

W2000 MEETS J2ME FOR THE FAST PROTOTYPING OF MOBILE WEB APPLICATIONS

Luciano Baresi

Dipartimento di Elettronica e Informazione
Politecnico di Milano
piazza L. da Vinci, 32
20133 Milano, Italy
email: baresi@elet.polimi.it

Luca Mainetti

Dipartimento di Ingegneria dell'Innovazione
Università degli Studi di Lecce
via per Monteroni
73100 Lecce, Italy
email: luca.mainetti@unile.it

ABSTRACT

Mobile devices are motivating a new family of Web applications. Limited sources, like displays, mice, keyboards, and memory, heavily impact the client-side capabilities of the application. This is why *mobile Web applications* must be re-modeled to take into account the peculiarities of the new devices.

This paper does not propose yet another “general purpose” modeling framework and does not even discuss the problems with modeling multi-device applications. It assumes that mobile Web applications will not mainly be used for browsing data, but navigation will be the means to access services. Given this hypothesis, the paper proposes μ W2000 as light-weight version of W2000 for this class of applications and a rule-based approach for fast prototyping the client side of these applications through self-contained J2ME (Java 2 Micro Edition) applications. The approach is exemplified through the Smart Ticket application by Sun.

KEY WORDS

Web applications, W2000, J2ME, Modeling

1 Introduction

Hand-held devices, like cellular phones and PDAs, are motivating a new family of Web applications. Nowadays, only a limited number of people uses their mobile phones to make reservations or buy goods through the Web, but in the (near) future these services will become much more common. If we leave apart sociology, problems come from both the communication infrastructure and availability of applications. Protocols, like GPRS and UMTS, are paving the ground to interesting solutions, but if we move to applications, it is not only a matter of using the right language to render a set of Web pages on a new device. The application must become a *mobile Web application*: It must be re-modeled to take into account the peculiarities of the new devices.

Small displays usually preclude the capability of visualizing composed and over-detailed pages. Mice and keyboards are minimal and do not support complex navigations and the insertion of long text strings. Also storage capabilities are limited: Programs must be optimized and down-

loaded data (e.g., pages) cannot be huge and contain fancy elements. Bandwidth should not be the problem if we consider speed. The new standards, like GPRS and UMTS, are fast enough, but users are charged per Kbytes and not per minutes anymore. The challenge now is not to increase the speed, but to minimize the amount of data exchanged.

These constraints lead us to imagine that mobile devices will mainly be used to access *service-based* applications, where the amount of information transmitted over the network is limited and the data that have to be visualized on the screen can suitably feed the small displays. This scenario assumes that the communication protocols are mainly GPRS and UMTS and that there is a “real” data exchange between the device and a server (or a set of servers). Other scenarios, like the use of mobile devices in local-area networks or to support hypertextual guides for museums or similar places, would impose different choices. In the former case, charging is not a problem, while in the latter case data are usually already on board and there is no real information exchange between the device (the client) and a server.

This paper does aim at being general, but limits its scope to the first scenario and does not even discuss the problems that arise when conceiving multi-device Web applications. It proposes a simple rule-based approach for fast prototyping the client side (oftentimes referred to as presentation tier [4]) of these applications. We do not supply a set of *special-purpose* pages that implies the adoption of a particular browser, but we release self-contained J2ME (Java 2 Micro Edition) applications. Almost all last-generation mobile devices are J2ME-enabled, thus the proposal is to implement the presentation tier as self-contained J2ME programs that run on the device and interact with the server only when needed. In this way, we have much more control on what is loaded on the device (not a set of pages, but an executable) and on server interactions. While designing/implementing the application we can *clearly* decide what should be client-side and what should relay on a server.

The paper starts with W2000 [3] as modeling means and presents μ W2000 as a lighter notation for explicitly specifying applications for handheld devices. Designed models can then be used as input to a *code generator* that

exploits some external rules to derive the J2ME code for the client side of the application. The approach is exemplified through the Smart Ticket application by Sun.

The rest of this paper is organized as follow. Section 2 briefly describe μ W2000, the base modeling language. Section 3 introduces the approach: a brief introduction to J2ME, our code generator and transformation rules, and some considerations on the transformation process. Section 4 explains how we have automatically generated the code for the Smart Ticket application and Section 5 concludes the paper.

