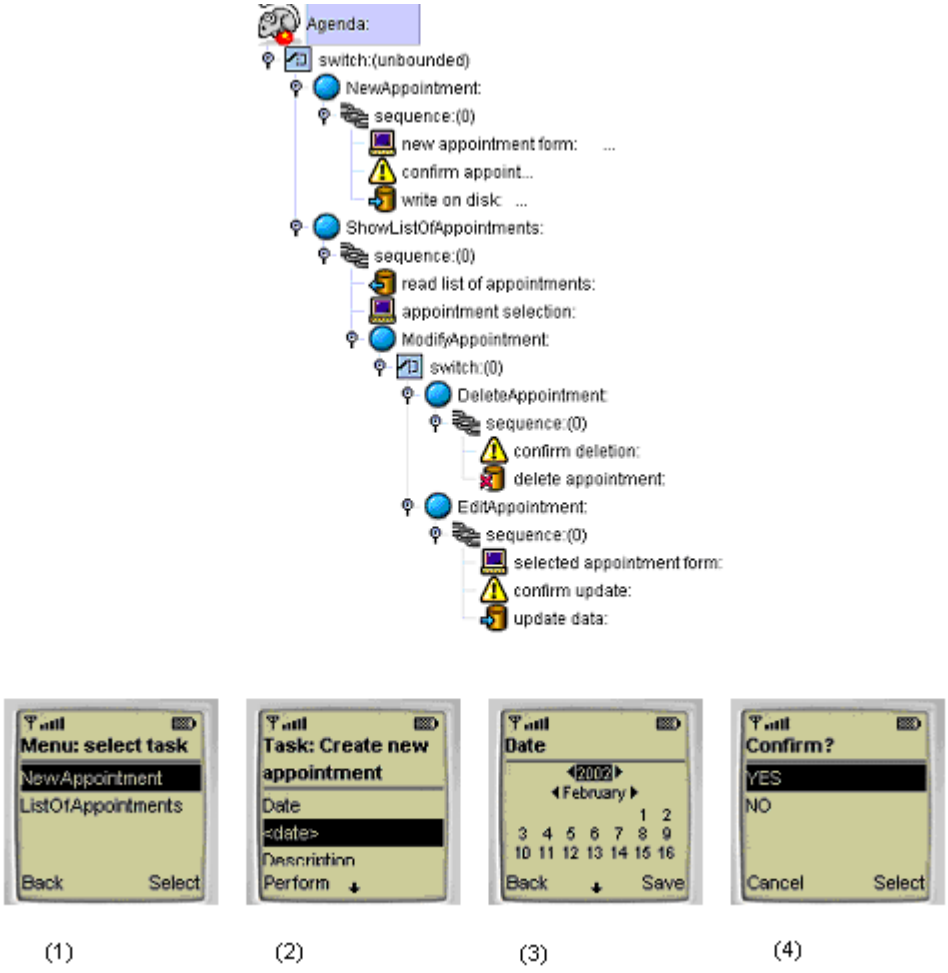

Figure 3: Agenda model in the Visual Editor and screenshots of the prototype running on the J2ME emulator

7. Conclusion and future work

Because of growing interest in small-device applications, the Sun J2ME-MIDP platform is going to draw the attention of the Java community. Nevertheless, the use of such a poor API may be uncomfortable to the majority of programmers who are used to develop their applications with the Java2 Standard Edition.

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. Thus, we have defined the interaction model with the Visual Editor, generated the source code and finally tested the prototype on Sun's Wireless Toolkit emulator. 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.

Next steps will consist in making a closer evaluation of this approach in real-world software development. We believe that the adoption of model-based techniques is a valid support for software prototyping and applications development not only for small device but for web and desktop applications as well.

8. 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 [PUERTA 1993] [SCHLUNGBAUM 1996] [JANSSEN 1993]. We think that model-based design techniques would positively affect the development of new application for the J2ME platform. Among model-based methodologies, we want to recall GOMS [CARD 1983] [JOHN 1996] (Goals, Operators, Methods and Selection rules) that is considered a valuable tool in real-world HCI design. Another tool which defines hierarchies and relationships among tasks is ConcurrentTaskTree [PATERNO 1999]. MASTERMIND [BROWNE] [SZEKELY 1995] 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 instead it should be considered 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.

Due to the fact that mobile phone models largely differ from one another in their physical characteristics, designing new applications is difficult and error-prone. Thus, following an approach already discussed in the field of pervasive computing [BANAVAR 2000], the main objective of this work is to experiment a process which allows the developer to design his application in terms of tasks and sub-tasks rather than in terms of components of the graphical user interface toolkit provided with the J2ME platform.

9. References

[AYERS 2001] Danny Ayers et al., Professional Java Mobile Programming, 2001, Wrox Press, ISBN 1861003897

[BANAVAR 2000] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski. An Application Model for Pervasive Computing. In Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, August 2000.

[BROWNE] 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.

[CARD 1983] Card, S. K., Moran, T. P., & Newell, A. (1983). The psychology of human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum Associates.

[GARTNER 2002] Gartner. New Mobile Technologies Set to Arrive in 2002, 1 February 2002. Nick Jones – Research. Note Number: AV-15-2487

[J2MEWT] Sun Microsystems, *Wireless Toolkit*, (application and documentation, available at http://java.sun.com/products/j2mewtoolkit/).

[JANSSEN 1993] 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.

[JOHN 1996] John, B. E. & Kieras, D. E. (1996a). Using GOMS for user interface design and evaluation: Which technique? ACM Transactions on Computer-Human Interaction, 3, 287-319.

[MIDP] Sun Microsystems, *MIDP Specification* (Final V1.0), 2001 (available at http://java.sun.com)

[PATERNO 1999] Paterno, F., Model-Based Design and Evaluation of Interactive Applications. Springer Verlag, ISBN 1-85233-155-0, 1999.

[PUERTA 1993] 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.

[SCHLUNGBAUM 1996] 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.

[SUTTON 1978] SUTTON, J., AND SPRAGUE, R. A study of display generation and management in interactive business applications. Tech. Rep. RJ2392 (31804), IBM San Jose Research Laboratory, San Jose,Calif., Nov. 1978.

[SZEKELY 1995] Szekely, P. et.al. Declarative interface models for user interface construction tools: the MASTERMIND approach. In Proc. EHCI'95 (Grand Targhee Resort, August 1995).