Demo Proposal

1 General Information

Demostration title: **FLiP – Product Line Derivation Tool**
Type of demonstration : Forum

Contact person: Paulo Borba, phmb@cin.ufpe.br, Informatics Center – UFPE, Universidade Federal de Pernambuco, Centro de Informática, Caixa Postal 7851, CEP 50732-970, Recife, PE, Brazil, +55 -81 2126 8430 ex. 4323

Other presenters: Fernando Calheiros, Meantime Mobile Creations, fernando.calheiros@cesar.org.br
Vilmar Nepomuceno, Meantime Mobile Creations, vilmar.nepomuceno@cesar.org.br
Andrea Menezes, Meantime Mobile Creations, andrea.menezes@cesar.org.br
Sérgio Soares, Computing Systems Department – UPE, sergio@dsc.upe.br
Vander Alves, Lancaster University, v.alves@comp.lancs.ac.uk

2 Abstract

With the growing academic and industrial interest in Software Product Lines, one area demanding special attention is tool support development, which is a pre-requisite for widespread software product lines practices adoption. In this demo, we present a suite of tools consisted of 3 modules: FLiPEx, FLiPG and FLiPC. FLiPEx is a refactoring tool that implements code transformations for extracting product variations from Java classes to AspectJ aspects. FLiPEx interacts with FLiPG, which integrates with Feature Model tool for updating a software product lines feature model accordingly to code transformations, which, for example, might turn mandatory into optional features. FLiPG interacts with the FLiPC tool that is responsible for using the information stored in FLiPG to build the final products. FLiP has been designed and tested in the context of real mobile game software product lines.

3 Description

The extractive and the reactive Software Product Line (SPL) [7] adoption strategies [12] involve, respectively, bootstrapping existing products into a SPL and extending an existing SPL to encompass another product. In both cases, product line refactorings [1,2] are useful to guide the SPL derivation process by extracting product variations and appropriately structuring them. They also help to assure the safety of the whole process by preserving SPL configurability [1] — the resulting SPL has at least the same instances than the initial set of independent products being bootstrapped or the initial SPL being extended.

In single system refactoring, ensuring safety and effectiveness of a refactoring process already requires automation, in practice, in the SPL context, where complexity increases due to the need to manage a high number of variants, such support becomes even more indispensable. In this context, we describe FLiP, which is a suite of tools implemented as Eclipse plugins, which is consisted of 3 modules: FLiPEx, FLiPG and FLiPC.

FLiPEx is a refactoring tool that implements code transformations [2] for extracting product variations from Java classes to AspectJ aspects. Aspects are used since we need a better modularization technique. FLiPEx interacts with FLiPG, which integrates with Feature Model (FM) [14] tools for updating a SPL FM accordingly to code transformations, which, for example, might turn mandatory into optional features. FLiPG interacts with the FLiPC tool that is responsible for using the information stored in FLiPG to build the final products. FLiP has been designed and tested in the context of mobile game SPLs.

3.1 Mobile games environment

Mobile games (and mobile applications, in general) must adhere to strong portability requirements. This stems from business constraints: in order to target more users, owning different kinds of devices, service carriers typically demand that a single application be deployed in a dozen or more platforms. Each platform generally provides vendor-specific Application Programming Interfaces (APIs) with mandatory or optional advanced features, which the developer is likely to use in order to improve game quality. In addition, devices have memory and display constraints, which further require the developer to optimize the application. In either case, adapting the game for each platform is mandatory.

We analyze and manage the specific kinds of variations arising from platform variation, where platform means a combination of MIDP [15][15], vendor-specific API, and hardware constraints. Accordingly, some of the specific challenges for managing variation in this domain are the following: UI features (such as screen size, number of colors, pixel depth, sound, keypad display); available memory and maximum application size; different profile versions (MIDP 1.0 and MIDP 2.0); different implementation of the same profile; proprietary APIs and optional packages; known device-specific bugs; different idioms.

These specific kinds of variation tend to be considerably fine-grained such that they generally crosscut the game core and are tangled with other kinds of variation. This suggests AOP as a suitable candidate for modularizing these variations. In fact, we used AspectJ [11][8], the most widely used aspect-oriented extension to Java, to implement the product line variations.

3.2 Refactorings

Contrary to the proactive approach, which is more like the waterfall model, we rely here on a combination of the extractive and the reactive approaches. There are a number of reasons for this. First, small to medium-sized organizations, which still want to benefit from PLs, cannot afford the high cost incurred in adopting the proactive approach. Second, in domains such as mobile game development, the development cycle is so short that proactive planning cannot be completed. Third, there are risks associated in the proactive approach, because the scope may become invalid due to new requirements.

Our method first bootstraps the PL and then evolves it with a reactive approach. Initially, there may be one or more independent products, which are refactored in order to expose variations to bootstrap the PL. Next, the PL scope is extended to encompass another product: the PL reacts to accommodate the new variant. During this step, refactorings are performed to maintain the existing product, and a PL extension is used to add a new variant. The PL may react to further extension or refactoring.

The method is systematic because it relies on a collection of provided refactorings. Such refactorings are described in terms of templates, which are a concise and declarative way to specify program transformations. In addition, refactoring preconditions (a frequently subtle issue) are more clearly organized and not tangled with the transformation itself. Furthermore, the refactorings can be systematically derived from more elementary and simpler programming laws [8][8]. These laws are appropriate because they are considerably simpler than most refactorings, involving only localized program changes, with each one focusing on a specific language construct.

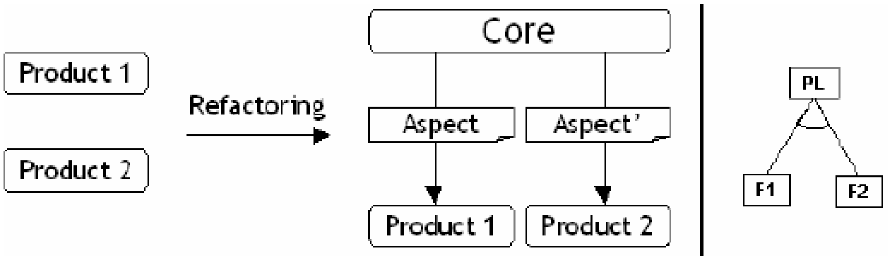


Fig 1. Bootstrapping the Product Line.

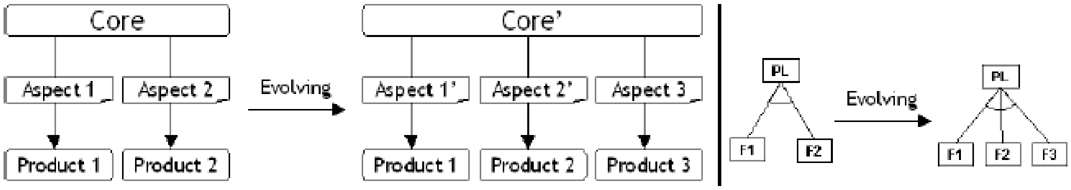


Fig 2. Evolving the Product Line.

3.3 Instance generation

After each refactoring, information about this process is stored, such as new aspects generated by the extraction and the features related with these aspects (see Figure 1). The tool responsible for managing this information currently is the Pure::Variants [14], but others tools can be used to manipulate the information. All this information is compiled to generate product Line instances that will be used by the build system in FLiP to create the final products.

3.4 Architecture

An overview of the architecture of the FLIP tool suite, built upon the Eclipse plugin platform is presented in Figure 3. The figure shows the relationship between FLiPEx, FLiPG, the Eclipse framework, and a Feature Model tool, currently Pure::Variants [14]. FLiP was built with extensibility as one of its most important design features.