2 μ W2000

μ W2000 is a light-weight version of W2000 [3], the modeling notation for Web applications developed at Politecnico di Milano. As the parent notation, it fosters *separation of concerns* by organizing a complete model into four different sub-models, as presented in Figure 1, which borrows UML packages to identify the different components¹. μ W2000 re-uses only the main constructs of W2000 and imposes additional constraints on the use of supplied elements. Lack of space forbids us to fully describe μ W2000; readers can refer to [5] for an in-depth presentation.

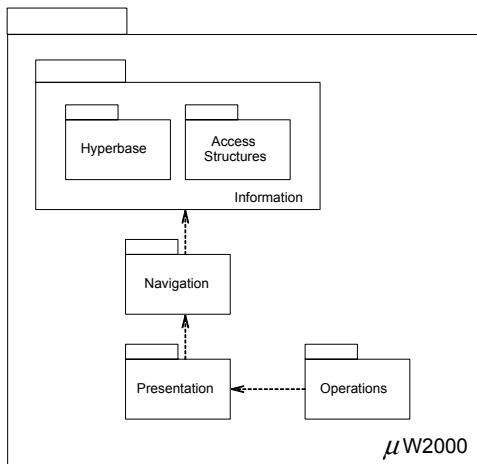


Figure 1. Overall organization of μ W2000 models

Conceptually, the starting point is the *information model* whose goal is the identification of all data that the application should deal with and organize them accordingly, where the former goal is up to the *hyperbase model* and the latter belongs to the *access structures model*. The *hyperbase model* comprises:

- *Entities*² that define conceptual “objects” that are of

¹Both W2000 and μ W2000 offer a UML-like syntax.

²Every μ W2000 element has several attributes that are not presented here. Interested readers can refer to [2] for a detailed description of W2000 as MOF metamodel.

interest for the user. They render the information contents that should belong to the application.

- *Components* that are used to structure the contents of *Entities* into meaningful chunks. They can further be decomposed in sub-components, but the actual contents can be associated with leaf nodes only.
- *Slots* that identify primitive information elements and are the typed attributes that specify the contents of leaf components. μ W2000 allows only “simple” types to be used for *slots*: This is to avoid the handling of complex data elements.
- *Semantic associations* that link pairs of *Entities* and identify navigational paths between two related concepts. μ W2000 requires that only the minimum set of *semantic associations* be identified. This requirements comes from the fact that usually mobile devices supply only essential navigation.
- *Association centers* that describe the set of “target” elements identified by a *semantic association*. If we had a 1 to n association, the center, which must be minimal in μ W2000, defines how to identify either the entire set of targets as a whole or each individual element in the set.

The *access structures model* is used to organize the information defined so far. It specifies the main access points to the application. It comprises only:

- *Collections* that define groups of elements that should be perceived as related by the user. *Collections* organize application data in such a way that the user can fully exploit them. Also *collections* can have *centers*. Minimalism is always an intrinsic requirements for μ W2000.

When we move to define how the user can browse through the application, we address the second layer, that is, the *navigation model*. It reshapes the elements in the previous model to specify the *actual* information chunks. The modeling elements here are:

- *Nodes* that define atomic units. Usually, they do not define new contents, but render information already defined in the *information model*. Since *nodes* are atomic elements, μ W2000 requires that they be “small”³.
- *Clusters* link together sets of *nodes* and define how the user can move around these elements. They can further be organized in: *Structural clusters* if all their elements come from the same *entity*; *association clusters* if they render associations; *collection clusters* if describe the topology of a collection, and *transactional clusters* if relate the set of nodes that the user should

³Needless to say, this is only a guideline. A rigorous constraint would be too heavy and would make the notation loose its flexibility.

traverse to complete a business transaction⁴. *Clusters* are also useful to identify how the current context varies while the user interact with the application.

In μ W2000, *nodes* roughly correspond to pages, thus the *navigation model* is of key importance. Too many *clusters* would mean a too complex application, but also too complex *transactional clusters* would lead to unusable applications. Thus the careful design of *clusters* – both in terms of their number and dimension – is a key activity for the successful deployment of the application.

