

Reengineering a PC-based System into the Mobile Device Product Line

Weishan Zhang and Stan Jarzabek
Department of Computer Science, SOC
National University of Singapore
Lower Kent Ridge Road, Singapore 117543
{zhangws, stan}@comp.nus.edu.sg

Neil Loughran and Awais Rashid
Computing Department
Lancaster University
Lancaster, LA1 4YR, UK
{loughran, awais}@comp.lancs.ac.uk

Abstract

There is a growing demand to port existing PC-based software systems to mobile device platforms. Systems running on mobile devices share basic characteristics with their PC-based counterparts, but differ from them in details of user interfaces, application models, etc. Systems running on mobile devices must also perform well using less memory than PC-based systems. Mobile devices themselves are different from each other in many ways, too. In this paper, we describe how we made an existing PC-based City Guide System available on a wide range of mobile devices, in a cost-effective way. We applied “reengineering into a product line architecture” approach to achieve the goal. Our product line architecture facilitates reuse via generation – we generate specific City Guide Systems for target platforms including PC, Pocket PC and other mobile devices, from generic meta-components that form the City Guide System product line architecture. In our project, we used a meta-programming technique of XVCL to build a product line architecture for City Guide Systems.

1. Introduction

Mobile devices (such as Pocket PC, mobile phones, etc.) are ubiquitous now. With advances in hardware, mobile devices today are capable of running software that yesterday could only run on a PC. Naturally, there is a growing demand to port many of the existing PC-based software systems to smaller mobile device platforms. While systems running on mobile devices share basic characteristics with their PC-based counterparts, they also differ from them in many details, for example user interfaces. Systems running on mobile devices must perform well using less memory than PC-based systems. There are also many variations across different mobile devices that affect system characteristics, such as 320x240 colorful screen versus 100x80 mono display, lower end mobile phone whose memory is less than 100kb versus Pocket PC which has up to 64M memory or higher, etc. Some of the high-end mobile devices can run

J2SE¹ application; while most of the mobile devices are only J2ME² enabled because of the limited resources.

We applied “reengineering into the product line architecture” approach to make an existing PC-based City Guide System available on a range of mobile devices, in a cost-effective way. Firstly, we analyzed the PC-based City Guide System (GS-PC for short, based on the Lancaster GUIDE project³) to identify candidates for reusable components. Then, we reengineered existing code into generic, adaptable meta-components that formed a GS product line architecture. The members of the GS product line are City Guide Systems for PCs (including desktop/laptop/tablet PCs) and for mobile devices (including Pocket PCs, mobile phone, etc.). We can build specific City Guide Systems by reusing (composing after possible adaptations) meta-components of the GS product line architecture. In this paper, we describe both the reengineering process and the structure of the product line solution. We used a meta-programming technique of XVCL to build a GS product line architecture in our project.

XVCL [1] [2] is a meta-language, method and tool we have developed for building flexible, adaptable and reusable software. It is also a powerful variability management mechanism. We have successfully applied XVCL in two medium-scaled product line projects and a number of smaller case studies (e.g., [3] and [4]). A more detailed description of XVCL can be found in [1] [2].

The paper is organized as follows: In the next section, we talk about the related work on reengineering; then section 3 introduces GS-PC briefly; section 4 provides an overview of “reengineering into the product line architecture” approach; then we analyze this City Guide System in section 5. Followed are the sections for City Guide domain analysis and positioning the GS-PC for the reengineering. The design and implementation of the City Guide System product line architecture are described in section 8; then we show how to get a specific City Guide

¹ Java™ 2 Platform Standard Edition, <http://java.sun.com/j2se>

² Java™ 2 Platform Micro Edition, <http://java.sun.com/j2me>

³ Davies, N et al. Lancaster GUIDE project homepage
<http://www.guide.lancs.ac.uk/>

System by customizing this architecture with two examples in section 9. Section 10 presents the discussion of the experiment results, such as the advantages and disadvantages. Finally, some concluding remarks are given in section 11.

2. Related work

Reengineering an old system often assures better return of investment than development from scratch. Existing systems are reengineered to improve maintainability, to port a system into a new platform, to position a system for major enhancements or to increase reusability.

Reengineering involves reverse engineering and forward engineering [5]. Success stories with reverse engineering have been reported (e.g., [6]), many tools have been implemented both in industry (e.g., [7]) and academia (e.g., [8] [9] [10]), but more experimentation is needed to fully explore the potential and limits of automated reverse engineering. Canfora, Cimitile and de Carlini [9] identify a number of problems that hinder wide adoption of reverse engineering tools. In particular, the authors note that current reverse engineering tools:

- fail to identify the right level of details in recovered design and lack proper design presentation strategies,
- recover only small portion of the design information software maintainers need, and
- are not flexible enough, cannot be easily tailored and enriched in the operating environment.

The authors further point out that although reverse engineering tools can recover design views that are commonly used during forward engineering, such as structure charts. However, the design information that is essential during software maintenance (and therefore, should be produced by reverse engineering tools) is different from the design information used during forward engineering.

The reuse oriented reengineering [11] is not new. Thomson et al [12] use domain knowledge to maximizing reuse during the mining of reusable components at the early stage of reengineering, but there is no mention of means for facilitating reuse in forward engineering. Component adaptability is not addressed but listed as the future work. Canfora [13] present an approach based on a COBOL project. Although the proposed reengineering process can be applied to other programming languages, it is hard to adapt the mined modules to accommodate unexpected changes; and also the software architectural level issues are not addressed.