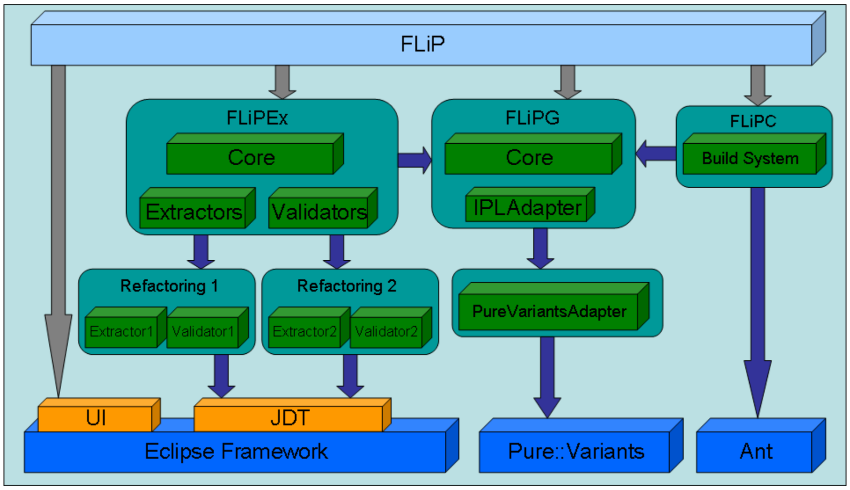


Figure 3 FLiP Architecture

All parts of the FLiPEx schema are independent plugins, *Core* is the main plugin and gives support to the *Extractors* and *Validators*. Each refactoring has an *Extractor*, which is responsible for removing the code from the Java class and creating the AspectJ code that inserts the variation back at its original site. Each *Extractor* has corresponding *Validators*, which are responsible for analyzing the selected code to check if it meets all the preconditions of its refactoring [2].

The validation system was designed to avoid repetition of validations, i.e., if two or more different refactorings share some preconditions, these preconditions can be encoded in one *Validator*, which will run only once, but its results will be available to each *Extractor* that depends on it.

The FLiPG module provides integration with the underlying feature model tool. This integration is made using the available plugin that provides an implementation of the adapter interface that specifies which services FLiPG needs from the feature model tool, such as retrieving the list of features from the feature model, the components in the configuration knowledge, and updating both.

All refactorings are added to the platform through the extractors and validators extension points that FLiPEx exports to plugin developers. Also, additional feature model tools can be used with FLiP, since it exports an extension point for feature model tools adapters, therefore integration is a simple matter of implementing the interface with the services FLiPG needs, but providing these services using the different feature model tools APIs.

4 Related Work

Prior work has evaluated the use of AOP for building J2ME product lines [13]. The use of refactorings to build and evolve a J2ME SPL has also been evaluated [3]. Other approaches for porting are subdivided in the following categories: preprocessing, general guidelines, specific guidelines, semi-automatic services, and formal approaches.

Tools like Antenna [17] and J2ME Polish [18] provide a preprocessing feature by which guidelines define a conditional compilation of the source code according to the device in question. However, the use of compilation directives may compromise source code legibility.

The guidelines approach can be specific, documenting source and target devices [13], or general, offering broader guidelines [9] that involve researching the target device, architecture organization and source code transformation.

Another alternative is a combination of developing the code according to some guidelines and automatic transformations on the code [16].

Formal approaches [5,6,10] propose an abstract specification of GUI elements, device characteristics, and user interface scenarios. However, this approach constrains the freedom of GUI development, and does not take into consideration issues like application size constraints.

5 References

[1] V. Alves et al. Refactoring Product Lines. *GPCE'06*, October 2006. ACM Press.

[2] V. Alves et. al. Extracting and Evolving Mobile Games Product Lines. *9th SPLC'05*. September 2005. Springer-Verlag.

[3] V. Alves. Implementing Software Product Line Adoption Strategies. PhD thesis, Informatics Center, Federal Universisty of Pernambuco, Recife, Brazil. March 2007.

[4] M. Anastasopoulos and D. Muthig. An evaluation of aspect-oriented programming as a product line implementation technology. International Conference on Software Reuse (ICSR), 2004.

[5] R. Cardone et. al. Using mixins to build flexible widgets. AOSD ’02: International Conference on Aspect-Oriented Software Development, pages 76–85. ACM Press, 2002.

[6] H. hua Chu et. al. Roam, a seamless application framework. Journal of Systems and Software, 69(3):209–226, 2004.

[7] P. Clements and L. Northrop. Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.

[8] L. Cole and P. Borba. Deriving refactorings for AspectJ. In AOSD’05: International Conference on Aspect-Oriented Software Development, pages 123–134, 2005.

[9] X. Facon. Porting Your MIDlets to New Devices. http://www.microjava.com/articles/techtalk/, 2004.

[10] K. Gajos and Daniel S. Weld. Supple: automatically generating user interfaces. IUI ’04: 9th international conference on Intelligent user interface, pages 93–100. ACM Press, 2004.

[11] G. Kiczales et. al. Getting Started with AspectJ. CACM, 44(10):59–65, October 2001.

[12] C. Krueger. Easing the transition to software mass customization. *4th International Workshop on Software Product-Family Engineering*. Spain, October 2001.

[13] Motorola. Porting guide: Motorola i95cl to T720. http://www.microjava.com/articles/-MJN Porting.Guide i95cl-T720.pdf, 2004.

[14] Pure::Variants. http://www.pureystems.com/Variant_ Management.49.0.html, 2007

[15] Sun Microsystems. JSR-37 Mobile Information Device Profile (MIDP).

[16] Tira Wireless. TiraJump. World Wide Web, http://www.tirawireless.com/jump/, 2004.