The last step – that is part of μ W2000 mainly for consistency with the parent notation – defines the *presentation model*. It organizes the application in a set of pages together with links among them. This model can clearly be “confused” with the *navigation model* since the limited sources – in this case, the screen – do not offer the possibility of aggregating nodes in fancy pages. It is true, anyway, that the *navigation model* could supply small *nodes* that need to be grouped to become available to the user. To do this, the *presentation model* offers:

- *Publishing units* that are the smallest information elements that are visualized on pages. They usually render *nodes*, but can be used also to define forms, navigable elements, and labels.
- *Sections* that group related *publishing units* to better structure a page and improve the degree of reuse of page fragments. In μ W2000 *sections* and *pages* can be “confused”, but remain two different concepts.
- *Pages* that conceptually identify the screens as perceived by the user. Usually *pages* can be seen as sets of *sections*, but given the peculiarities of addressed devices, μ W2000 suggests the definition of *single-section* pages.
- *Links* that connect pages and identify the actual navigation capabilities offered to users. Statically, a *link* can be associated with multiple targets, the actual destination *page* is identified only at run-time by evaluating other information in the model (for example, the context). *Links* can also “hide” the enabling of computations (i.e., the *operations* defined in the *operations model*) on application data.

The *operations model* comprises all the *operations* that can be performed by the user on the application data. Each *operation* should clearly be specified in terms of pre- and post-conditions. Even if we did exercises on using a OCL-like language to specify *operations* ([1]), in this context, requirements are stated using natural language. We simply suggest rigor and completeness. For example, each *operation* should identify how it modifies data (not only in

⁴Since the term *transactional* could be misleading, it is important that we point out that such clusters identify only the nodes that belong to particular business transactions. The design and implementation of “real” physical transactions is a topic not addressed by this research.

terms of *entities*, but also considering back-end data), how it modifies navigation (e.g., new *nodes* and *links*) and how it modifies presentation (new *pages*). But it should also identify – if needed – the new page in which the user is moved after executing the operation.

Notice that *operations* are associated with *links* and thus *pages*. The business process that the application embodies can easily be obtained by considering the *transactional clusters*: Simple activity diagrams could be used to visualize these processes, where the activities would be the *operations* defined so far.

3 Our approach

The method described in this section is a traditional *rule-based* translation approach. In this case, we derive Java code from μ W2000 models. The following subsections briefly introduce J2ME, describe the high-level organization of the code generator (i.e., the rule interpreter) and transformation rules, and present some considerations on the transformation process.

3.1 J2ME

Our approach is based on the Java 2 Platform, Micro Edition (J2ME) [9]. Configurations supply the base functionality for classes of portable devices that share the same properties. The idea is to define an abstract machine on which developers can base their applications.

In our research, we exploit the CLDC (*Connection Limited Device Configuration*), which is the smaller configuration, specifically designed for devices like mobile phones: 16- or 32-bit CPUs, and a minimum of 128 to 512 Kbytes of memory available for the Java platform implementation and associated applications. On top of the CLDC, we have *profiles*, which supply a set of higher level APIs to control the *abstract* device. Applications are written for particular profiles, which are based on configurations. The first profile is the *Mobile Information Device Profile* (MIDP) that, designed for mobile phones and entry-level PDAs, provide the core functionality to control the user interface, network connectivity, local data storage, and application management. Combined with CLDC, MIDP provides a complete Java runtime environment that leverages the capabilities of handheld devices and minimizes both memory and power consumption. Applications developed on the CLDC-MIDP platform are commonly called *MIDlet*.

3.2 Code generation

The tool is a simple rule interpreter that, as exemplified in Figure 2, takes as input the set of rules and the μ W2000 model and produces as output a set of Java classes. Both rules and models are supplied as XML files: Rules, together with their format are described later in this section;

as to models we use XMI (XML Metadata Interchange [6]) as standard means to transform a model into an XML file⁵. The resulting XML file is slightly modified in such a way that: (1) each *entity* is immediately followed by its *components*, which precede their *slots* and (2) *nodes* follow the *transactional cluster* they belong to.