Reengineering using design patterns [14] [15] involves design pattern recognition and re-implementation using identified patterns. The use of design pattern is not a “free lunch” [16]. While design patterns may facilitate the

reengineering, there are two critical shortcomings: lack of extensibility and performance degradation because they incur additional runtime overhead [17]. The performance is crucial to the mobile devices where available resources are very limited. From our experience, design patterns are ill-suited to deal with the multiple platform reengineering situations as they lack the desired flexibility.

Research on the architecture reengineering and architecture reconstruction [18] [19] [20] are addressing the reengineering issues at architectural level and this is the right direction in order to improve the reusability in reengineering. Use case based [18] or scenario based [19] architecture reconstruction seems practical. Combining both static and dynamic views for architecture reconstruction [20] is useful for the architecture description of the system. These approaches are on the right track for reengineering to one system, but they are inflexible in handling the large number of variants during the reengineering to a product line where adaptability and variability play much more important role than that for a single system.

A *software product line* is a set of software-intensive systems sharing a common, managed set of features developed from a common set of core assets in a prescribed way [21]. Product line members keep their own personalities through the satisfying of variant requirements. Naturally the City Guide System running on a range of mobile devices forms a software product line as the City Guide System should be adapted accordingly in order to accommodate different devices. This mobile Guide system product line can cover Guide system running on both J2SE and J2ME platforms.

There is a distinct scarcity of published work on how to reengineering an existing system to a product line. Stoermer [22] briefly mentioned this problem but there are no serious technical issues presented. Software product line is an effective approach to achieve reusability and productivity. The reengineering of the PC based system to a product line is attractive to us because of the promised benefits of a product line. This motivates us to investigate the problem of how to make existing PC based Guide system function on a wide range of mobile devices in a cost-effective way. A cost-effective solution should have the following important characteristics:

- we should be able to obtain custom systems running on various mobile devices from the customization of a common base of generic, adaptable and reusable meta-components, rather than implementing each system separately in an ad hoc way, and
- we should be able to reuse as much as possible of existing software system to build that common base of generic, adaptable and reusable meta-components

We will elaborate on this solution in the following sections.

3. A City Guide System

The PC-based City Guide System (Figure 1), serves as a city tour guide, helping users find information about the places of their interest. It works like a customized web browser, guiding and navigating users through the information space using backward, forward, refresh, and other similar functions. The system provides history storage and bookmark services, and uses a cache for efficiency. Other useful functions include context-sensitive services such as the display of the map detailing the area where a user stands. Our GS-PC was developed with JDK1.4 (J2SE).



Figure 1. A snapshot of City Guide System (GS-PC)

4. An overview of the reengineering process

The stages involved in the process of reengineering the GS-PC into the City Guide product line architecture are depicted in Figure 2.

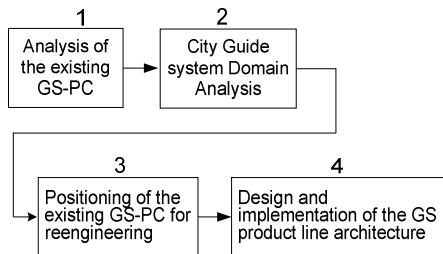


Figure 2. The main stages in “reengineering into the product line architecture”

For start, we analyzed the GS-PC code and its documentation. In addition to functional and non-functional requirements, we analyzed system’s runtime

component architecture and mappings of requirements to components. With our understanding of the goal, we also identified those components that we thought might be useful in building the GS product line architecture.

Secondly, we analyzed the City Guide domain, taking into account both PC-based and mobile device-based systems. The objectives here were to identify requirements common to all (or most) of the systems in our future GS product line, as well as variant requirements. Both common and variant requirements scope the product line, defining the range of systems we wish to address in the product line. Our domain analysis was guided by business needs – we wished to provide City Guide Systems running on a variety of devices, customizable to the needs of many customers.

In the third stage, we positioned components of the existing GS-PC for reuse in the product line architecture. When creating first-cut meta-components in this stage, we addressed mainly variants across PC-based City Guide Systems. In many cases, this also helped us address variants related to mobile devices. We could also discard code that was not relevant in the product line situation.

The actual reengineering was completed in the fourth stage. We refined meta-components so far and added more meta-components. The refinements included extensions of meta-components and also building a proper variability handling mechanism into them. At the end of this stage, we obtained the product line architecture and meta-components for both PC-based and mobile device-based City Guide Systems.

5. Analysis of the existing GS-PC

Having understood functional requirements for the GS-PC, we analyzed its runtime software architecture, both structure and components [23]. We use the software architecture concept from Shaw and Garlan [24] and describe it as a set of interacting components. The runtime architecture of the GS-PC is depicted in Figure 3 as an UML component diagram.

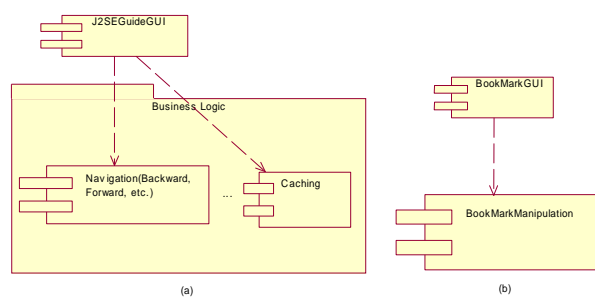


Figure 3. Runtime architecture of the GS-PC

The GS-PC runtime architecture consists of two component layers, namely presentation layer (user

interface) and business logic layer. The 2-layered structure of runtime architecture remains the same for all the members of the intended product line. However, PC-based City Guide Systems will have graphical user interface (GUI) while most of the mobile device-based systems will use a simple textual user interface (TUI).