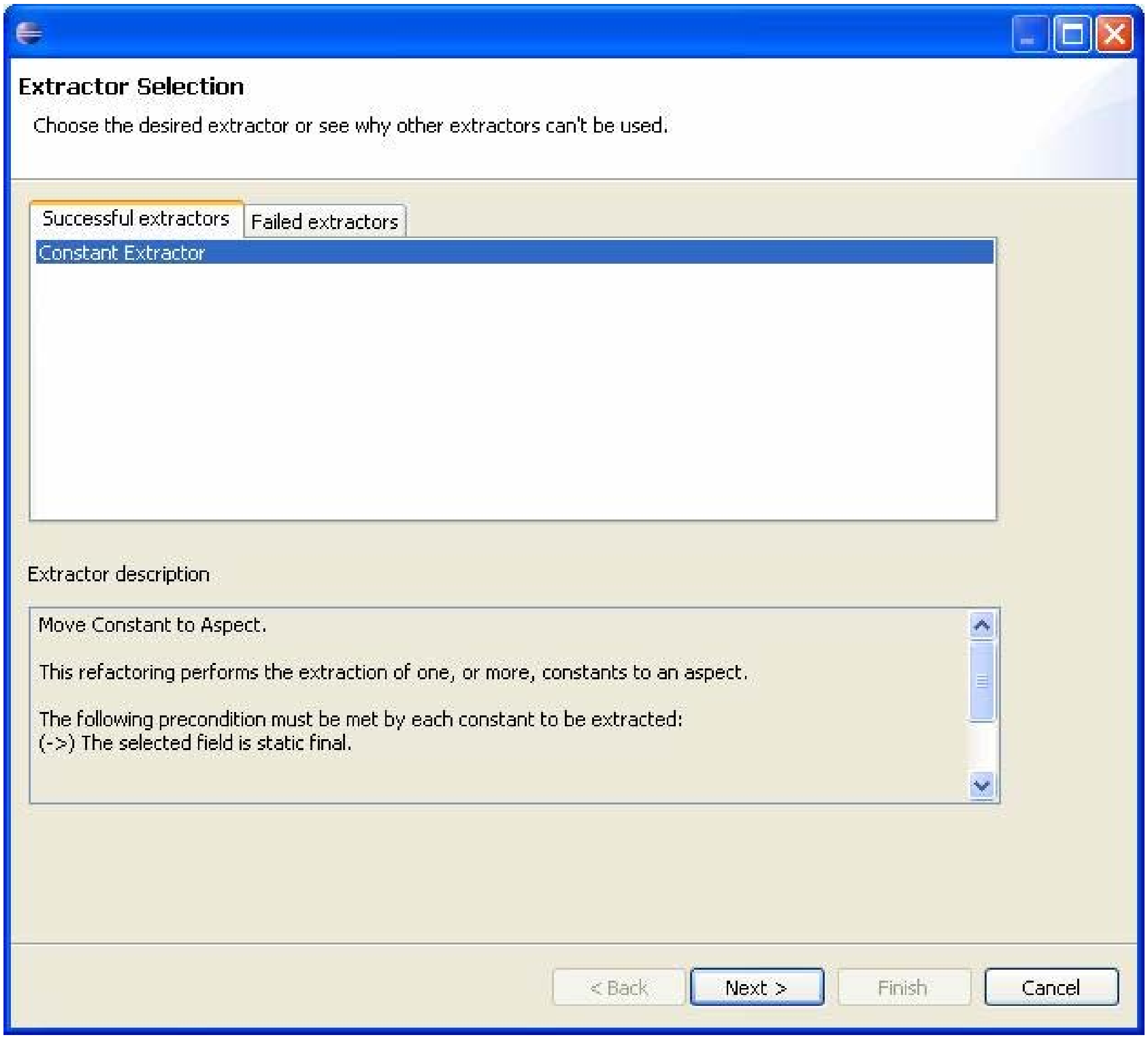
[17] World Wide Web, http://antenna.sourceforge.net. Antenna Preprocessor, 2004.

[18] World Wide Web, http://www.j2mepolish.org. J2ME Polish, 2004.

6 Hardware and presentation requirements

This demo requires only a multimedia projector, since we will use our own PC.

Demo Screenshots



Extractor Selection

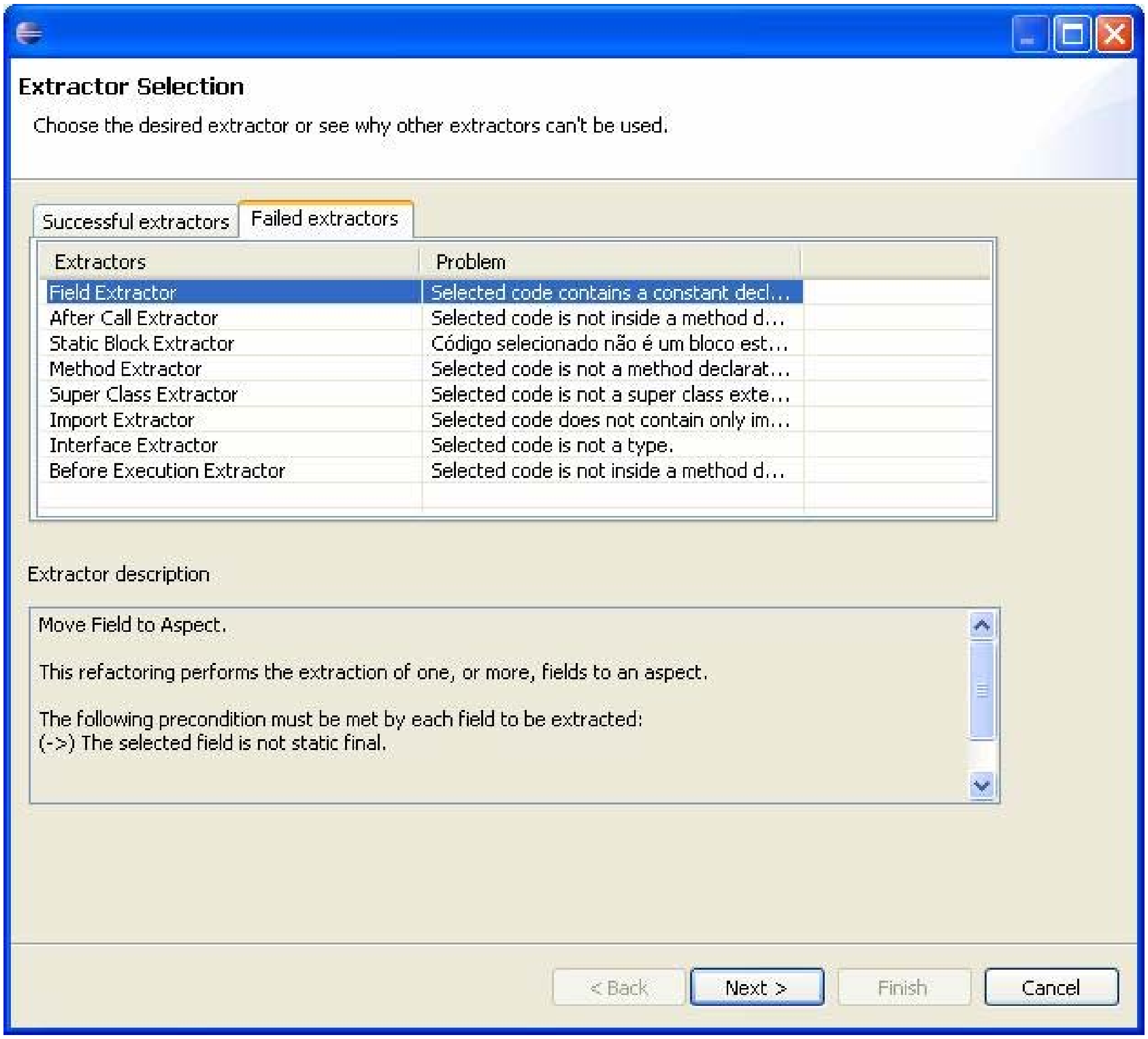
Choose the desired extractor or see why other extractors can't be used.

Extractor description

Move Constant to Aspect.

This refactoring performs the extraction of one, or more, constants to an aspect.

The following precondition must be met by each constant to be extracted:
(->) The selected field is static final.



Extractor Selection

Choose the desired extractor or see why other extractors can't be used.