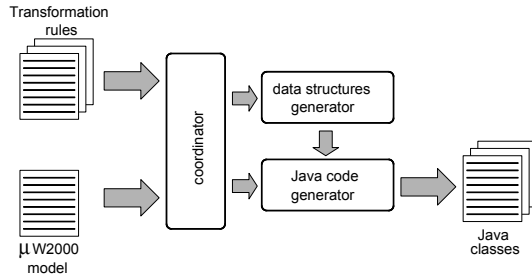


Figure 2. High-level architecture of the prototype interpreter

Rules are supplied as external inputs (files) to increase the degree of flexibility and be able to change them if needed. We must recall that the prototype has been used to conduct our experiments and – at least – in these first phases, extensibility and flexibility have been two key requirements for the supporting tool.

To produce the code (i.e., the set of Java classes), the user must select the rules that have to be applied. This is to consider also partial generation processes and do not force the user to produce useless code. The generation process is controlled by the *coordinator*; the application of each rule first produces the data structures (*data structures generator*) that are required by the rule and then, by exploiting these structures, the code itself (*Java code generator*).

```

<RULE>
<NAME> rule name </NAME>
<INPUT> μW2000 model element </INPUT>
<ACTIVEFRAGMENTS>
list of default active fragments
</ACTIVEFRAGMENTS>
<OPTIONS> options for code generation </OPTIONS>
<OUTPUT> code structure </OUTPUT>
</RULE>
  
```

Figure 3. Generic structure for transformation rules

The overall structure of transformation rules is presented in Figure 3. Further details and example rules can be found in [5]. The main elements of Figure 3 are:

- *Name* that defines the rule name.

⁵μW2000 models can be edited using a special-purpose add-in to Rational Rose [7], which fully supports W2000.

- *Input* that identifies the μW2000 element on which the rule is to be applied. For example, if we specified `page`, the rule should be applied on all pages of the model. It is the field `OPTIONS` that allows us to fine tune the rule application.

If we specified `DEFAULT`, the rule would be applied on the whole model, but just once. This is necessary to generate those parts of the code that do not depend on particular elements of the model, but are mandatory to run the application. For example, the code that is needed to start the MIDlet is produced by one of such rules.

- *Active fragments* that sets the fragments that should be taken into account while generating the code. To fully understand this section, we must consider that the directives contained in section `OUTPUT` are organized in fragments. Here we state those that must be active (taken into account) by default. In section `OPTIONS` we put further statements to control the application of optional fragments on the different instances of the notation element. The code generator crosschecks the two sections.
- *Options* that contains a set of further constraints that the generator must evaluate before applying the rule. For example, we can have `scan` and `condition` statements to selectively apply the rule only on those elements that satisfy the constraints or only if a general constraint is satisfied, respectively. For example, if we wanted to define a rule only for pages of a given type, we should use a `scan` statement to identify them in the set of all pages.
- *Output* that contains the directives to generate the Java code. In this section, we can refer to particular elements or properties of the model under analysis by using the usual dot notation. Again, if we were applying a rule for transforming pages, we could say that `page.sections[0].publishingUnit[0].-type` identifies the `type` of the first *publishing unit* of the first *section* of the current page (`page`).

3.3 Some comments

This section wants to discuss the limitations with the proposed approach. When we think of automatic code generation, we must always consider the starting point: The model can be as complete as the to-be-generated code, or it can be lighter and leave room to some human interventions. Clearly, since μW2000 is a technology-independent notation, we followed the second approach, where the model of considered application does not provide all information needed to generate the whole code. Currently, to bypass this problem, in many cases we provide default solutions or we clearly signal these situations with comments in the code that ask for the missing statements. A different solution would have been the addition of some new features to

the notation, but this would have implied heavier and less flexible models. More precisely, the problems that need further attention are:

- *Data retrieval:* Data can become available either because they are stored in a small local database or are supplied by a server. The code that uses these data is generated automatically, but currently we are able to supply only one default way to retrieve them from a server. More special-purpose solutions can easily be implemented by changing the few lines of code that manage data retrieval.
- *Server connections:* Connections are in many cases managed by methods supplied by the MIDP, but we need to decide when to open and close them. Again, this is nothing difficult and default solutions can always be foreseen. Needless to say, more case-specific solution would help better manager client-server communications.
- *Initialization:* To initialize such an application, we must identify the starting point, that is, the first page that must be displayed on the screen. Currently, μ W2000 does not supply this feature, but the simple hypothesis that the *main page* is the first page can help solve this problem. More sophisticated solutions must be conceived if we want to have different access points associated with different privileges on the application.
- *Resume:* The way data must remain consistent even in case of failures is a key topic for these applications. Currently, we simply assume that the server is in charge of dealing with these problem, but more sophisticated solutions (which would require by-hand programming) could relay also on client-side capabilities.
- *Sophisticated user-interfaces:* Currently we supply only general purpose and light solutions, but the implementer is always free to change and improve them the way he wants. In this context – but also for the aforementioned problems – we must have clearly in mind that the approach generates only first prototypes that usually do not supply complex, special-purpose, and fancy solutions.