After identifying components of the GS-PC, we documented mappings among functional requirements and components. During analysis, we evaluated components' potential for the future product line [25], taking into account:

- components' portability across the J2SE and J2ME platforms
- the amount of re-work required to adapt a component for the product line architecture

Some of the candidates for meta-components in the future product line architecture could be identified already at this early stage. For example, the navigation service must be provided by all the City Guide Systems, members of the product line. Therefore, we should consider reengineering of the existing navigation components into the meta-components for reuse across the product line members. Also, user interface elements (such as windows, buttons, etc.) can be considered for product line meta-components that will facilitate the building of user interfaces for product line members. Certain user interface elements are usually used together with other elements. For example, a label is used together with text field, list, etc and they are considered together as composite elements. We will develop meta-components for such composite elements to foster productivity even further.

6. Domain analysis of the City Guide domain

The City Guide System, running on either the J2SE or J2ME platform, should provide us with the basic navigation service, and this is the common part to all the members of the GS product line (GS-PL for short). City Guide Systems use an event-driven mechanism for component invocation: when an event occurs, a dispatcher implicitly invokes components that registered an interest in the event via their respective event handlers.

6.1. Variants in GS-PL

The following are some of the variants in the City Guide domain:

- Cached or not cached. A system may have cache facilities or not.
- Cache size. Different system may need different cache size.
- Caching scheme. For instance, most often accessed or most recent information can be prioritized. Caching strategy will also affect the record deletion mechanism of the cache.

- Record deletion amount. Percentage of (cleared cache)/(total cache), for example, 10%, 20% or 50%.
- User interface variants, such as look and feel, icons, layout, etc.
- A system may or may not have a bookmark mechanism.
- Application Models. The application model defines how an application is managed and how management responsibilities are divided between the application and the underlying operating system (or other system-level software) [26]. J2ME currently supports four different application models.
 - The Traditional Application Model. This is the simplest application model [James] which is also used in J2SE. The entry point to the application is a static method called main().
 - The Applet Model. The main class in an applet extends `java.applet.Applet` and main class defines four life-cycle notification methods: `init()`, `start()`, `stop()`, and `destroy()`.
 - The MIDlet Model. A MIDlet's main class extends `javax.microedition.midlet.MIDlet`. The main class defines three life-cycle notification methods: `startApp()`, `pauseApp()`, and `destroyApp()`.
 - The Xlet Model. An Xlet's main class implements the `javax.microedition.xlet.Xlet` interface, which declares four life-cycle notification methods: `initXlet()`, `startXlet()`, `pauseXlet()`, and `destroyXlet()`.

Differences in J2SE and J2ME platforms affect implementation of some of the components (for example, GUI for J2SE and TUI for J2ME platform require different implementations). Because of similarities between those two platforms, implementation of yet other components remains the same on both platforms.

6.2. Feature model for the City Guide domain

It is important to clearly understand the range of common and variant requirements that we plan to implement, or have already implemented, in the product line architecture. Furthermore, variants are often mutually dependent. Inter-dependencies among variants usually manifest in one of the following ways:

- once we have selected one variant we should also select other variant(s) ;
- we cannot select a certain variant unless we have already selected other variant(s);
- selection of one variant makes it impossible for us to select another variant(s).

Inter-dependencies among variants are important as they allow us to decide which configurations of variants are legal and which are not (a legal configuration of variants can be accommodated in a single product line member).

There are a number of possible types of requirements that we should model during domain engineering. A requirement may be a functional or quality requirement (e.g., BookMark an item of interest, system response time), may refer to design decisions (selection of a Caching strategy or architectural style) and to the implementation (the choice of cache deleting algorithm, option for platform). Clearly, in domain engineering, depending on the objectives determined during product line scoping, we may need to deal with any or all of the above variant requirements.

Some of the requirements can be enumerated. We classify requirements as follows:

- **Mandatory Requirements.** Mandatory requirements appear in all the instances of a concept being

described. The “concept” may refer to anything that matters in a domain, in particular, a real-world entity from the problem space, specific requirement, architecture element, design decision, code component, etc.

- **Variant Requirements.** Variant requirements (or variants for short) appear in some of the instances of a concept being described. Variants are further qualified as optional, alternative and or-requirements. An alternative describes one-of-many requirements. An or-requirement describes any-of-many requirements.

Feature diagrams (Figure 4) are used to graphically model common and variant requirements in a domain, and some of the variant inter-dependencies [27] [28].

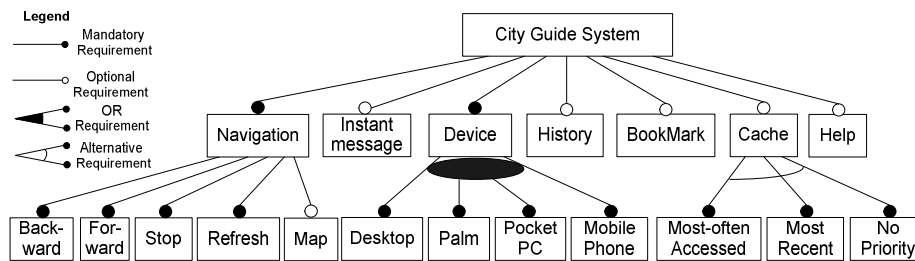


Figure 4. A feature diagram for the City Guide System domain

7. Positioning of the existing GS-PC for reengineering

We started by evaluating the impact of variants on system components. Some variants have localized impact on one component only: for example, the button size only affects the buttons. Other variants may have a wider impact on many components. For example, the bookmark facility affects the user interface (e.g., we need a button or menu item to open a bookmark management window and the new window itself) and requires business logic to implement functions for bookmark manipulation. The ways to deal with impact of a variant may range from setting a component parameter or making a small change to a component’s code, to adding new components, modifying component interfaces, etc.