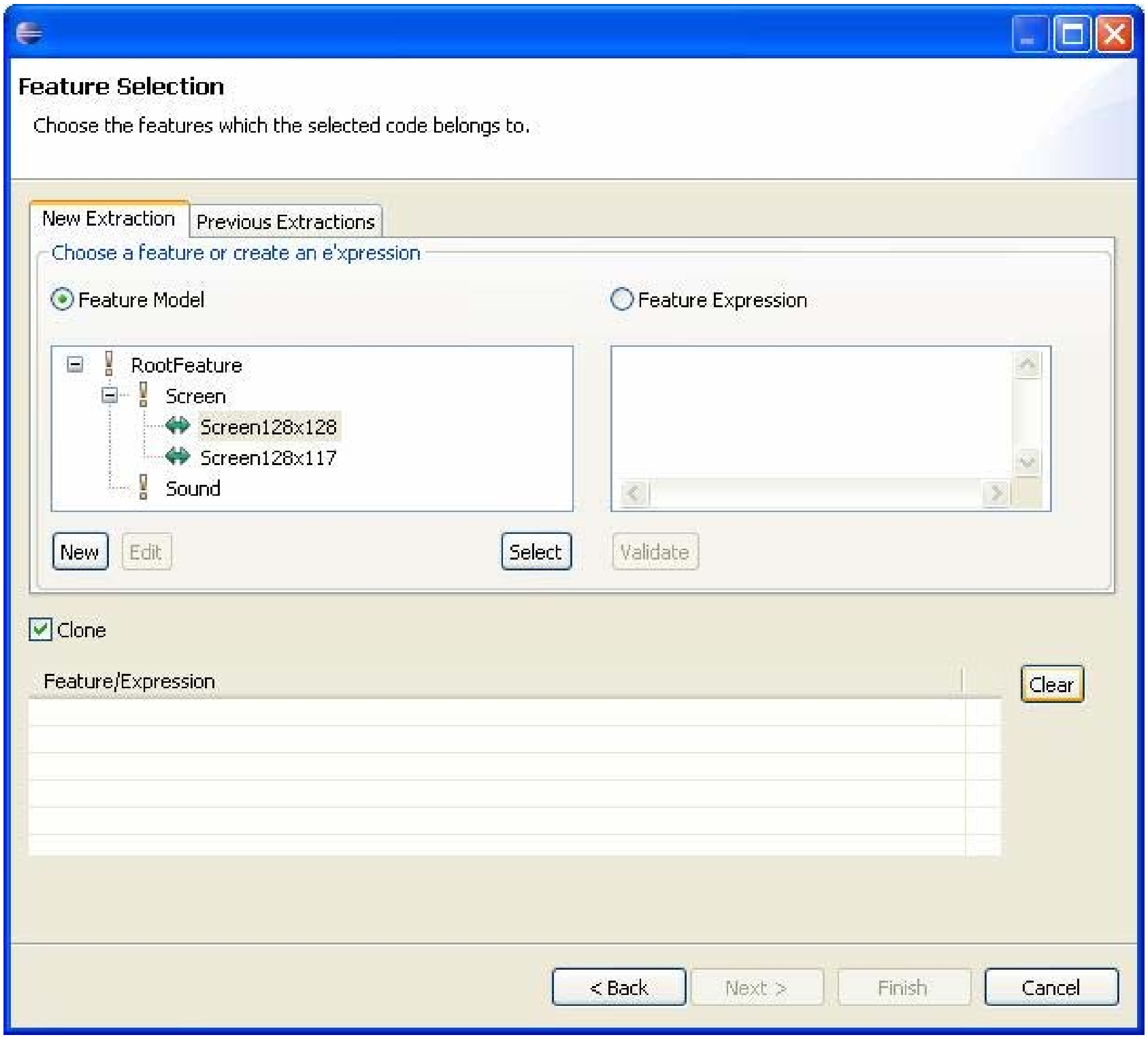
Extractors	Problem
Field Extractor	Selected code contains a constant decl...
After Call Extractor	Selected code is not inside a method d...
Static Block Extractor	Código selecionado não é um bloco est...
Method Extractor	Selected code is not a method declarat...
Super Class Extractor	Selected code is not a super class exte...
Import Extractor	Selected code does not contain only im...
Interface Extractor	Selected code is not a type.
Before Execution Extractor	Selected code is not inside a method d...

Extractor description

Move Field to Aspect.

This refactoring performs the extraction of one, or more, fields to an aspect.

The following precondition must be met by each field to be extracted:
(->) The selected field is not static final.



Feature Selection

Choose the features which the selected code belongs to.

Choose a feature or create an e'xpression

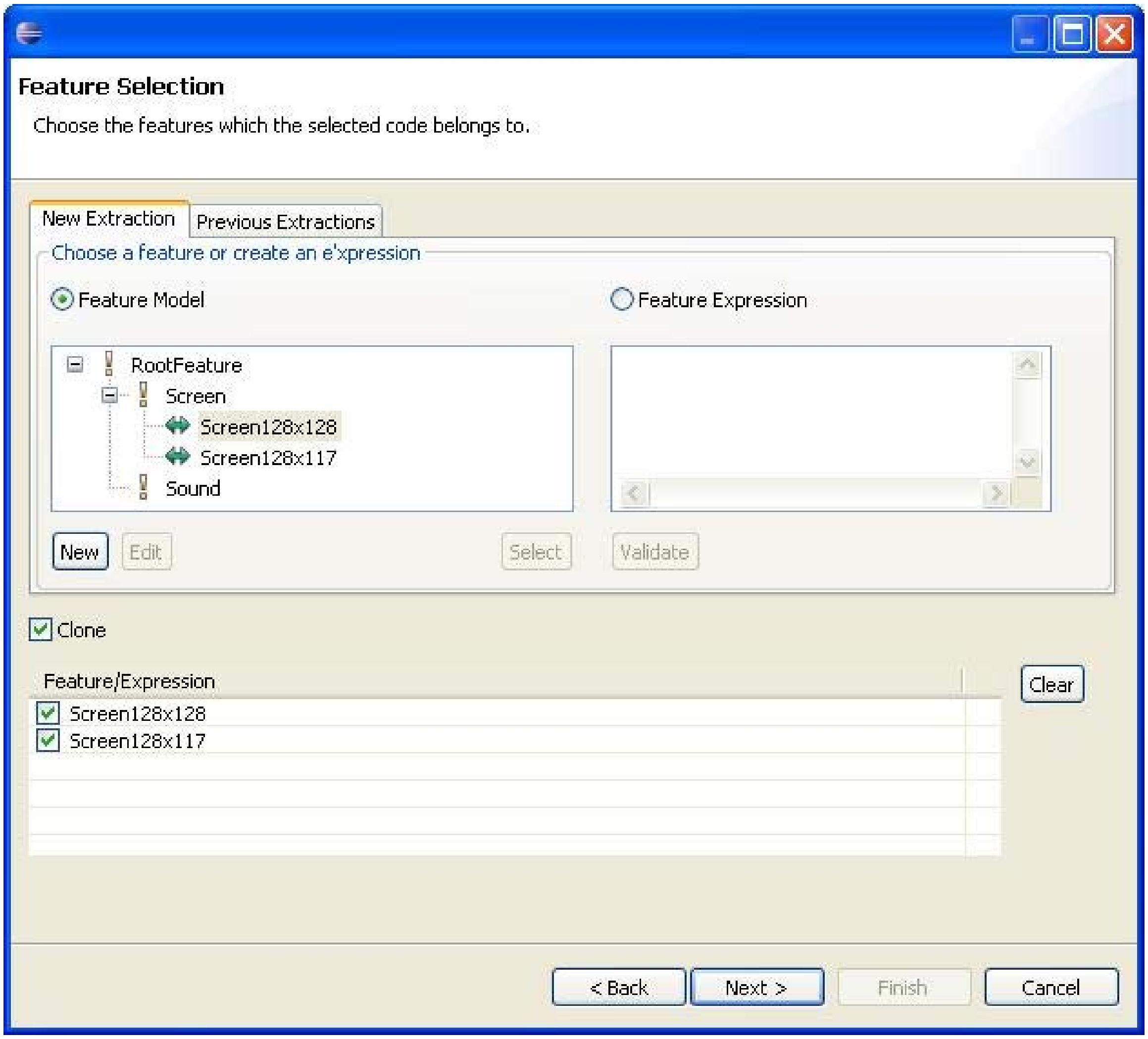
● Feature Model

○ Feature Expression

```
⊟ RootFeature
   ⊟ Screen
       Screen128x128
       Screen128x117
     Sound
```