4 Case study

After conceiving the approach and implementing the prototype, to assess the quality of the proposed method, we decided to try to generate the code of an existing application. Since the idea was to use a *widely-recognized benchmark*, we opted for the Smart Ticket application by Sun [8]. Roughly, as exemplified in Figure 4, users can buy movie tickets through their mobile phones. First-time users create an account and are then allowed to login to the application. Users can then select a movie, chose a theater and show time, select seats, and purchase tickets.

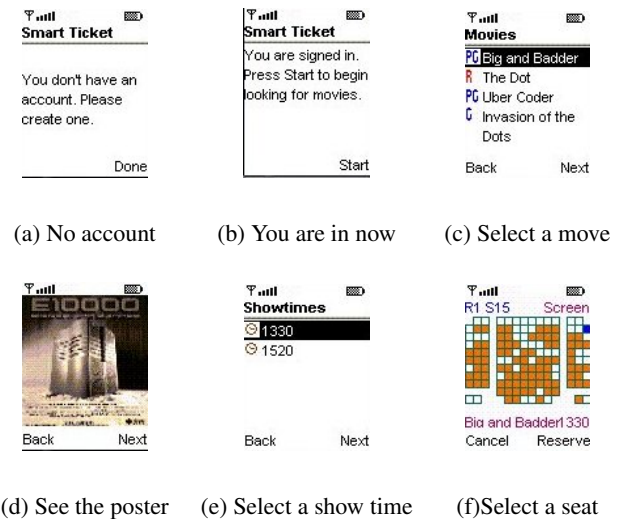


Figure 4. A few snapshots of the Smart Ticket application

The original application is organized around a well-known set of tiers. The *database layer* is responsible for the information with which the application deals with, the *EJB layer* handles the accesses to the database, the *servlet layer* handles HTTP requests from the MIDP client, and the *client layer* is the MIDlet application that sets up the screens and handles the command events from each screen. Up to the *EJB layer*, the application is just a standard J2EE back-end that could be interfaced with other well-known technologies – like JSPs – to deliver a conventional Web application.

We concentrated on the *client layer* and started modeling it with μ W2000 both to assess the capabilities of the notation and to get the complete picture. Readers can refer to [5] for the complete μ W2000 model; Figure 5 shows only a few excerpts.

Figure 5(a) presents the entity *Movie* (*hyperbase model*). The left-hand part defines the entity with a single component, while the right-hand part specifies the slots that belong to the component: A movie is characterized by an *id*, a *title*, a *rating*, and a *posterURL*. Figure 5(b) describes the transactional cluster that we foresee for buying a ticket (*navigation model*). Rectangles represent nodes (pages): We start from selecting the movie, then we see the poster, select a theater, decide the shorttime, and choose the seat. After all these steps, we are ready to confirm the reservation. Notice that the example is intentionally simple, but clusters can contain also branches and loops to fully model the order with which nodes can be traversed to complete a business process. Finally, Figure 5(c) describes the page to select a movie (*presentation model*). The page comprises a single section, which in turn is made of three presentation units. The first unit embodies a link to move back to the *main page*, the second unit displays a list of selectable movies, and renders a node which in turn materializes the