To facilitate the management of variants and to improve the reusability, we introduce a meta-representation for components. Meta-components are component building blocks, designed for ease of adaptation and reuse. We generate concrete components from meta-components.

Meta-components in a product line should be generic and adaptable to accommodate variants, without compromising quality attributes such as maintainability or reliability. Quality attributes are important in a product line because they affect the component’s total cost of ownership across a family of products [25]. We use XVCL in our project to implement meta-components. The

technique of XVCL is capable of both: handling variants of small and large impact on system components, and preserving quality attributes during customization of the product line architecture.

Meta-components contain Java code inter-mixed with XVCL commands. XVCL commands indicate how a meta-component can adapt meta-components below, and how a meta-component can be adapted by meta-components above. The flow of adaptations is indicated by *<adapt>* arrows. (For readability, we enclose XVCL commands inside angle brackets.)

The *<adapt>* relationship among meta-components defines a hierarchical structure that we call an x-framework. An x-framework is an implementation of the product line architecture concept in XVCL.

Figure 5 depicts meta-components that we identified based on analysis of the GS-PC. Initially, we considered only the J2SE user interface variants. Then, we incorporated other variants into the GS product line architecture incrementally until all variants have been addressed.

The top-most meta-component, called *SPC* (Figure 5) specifies how to construct all the components for a specific City Guide System.

A *template* meta-component (*template* for short) is a special type of meta-component that facilitates generation of a group of related components. It defines a composition of the underneath meta-components. For example, GuideMain and BookMark in Figure 5 are

templates for GS-PC. Templates are designed according to the following pattern: for each screen, there is a

template that plays a role of a container for the adapted meta-components.

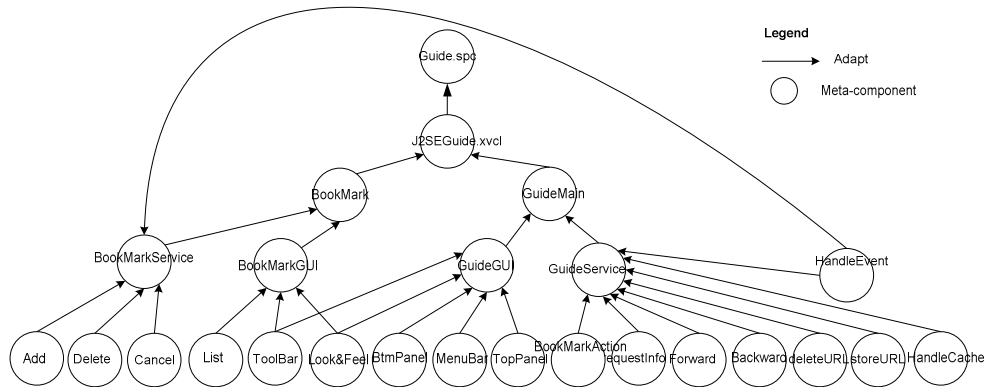


Figure 5. An x-framework for the GS-PC

The XVCL processor traverses the meta-component hierarchy (x-framework), starting with SPC. The traversal order is dictated by `<adapt>` commands embedded in visited meta-components. For each visited meta-component, the processor interprets XVCL commands embedded in that meta-component and generates source code for the customized components of the City Guide System (Figure 6).

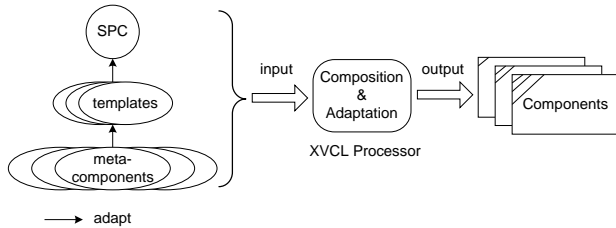


Figure 6. Component construction with XVCL

The SPC (Figure 7) controls the overall process of components generation.

<i>name</i> : SPC	
set	<i>BookMark</i> = Yes
set	<i>CacheSize</i> = 20
set-multi	<i>ToolButtons</i> = <Back,Forward,Stop,Refresh,Map,BookMark,Help>
set-multi	<i>Events</i> = <Back,Forward,Stop,Refresh,Map,BookMark,Help>
set	<i>ToolBarWidth</i> = 50
set	<i>ToolBarHeight</i> = 50
.....	
ifdef	<i>CacheSize</i>
	adapt <i>Caching outfile</i> ="Caching.java"
Select	<i>option</i> = <i>BookMark</i>
	Yes Adapt <i>BookMark.xvcl outfile</i> ="BookMark.java"
	No

Figure 7. SPC for J2SE Guide system

Meta-variable *BookMark* is a flag that indicates whether the bookmark facility should be included into the Guide system or not. This flag controls the `<select><option>` command. Only when *BookMark* is `<set>` to *Yes*, the bookmark meta-components will be `<adapt>`ed and the related bookmark code components will be generated. A meta-variable *CacheSize* is another flag indicating whether a system should have a Cache function or not. If *CacheSize* is defined then the Cache facility will be `<adapt>`ed into the system (with command `<ifdef>`). In this way `<select></option>` and `<ifdef></ifndef>` commands can accommodate anticipated variants. Multi-value meta-variable *Toolbuttons* is defined to manipulate the creation of buttons. In our case, *Back,Forward,Stop,Refresh,Map,Help* are all of the button items used in this system. The button size is represented by two XVCL meta-variables, too. Meta-variable values are propagated to the descendent meta-components according to the meta-variable scoping rules of XVCL.