☑ Clone

Feature/Expression



Feature Selection

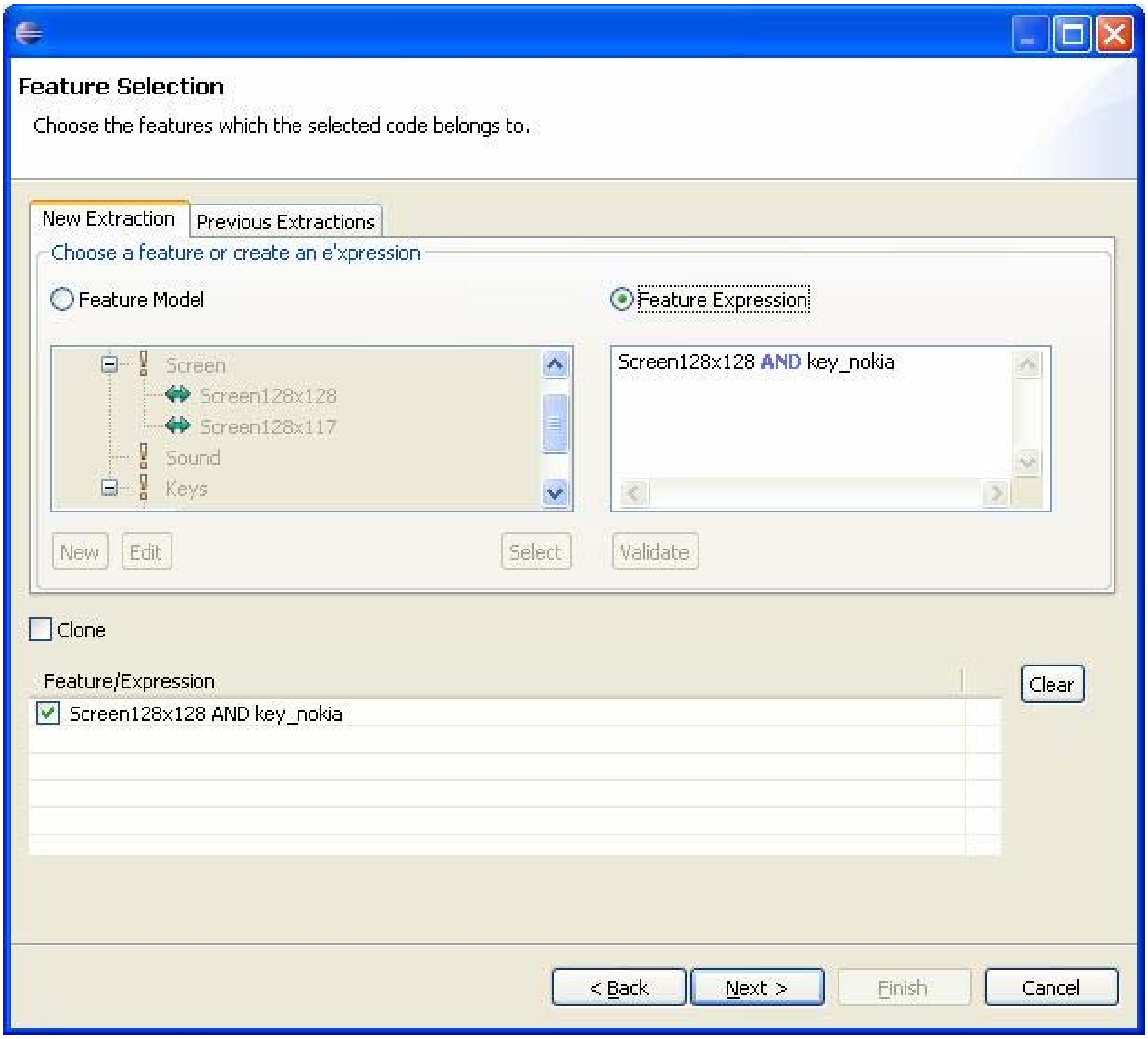
Choose the features which the selected code belongs to.

● Feature Model ○ Feature Expression

- RootFeature
  - Screen
    - Screen128x128
    - Screen128x117
  - Sound

☑ Clone

Feature/Expression
☑ Screen128x128
☑ Screen128x117



Feature Selection

Choose the features which the selected code belongs to.

New Extraction | Previous Extractions

Choose a feature or create an e'xpression

○ Feature Model

● Feature Expression

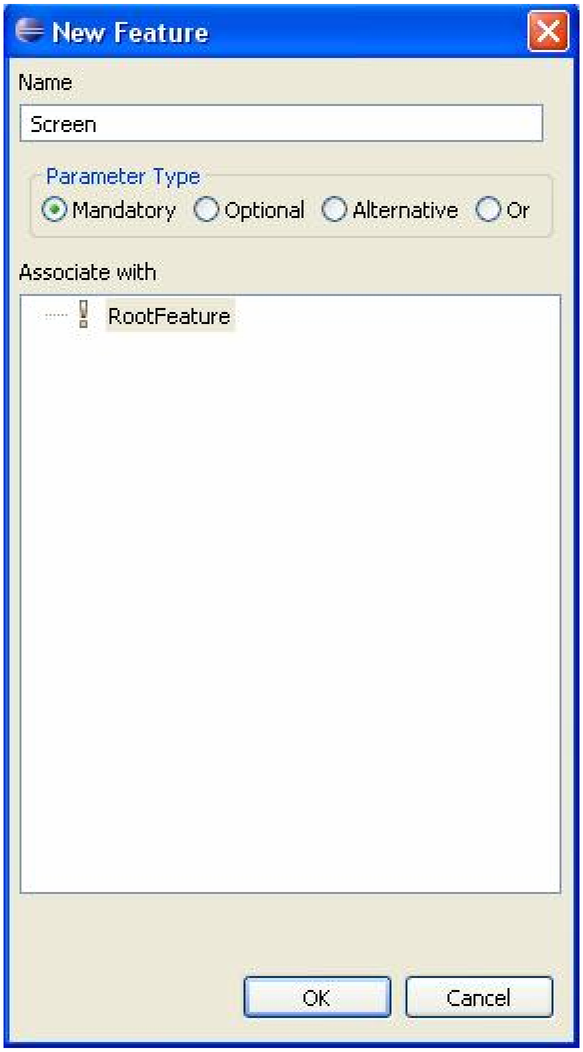
Screen
Screen128x128
Screen128x117
Sound
Keys