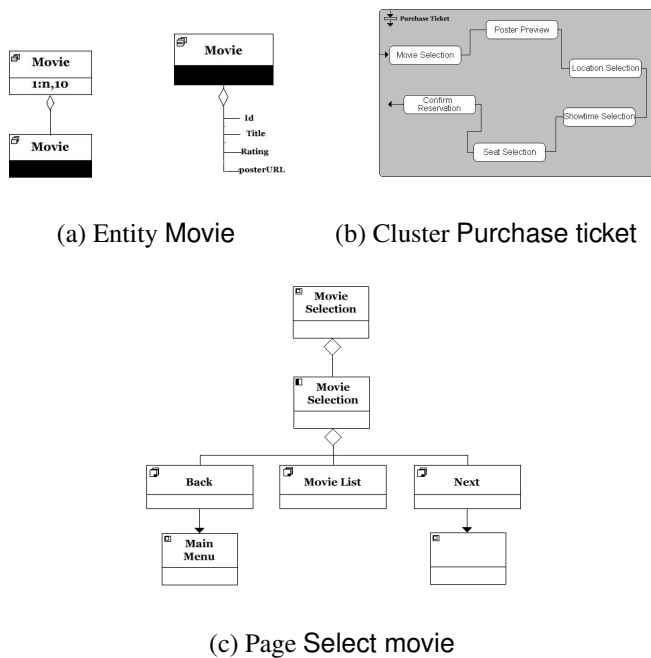


Figure 5. Excerpts from the μ W2000 model of the Smart Ticket application

collection of all available movies, and the third unit lets the user move to the next page (since it is not important in this context, the target is left unspecified).

4.1 Obtained results

Given the model sketched so far, we applied our prototype code generator and analyzed the code obtained. First of all, the idea was not to mimic the example code, but was to assess the correctness, degree of completeness (with respect to the constraints already identified in Section 3.3), and actual improvement in the development process.

The first analyses highlighted that the improvement in terms of saved effort can be quantified in 70-80 %. This was measured by taking into account the time needed to develop such an application: If 100 is the total, 70-80 comes from the code generator, the implementer is responsible for the remaining 20 only. This is also true if we consider the lines of codes. In this case, the measure must take into account that automatically generated code is usually not as optimized as hand-written one.

It must be clear that this is only a first prototype: It can run with a reasonable limited effort, but it is not optimized and does not supply fancy graphical interfaces (like those that could be programmed using the API).

5 Conclusions and future work

The paper presents μ W2000 and an approach for fast-prototyping the client side of mobile Web applications with

J2ME. The results gained so far – by exploiting a prototype implementation of the *code generator* – are encouraging and motivate our future work in different directions. First of all μ W2000 must be refined to better address the class of considered devices. Also its relationship with J2ME will be further analyzed. Some specific features are not exploited because of the neutrality of the notation; a few highly-specialized new modeling constructs could help better exploit J2ME.

We need also to further investigate the client-server communication. Currently we assume only server-side computations and also the information exchange is foreseen in a single standard way. In the future, we would like to be able to relax these constraints to support client-side computations and also more flexible communications (for example, through Web services).

Acknowledgments

The authors want to thank Nicola Laratro and Michele Panigada for their invaluable work. Their ideas, comments, and work on implementing the first prototype have helped shape this work.

References

- [1] L. Baresi, G. Denaro, L. Mainetti, and P. Paolini. Assertions to Better Specify the Amazon Bug. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 585–592. ACM Press, 2002.
- [2] L. Baresi, F. Garzotto, and M. Maritati. W2000 as a MOF Metamodel. In *Proceedings of the 2002 World Multiconference on Systemics, Cybernetics and Informatics*, volume I, 2002.
- [3] L. Baresi, F. Garzotto, and P. Paolini. From Web Sites to Web Applications: New Issues for Conceptual Modeling. In *Proceedings of the International Workshop on The World Wide Web and Conceptual Modeling, co-located with the 19th International Conference on Conceptual Modeling*, 2000.
- [4] J. Conallen. *Building Web Applications with UML*. Addison-Wesley, 2nd edition, 2002.
- [5] N. Laratro and M. Panigada. An Approach for Fast Prototyping with W2000 and J2ME. Master’s thesis, Politecnico di Milano, 2002. In Italian.
- [6] Object Management Group. XML Metadata Interchange (XMI) Specification (version 1.2). Technical report, OMG, 2002.
- [7] IBM Rational Software Corporation. *IBM Rational Rose: User’s Manuals*, 2004.
- [8] Sun Microsystems. Java™ 2 Platform, Micro Edition (J2ME), 2002. <http://java.sun.com/j2me/>.
- [9] Sun Microsystems. Java Smart Ticket Sample Application 1.2, 2002. <http://developer.java.sun.com/developer/releases/smartticket/>.