This kind of parameterization via meta-variables and meta-expressions plays an important role in building generic, reusable programs. It provides means for creating generic names and controlling the traversal and adaptation of an x-framework. The following is the toolbar meta-component used to create the button items.

name :	<i>Toolbar.xvcl</i>
while	<i>Using-items-in="@ToolButtons"</i>
	JButton <i>j@ToolButtons</i> =new JButton(" <i>@ToolButtons</i> ");
break	<i>Icon</i>
	<i>j@ToolButtons</i> .setIcon(new ImageIcon("ICONS/"+" <i>@ToolButtons</i> +".gif"));
	<i>j@ToolButtons</i> .setPreferredSize(new Dimension(<i>@ToolbarWidth</i> , <i>@ToolbarHeight</i>));

Figure 8. Toolbar meta-component

All the meta-variables used here are defined in SPC (Figure 7) and their values are propagated down to this toolbar meta-component. A reference to meta-variable, such as *@ToolbarWidth*, is replaced by the meta-variable's value during processing. The value of meta-variable *ToolbarWidth* may be *<set>* to 50, 40, etc., as required at the adaptation point. Multi-value meta-variable *Toolbuttons* is used to create a generic button name *j@ToolButtons*. *<while>* command iterates over *Toolbuttons* and generates code for a toolbar in a loop, including button icon, button size. In each loop, value of *Toolbuttons* will be evaluated when it is referred and accept one value from the meta-variable's value list in sequential order, and then this value is concatenated with a button name prefix *j* to generate button name. In this way, a list of buttons' names *jBackward*, *jForward*, etc are created according to the list of values in SPC.

Inserting code and specifications at designated break points is a simple yet powerful means to handle unexpected variants. For instance, the toolbar may be required to have tool tips, various button font style, etc and these requirements are not predictable in advance. We solve this kind of problem with *<insert>/before/after* and *<break>* commands in XVCL. Inside the *<while>* loop in Figure 8, there is a *<break>* point which can be used for injecting new code to change the default implementation. For example, we can insert tool tips to overwrite the default or keep this default icon feature while inserting new feature by *<insert-before>/<insert-after>* new code. When icons are not necessary, we can insert empty content when *<adapt>*ing this toolbar meta-component; the content within the break name "icon" will become empty. In general, a *<break>* point can have a default content (Java code with XVCL commands) and higher-level meta-components can replace the default content or insert extra content after/before the *<break>* points in *<adapt>*ed meta-components. If no *<insert>* command from higher-level meta-components affects the *<break>* point, then the XVCL processor emits the *<break>*'s default content to the output.

In summary, meta-components are more generic, flexible and reusable than code components. Adaptations of meta-components are achieved by means of

parameterization via meta-variables and meta-expressions, insertions of code and specifications at designated break points, selection among given options based on conditions, code generation by iterating over certain sections of meta-components, etc. The meta-components are organized into a hierarchy called an x-framework that forms a product line architecture.

8. Design and implementation of the GS product line architecture

A product line architecture should be reusable, extendable, and configurable to accommodate variant requirements to product line members. From a product line architecture, we build product line members that satisfy specific requirements. A GS product line architecture must accommodate both the J2SE and J2ME variants we have identified in the domain analysis phase.

8.1. J2ME platform versus J2SE

J2ME contains only a subset of the features available in J2SE. This means that some of the code in GS-PC, in particular GUI code, may not run on J2ME, and some may be difficult to interface with (e.g., code related to event listening).

For small devices such as mobile phones, we use textual user interface rather than the GUI that is commonly used on the J2SE platform. Other differences are related to the application model: we use The MIDlet Application Model for J2ME City Guide Systems which extend the abstract MIDlet class as explained before. On the other hand, for J2SE City Guide Systems we use The Traditional Application Model.

J2SE and J2ME platforms use slightly different forms of the event listening mechanism. On the J2SE platform, you register listeners with application's components and then the event handlers are invoked when the relevant event occurs (Figure 9).

```
public void actionPerformed(ActionEvent e)
{ String event=e.getActionCommand();
.....
}
```

Figure 9. Event listening in J2SE

But in the MIDlet application model of the J2ME platform, we can only create events based on the label of the component (Figure 10).

```
public void commandAction(Command c, Displayable d)
{ String event=c.getLabel();
.....
}
```

Figure 10. Event listening in J2ME MIDlet application

However, apart from the differences in interfaces as shown above, the majority of the code for the business logic part remains exactly the same on both J2ME and J2SE platforms and can be reused across the systems running on those platforms. Based on the above analysis we conclude that it is feasible to support J2ME and J2SE versions of the City Guide System by a single product line architecture.

8.2. Extending the x-framework to J2ME platform

We extended the x-framework described in section 7 to address differences between J2ME and J2SE platforms. In the first step, we created an x-framework for J2ME systems. The new template meta-component, J2MEGuide.xvcl is responsible for generation of the MIDlet application. As the city Guide system on J2ME is also using the event-driven mechanism, we can reuse the event handling meta-component HandleEvent.xvcl. Other guide services, such as the deleteURL, storeURL, backward and forward action, remain the same for J2ME. Therefore, we can design the following x-framework for the J2ME Guide system:

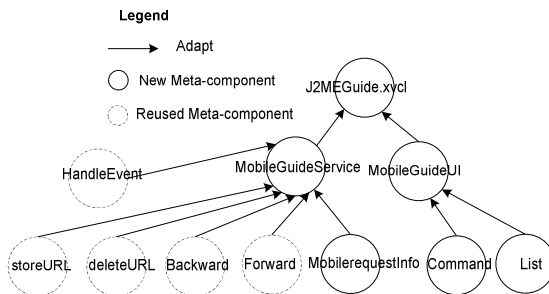


Figure 11. X-framework for the J2ME GS

8.3. Implementation of the J2ME related meta-components

For different event listening mechanisms, we separately tag the J2SE and J2ME specific code with `<select>` and `<option>` commands. Figure 12 depicts the event handler meta-component for handling events throughout J2SE and J2ME platforms.