Screen128x128 AND key_nokia

□ Clone

Feature/Expression

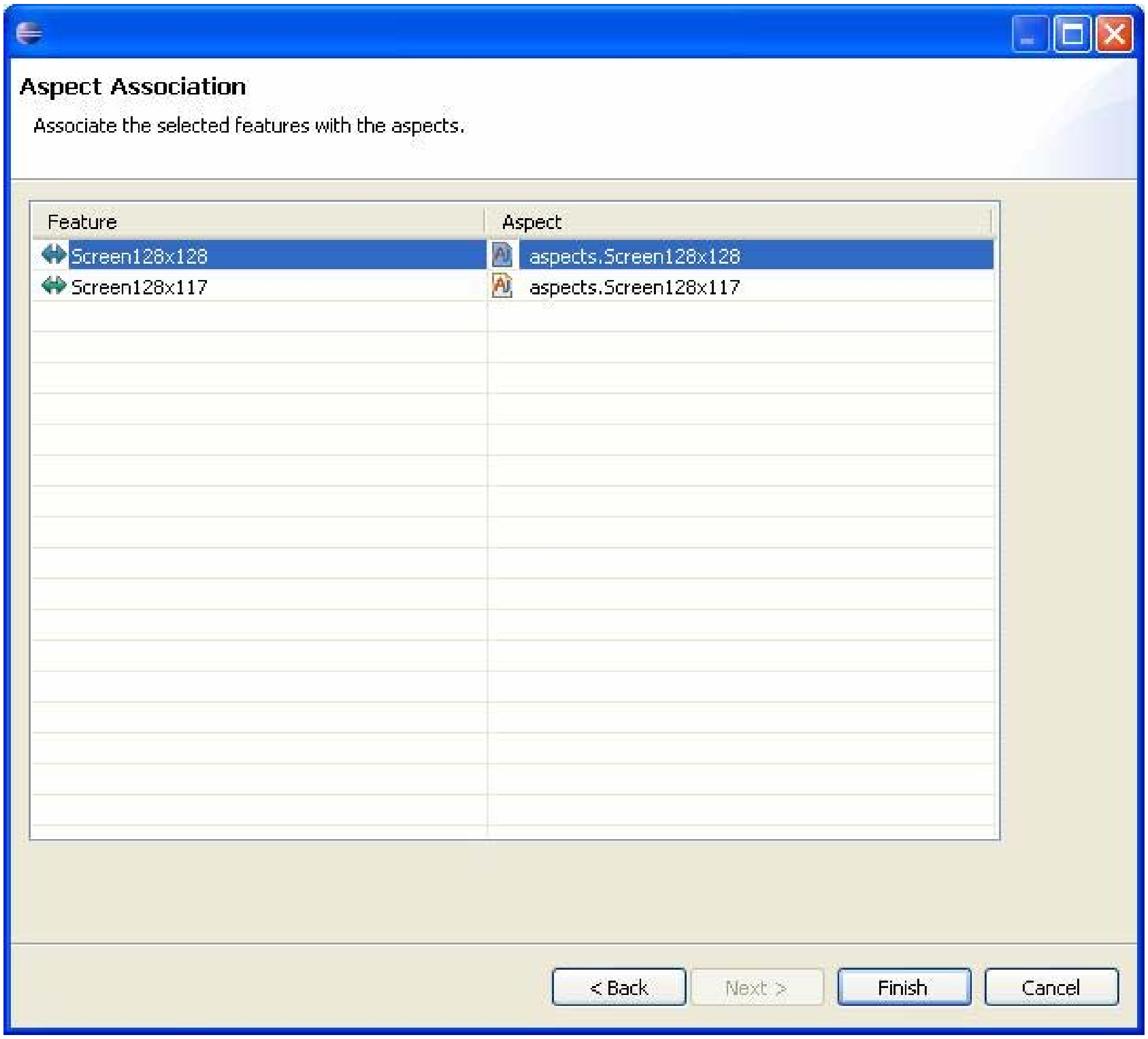
☑ Screen128x128 AND key_nokia



Name
Screen

Parameter Type
● Mandatory ○ Optional ○ Alternative ○ Or

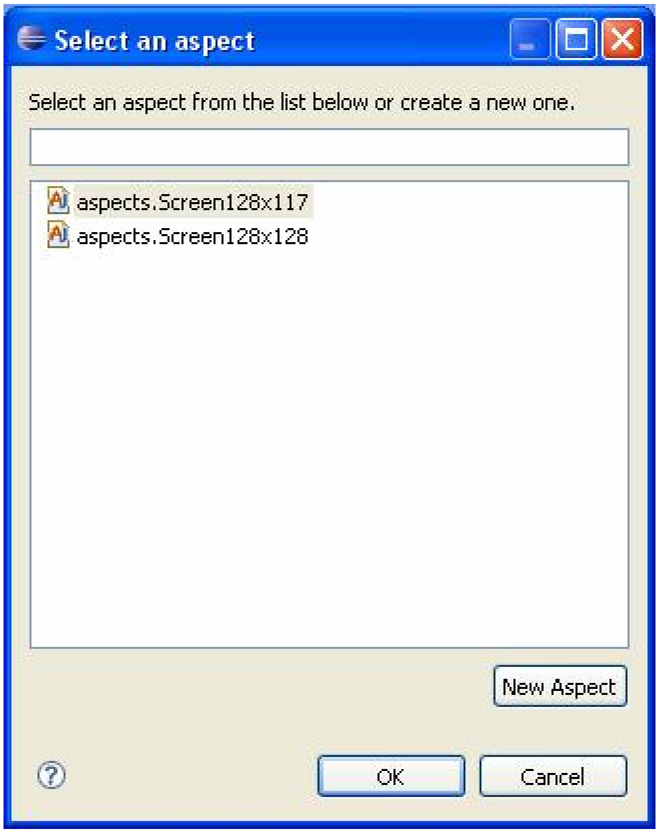
Associate with
RootFeature



Aspect Association

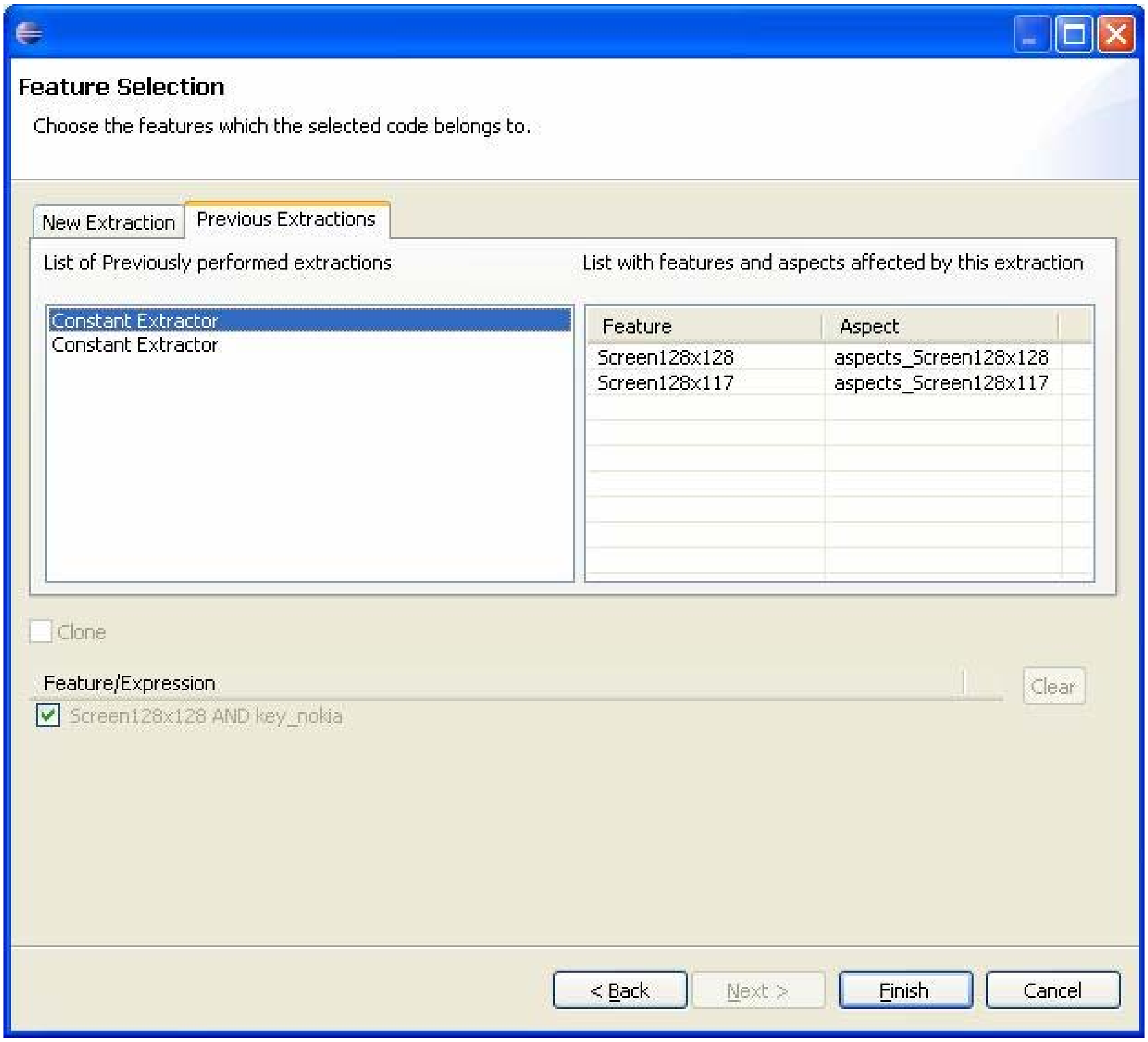
Associate the selected features with the aspects.