<i>name</i> : <i>HandleEvent.xvcl</i>	
select	<i>option</i> ="Platform"
J2SE	public void actionPerformed(ActionEvent e) { event=e.getActionCommand();
J2ME	public void commandAction(Command c, Displayable d) { String event=c.getLabel();
while	<i>Using-items-in</i> ="Events"
	if(event.equals("@Events")) do@Events ();
text	}

Figure 12. Event handling meta-component across both platforms

Based on the options defined in meta-variable *Platform*, the corresponding event listening mechanism will be chosen during processing. Multi-value meta-variable *Events* controls the generation of the code for the dispatching of events.

The template meta-component *MobileGuideService* is a container which composes all the mobile Guide services together.

<i>Name</i> : <i>MobileGuideService.xvcl</i>
adapt <i>Eventhandler.xvcl</i>
adapt <i>MobileRequestInfo.xvcl</i>
adapt <i>StoreURL.xvcl</i>
adapt <i>DeleteURL.xvcl</i>
adapt <i>BackActionL.xvcl</i>
adapt <i>ForwardAction.xvcl</i>

Figure 13. MobileGuideService meta-component

8.4. Realizing a GS product line architecture as an x-framework

The meta-components we developed in the above steps have incorporated both the domain defaults and variants across J2ME and J2SE platforms. Now we can merge all the related meta-components together to form an x-framework for City Guide product line (Figure 14). A generic SPC can be designed to cater for needs of both platforms. We discuss two examples of SPCs in the next section. We can build a Guide system meeting specific requirements by customizing the above x-framework, whether it is for the J2SE platform or for J2ME platform.

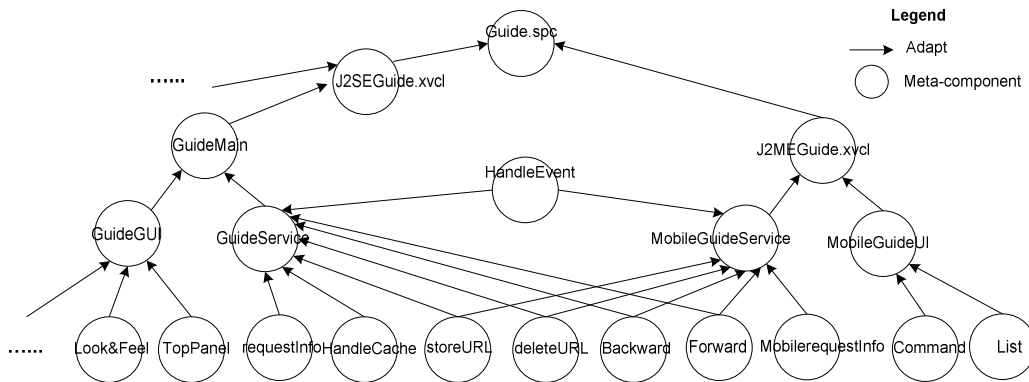


Figure 14. X-framework for the whole GS-PL

9. Customizing the x-framework

The application engineering is a process of customization of the product line architecture to build new products through the reuses of existing components. We demonstrate this with two mobile City Guide Systems running on Pocket PC and a high end mobile phone. These two applications are created by customizing the general x-framework (Figure 14) we have described in previous sections.

To develop a specific Guide system, we first examine the feature diagram (Figure 4) to select the required variants. Then, we configure variants in the SPC to meet specific requirements for the target system.

Suppose we need a City Guide System running on a Sony Ericsson P800 using J2ME platform, a high end mobile phone with 12M internal memory. Our City Guide System will have a caching facility but no BookMark facility. It will use the textual user interface (TUI). An SPC for such City Guide System is shown in Figure 15.

name : SPC	
set	<i>BookMark</i> = No
set	<i>CacheSize</i> = 10
set	<i>DeletePercentage</i> = 30
set	<i>Platform</i> = J2ME
set-multi	<i>Events</i> = <Back,Forward>
.....	
select	<i>option</i> = Platform
	J2ME adapt J2MEGuide.xvcl
	J2SE adapt J2SEGuide.xvcl

Figure 15. SPC for City Guide System on Sony Ericsson P800

Meta-variable *Platform* controls generation of components for different platforms. Figure 16 shows the runtime architecture of the City Guide System.

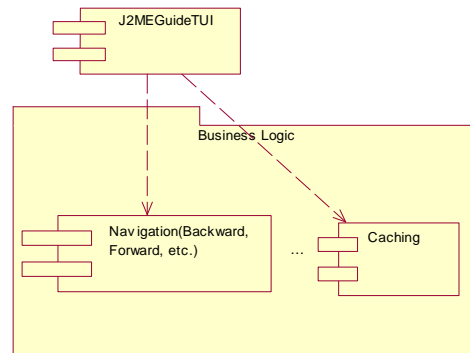


Figure 16. Runtime architecture of City Guide System on P800

A snapshot of this mobile City Guide System is shown in Figure 17.



Figure 17. Running result of J2ME City Guide System on P800

Another example is a Guide system on Pocket PC that has bookmark, less cache compared to GS-PC and also it does not have toolbar icons. As Pocket PC is powerful enough to run J2SE, therefore this Guide system is a J2SE application.