Feature	Aspect
Screen128x128	aspects.Screen128x128
Screen128x117	aspects.Screen128x117



Select an aspect from the list below or create a new one.

aspects.Screen128x117
aspects.Screen128x128



Feature Selection

Choose the features which the selected code belongs to.

List of Previously performed extractions

- Constant Extractor
- Constant Extractor

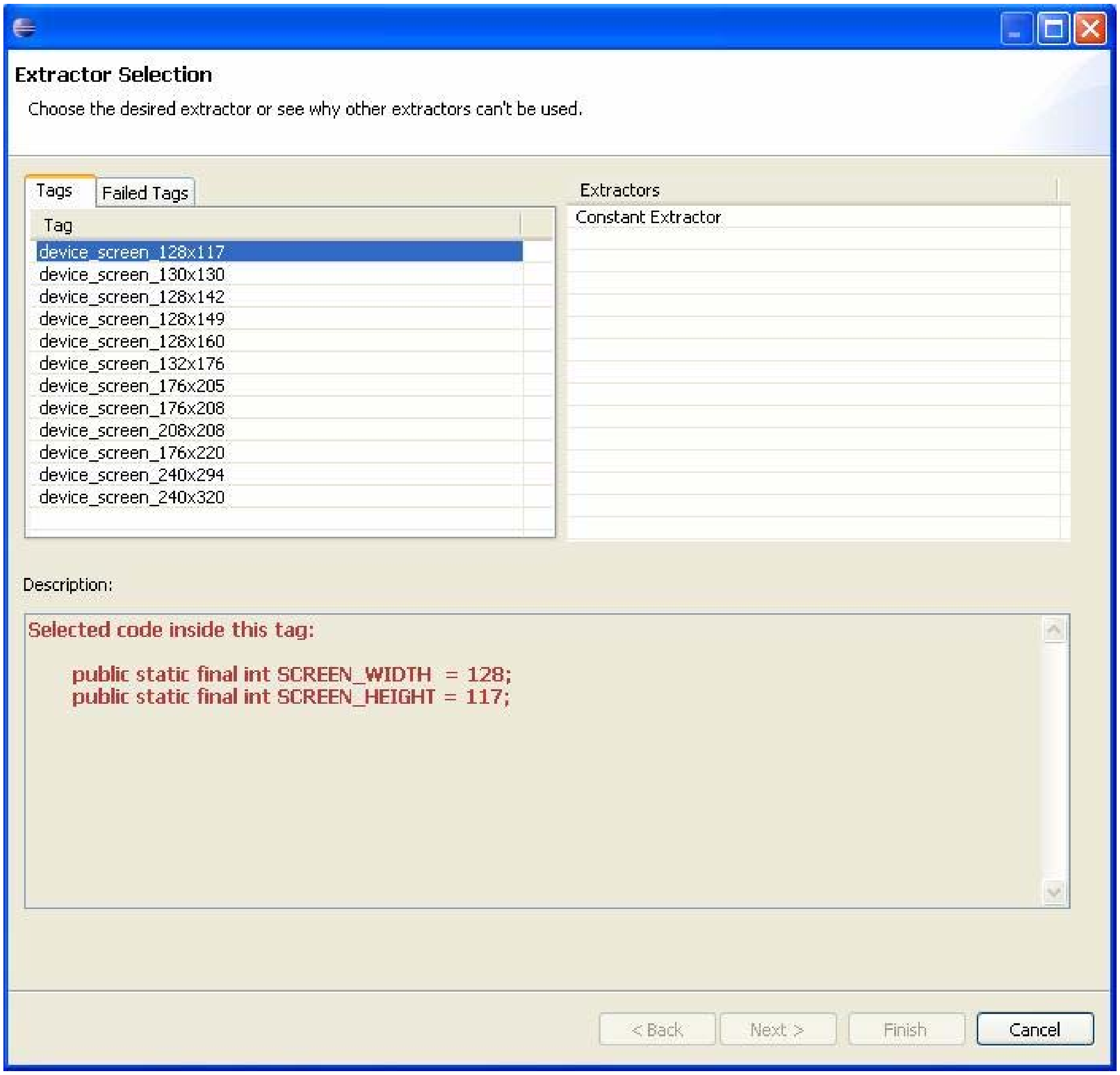
List with features and aspects affected by this extraction

Feature	Aspect
Screen128x128	aspects_Screen128x128
Screen128x117	aspects_Screen128x117

□ Clone

Feature/Expression

☑ Screen128x128 AND key_nokia



Extractor Selection

Choose the desired extractor or see why other extractors can't be used.