<i>name</i> : SPC	
set	<i>BookMark</i> = Yes
set	<i>CacheSize</i> = 20
set	<i>DeletePercentage</i> = 20
set	<i>Platform</i> = J2SE
set-multi	<i>ToolButtons</i> = <Back,Forward,Stop,Refresh,Map,Help>
set-multi	<i>Events</i> = <Back,Forward,Stop,Refresh,Map,Help>
set	<i>ToolbarWidth</i> = 30
set	<i>ToolbarHeight</i> = 30
.....	
select	<i>option</i> = Platform
	J2ME adapt J2MEGuide.xvcl
	J2SE adapt J2SEGuide.xvcl
	insert Icon

Figure 18. SPC for City Guide System on Pocket PC

The runtime architecture for this Guide is exactly the same as the one in Figure 3. As we explained in section 7, we can overwrite the default content inside a <break> by <insert>ing some other contents. This is accomplished in the above SPC with <insert break="icon"/> in order to delete the default toolbar icons. Although the named <break> point *Icon* does not really exist in meta-component *J2SEGuide*, but this <insert> will be propagated down to the descendent meta-components where *Icon* is defined and in our case, the toolbar meta-component in Figure 8.

10. Discussion

We illustrated reengineering of a PC-based system into the mobile device product line with a meta-programming technique of XVCL. We started by building a simple product line architecture for PC-based systems. Then, we evolved the architecture to accommodate features required on mobile device platforms. XVCL is based on “composition with adaptation” technique, that simplifies target system generation, evolution of the architecture and allows us to address runtime performance issues (both processing speed and memory consumption), so important on mobile device platforms.

Our product line architecture consists of 33 generic meta-components, comprising 560 LOC. Sizes of systems that we can generate from this base of generic code are as follows: J2SE version of the City Guide System – 388 LOC, P800 – 198 LOC, and Pocket PC – 371 LOC. Instead of having to maintain 957 LOC (388+198+371), we need maintain only 560 LOC covering features from

all the members of this product line. In the case study described in this paper, we can say that applying XVCL brings a benefit of managing a product line from a common base of generic code, without complicating programs in any significant way.

In other case studies, we applied XVCL to eliminate redundant code in order to improve maintainability (e.g. [29]). We achieved substantial code reduction (up to 68%), but the XVCL solution also introduced extra complexities. Being generic, meta-components are always more difficult to understand than concrete components. The verbose XML syntax also has negative impact on the understanding of meta-components. For the ease of use and understanding of meta-components, we developed a number of tools, such as the debugger [30], backward propagation tool (BP tool) [31], etc to alleviate these problems.

The debugging environment, which is an Interactive XVCL Processor (IXP for short), helps us trace and debug the XVCL processing sequence in an interactive way. With IXP, the user can:

- feed a value to an undefined meta-variable, or designate a path for an undefined adapting meta-component if IXP is halted due to errors of not finding meta-components
- set debugging points for future debugging purpose
- modify values of meta-variables
- modify values of attributes of the suspended XVCL command

During early stages of x-framework development, there may be lots of errors in the generated code. Rather than fixing errors directly in meta-components, it is often preferable to amend the generated code, recompile, fix again. At the end of this cycle, a programmer may choose to propagate changes back to the affected meta-components. BP (Backward Propagation) Tool is designed to accommodate such a scenario. It helps the user propagate modifications from the generated code back to meta-components.

Other tools are in development, such as a smart editor which hides the complexity of XVCL from the end user and helps to “think in XVCL” and “work with XVCL”.

We also explored the reengineering approach with aspect-oriented programming (AOP) [32] techniques such as aspectJ [33]. We found that while AOP is very strong in terms of advanced separation of concerns and addressing the cross-cutting concerns, it has relatively weak configurability and reusability which are vital to a product line. However, a very promising area is that of combining AOP and Frame technology in order to produce *framed aspects* [34]. We believe that framed aspects can alleviate many of the problems associated with crosscutting code and improve support for evolution in product line frameworks.

11. Conclusions

In this paper, we described how we made an existing PC-based City Guide system available on a wide range of mobile devices, in a cost-effective way. We applied “reengineering into a product line architecture” approach to achieve the goal. Our product line architecture facilitates reuse via generation – we generate specific City Guide Systems for target platforms including PC, Pocket PC and other mobile devices, from generic meta-components that form the City Guide System product line architecture. In our project, we used a meta-programming technique of XVCL to build a product line architecture for City Guide Systems.

There is a growing demand to port existing PC-based software systems to mobile device platforms. Systems running on mobile devices share basic characteristics with their PC-based counterparts, but differ from them in details of user interfaces, application models, etc. Systems running on mobile devices must also perform well using less memory than PC-based systems. Mobile devices themselves are different from each other in many ways too. The solution described in the paper addresses these problems in a practical and flexible manner.