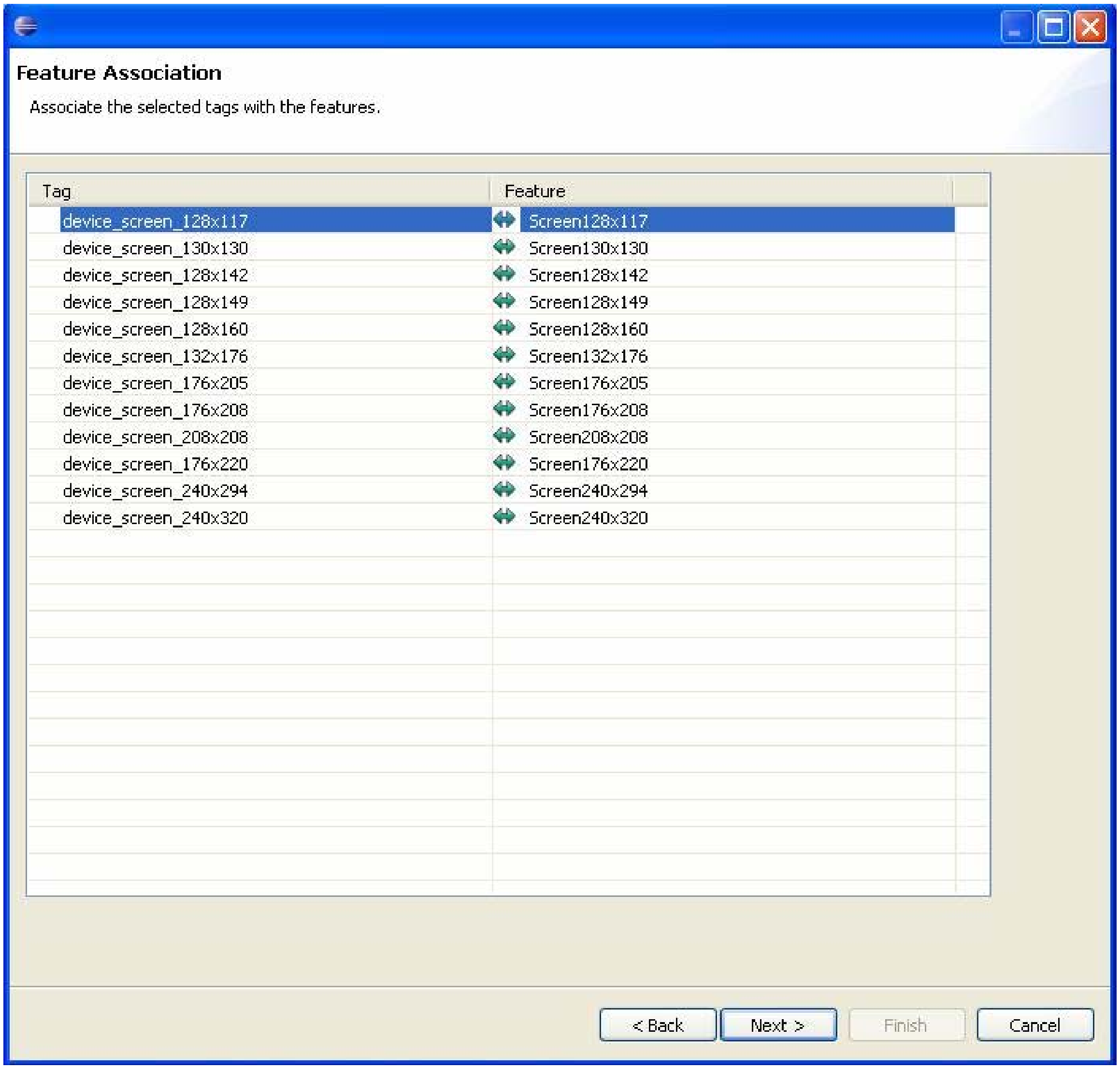
Tag
device_screen_128x117
device_screen_130x130
device_screen_128x142
device_screen_128x149
device_screen_128x160
device_screen_132x176
device_screen_176x205
device_screen_176x208
device_screen_208x208
device_screen_176x220
device_screen_240x294
device_screen_240x320

Extractors
Constant Extractor

Description:

Selected code inside this tag:

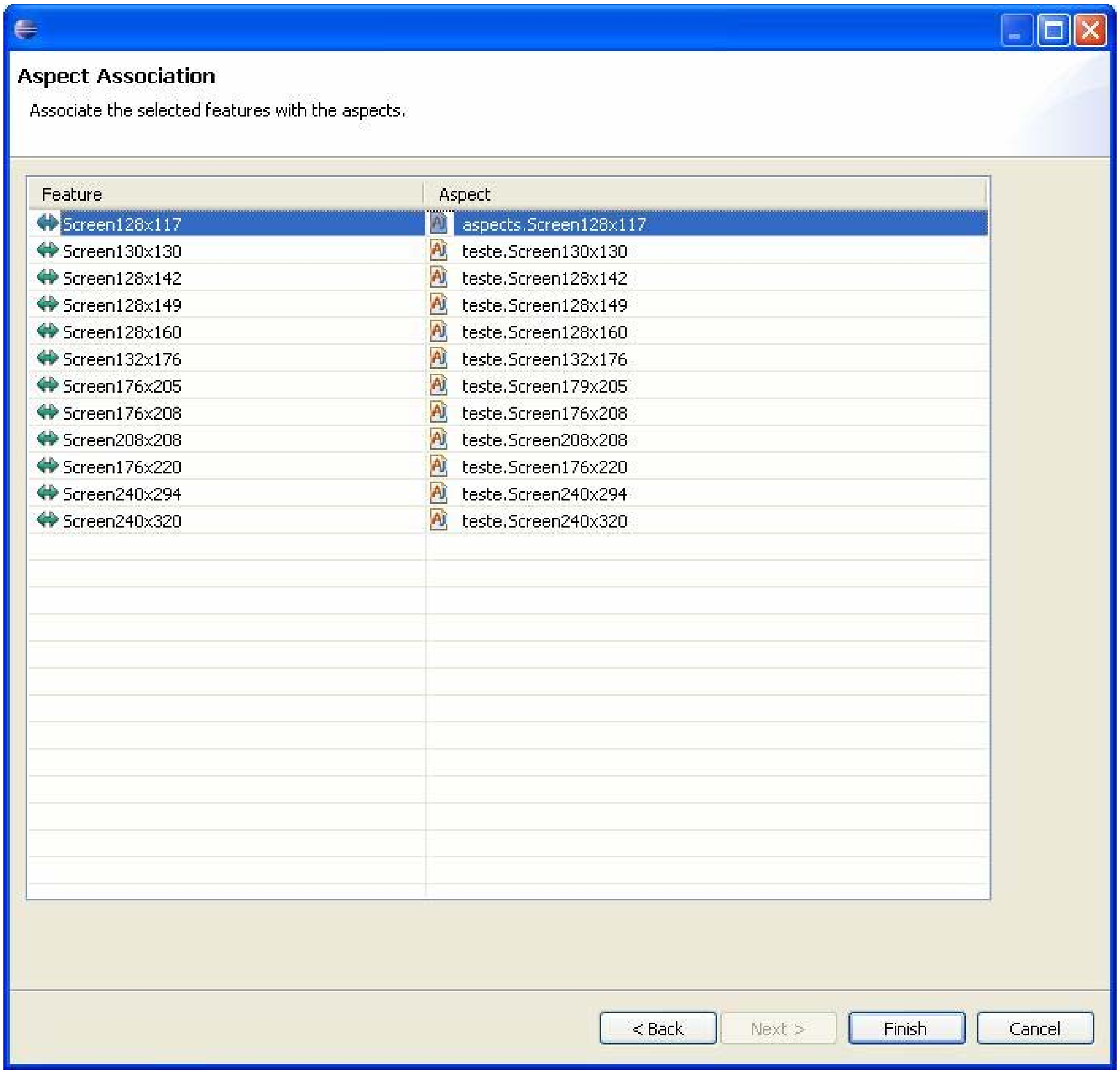
```
    public static final int SCREEN_WIDTH  = 128;
    public static final int SCREEN_HEIGHT = 117;
```



Feature Association

Associate the selected tags with the features.

Tag	Feature
device_screen_128x117	Screen128x117
device_screen_130x130	Screen130x130
device_screen_128x142	Screen128x142
device_screen_128x149	Screen128x149
device_screen_128x160	Screen128x160
device_screen_132x176	Screen132x176
device_screen_176x205	Screen176x205
device_screen_176x208	Screen176x208
device_screen_208x208	Screen208x208
device_screen_176x220	Screen176x220
device_screen_240x294	Screen240x294
device_screen_240x320	Screen240x320



Aspect Association

Associate the selected features with the aspects.

Feature	Aspect
Screen128x117	aspects.Screen128x117
Screen130x130	teste.Screen130x130
Screen128x142	teste.Screen128x142
Screen128x149	teste.Screen128x149
Screen128x160	teste.Screen128x160
Screen132x176	teste.Screen132x176
Screen176x205	teste.Screen179x205
Screen176x208	teste.Screen176x208
Screen208x208	teste.Screen208x208
Screen176x220	teste.Screen176x220
Screen240x294	teste.Screen240x294
Screen240x320	teste.Screen240x320