References

- [1] XVCL homepage, <http://fxvcl.sourceforge.net>.
- [2] Wong, T.W.; Jarzabek, S.; Myat Swe, S.; Shen, R. and Zhang, H.Y. “XML Implementation of Frame Processor,” Proc. ACM Symposium on Software Reusability, SSR’01, Toronto, Canada, May 2001, pp. 164-172.
- [3] Jarzabek, S. and Zhang, H.Y. “XML-based Method and Tool for Handling Variant Requirements in Domain Models”. In *Proc. of 5th IEEE International Symposium on Requirements Engineering, RE’01, IEEE Press*, August 2001, Toronto, Canada, pp. 166-173.
- [4] Soe, M.S.; Zhang, H.Y. and Jarzabek, S. “XVCL: A Tutorial”. *Proc. of 14th Int. Conf. on Software Engineering and Knowledge Engineering, SEKE’02*, ACM Press, July 2002, Italy, pp. 341-349.
- [5] Holger Bar, et al. The FAMOOS Object-Oriented Reengineering Handbook, Version: October 15, 1999 <http://www.iam.unibe.ch/famoos/handbook/>.
- [6] Sneed, H. “Reverse Engineering as a Bridge to CASE”. *Proc. 2nd Working Conf. on Reverse Engineering, WCRE’95*, IEEE Computer Society Press, Los Alamitos, Toronto, pp. 300-313.
- [7] Rock-Evans, R. and Hales, K. *Reverse Engineering: Markets, Methods and Tools. Ovum Report* vol. 1, Ovum Ltd. London, England.
- [8] Biggerstaff, T. “Design Recovery for Maintenance and Reuse”. *IEEE Computer*, 7, 1989. pp. 36-49.
- [9] Canfora, G.; Cimitile, A. and de Carlini, U. “A Logic-Based Approach to Reverse Engineering Tools Production”. *IEEE Transactions on Software Engineering*, vol. 18, no. 12, 1992. pp. 1053-1064.
- [10] Jarzabek, S. and Ling, T.W. “Model-based Support for Business Reengineering”. *Journal of Information and Software Technology*, vol. 38, No. 5, May 1996, pp. 355-374.
- [11] Gianluigi Caldiera and V.R. Basili, “Reengineering Existing Software for Reusability”, UMIACS-TR-90-30, CS-TR-2419, February 1990.
- [12] Thomson, R.; Huff, K.E.; Gish, J.W.; “Maximizing reuse during reengineering”. In *Proc. Of Third International Conference on Software Reuse: Advances in Software Reusability*, 1-4 Nov 1994. pp. 16 -23.
- [13] Canfora, G.; Fasolino, A.R.; Tortorella, M.; “Towards reengineering in reuse reengineering processes”. In *Proc. Of International Conference on Software Maintenance*, 17-20 Oct 1995. pp. 147 -156.
- [14] Masiero, P.C.; Braga, R.T.V.; “Legacy systems reengineering using software patterns”, Computer Science Society, 1999. *Proc. Of SCCC ’99. XIX International Conference of the Chilean*, 1999 pp. 160 -169.
- [15] Cagnin, M.I.; Penteadro, R.; Braga, R.T.V.; Masiero, P.C.; “Reengineering using design patterns”. *Proc. Of Seventh Working Conference on Reverse Engineering*, 2000 pp. 118 -127.
- [16] Wendorff, P.; “Assessment of design patterns during software reengineering: lessons learned from a large commercial project”. *Fifth European Conference on Software Maintenance and Reengineering*, 2001 pp. 77 - 84.
- [17] Bromling, S. et al "Pattern-based Parallel Programming", submitted to the *2002 International Conference on Parallel Programming (ICPP-02)*, Vancouver, British Columbia, August 2002.
- [18] Bojic, D.; Velasevic, D.; “A use-case driven method of architecture recovery for program understanding and reuse reengineering”. *Proceedings of the Fourth European Software Maintenance and Reengineering*. Feb 2000 pp. 23 -31.
- [19] Bengtsson, P.; Bosch, J.; “Scenario-based software architecture reengineering”. *Proc. Of Fifth International Conference on Software Reuse*. 2-5 Jun 1998. pp. 308 - 317.
- [20] Riva, C.; Rodriguez, J.V.; “Combining static and dynamic views for architecture reconstruction”. *Proc. Of Sixth European Conference on Software Maintenance and Reengineering*. 2002 pp. 47 -55.
- [21] Clements, P. & Northrop, L. “Software Product Lines: Practices and Patterns”. Boston, MA: Addison-Wesley, 2001.
- [22] Stoermer, C.; O'Brien, L.; Verhoef, C.; “Practice patterns for architecture reconstruction”. *Proc. Of Ninth Working Conference on Reverse Engineering*. 2002 pp. 151 -160.

- [23] Muller, Hausi; et al. "Reverse Engineering: A Roadmap". *Future of Software Engineering*. New York: ACM Press Books, 2000.
- [24] Shaw M. and Garlan D. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, 1996.
- [25] Bergey J.; O'Brien, Liam; Smith, Dennis. "Mining Existing Assets for Software Product Lines". CMU/SEI-2000-TN-008, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- [26] Eric Giguere. *Understanding J2ME Application Models*. October 2002. <http://wireless.java.sun.com/midp/articles/models/>.
- [27] Kang, K et al. "Feature-Oriented Domain Analysis (FODA) Feasibility Study". CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
- [28] Czarnecki, K. and Eisenecker, U., 2000. *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley.
- [29] Jarzabek, S. and Shubiao, L. "Eliminating Redundancies with a "Composition with Adaptation" Meta-programming Technique," accepted for *ESEC-FSE'03, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, September 2003, Helsinki.
- [30] Chen, Liang. *Debugging Meta-Programs in XVCL. Honours Year Project Report*, Department of Computer Science, National University of Singapore, Singapore. 2003.
- [31] Peng, Chuanning. *BP Tool (Backward Propagation of Changes). Research Presentation*, Department of Computer Science, National University of Singapore, Singapore. 2003.
- [32] Kiczales, G.; et al. "Aspect Oriented Programming", Proc. of European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag, LNCS 1241, June 1997. pp. 220-242.
- [33] Xerox PARC, USA, AspectJ Home Page, <http://aspectj.org/>.
- [34] Loughran, Neil; Rashid, Awais; Zhang, Weishan and Jarzabek, Stan. "Supporting Product Line Evolution with Framed Aspects". Submitted to *PFE-5, Fifth International Workshop on Product Family Engineering*. Siena, Italy. November 2003.