Automated Generation of Platform-Variant Applications from Platform-Independent Models via Templates

Nuno Amálio, Christian Glodt, Frederico Pinto, Pierre Kelsen

University of Luxembourg

Introduction

* Manual code development is expensive
* Platform-diversity: need to support various platforms
* Not easy to focus on design within code-based development

Talented people may be inspired to create novel applications, but only few have the time, energy and technical skill to dig into the intricacies of low-level programming...

Model-Driven Development

* Raise level of abstraction
  * from code to models
* Sw development focused on design
* Reduce reliance on code experts

Model-Centric approach

* All platform-code generated from functional models (PIMs)
* Models are total: structure+behaviour
* Use of abstraction enables coverage of various platforms
* Early experimentation from models

Our Approach

* Product families described as PIMs
* Platform-specific artifacts generated from templates
* A catalogue of templates per platform

Our Languages

* To describe PIMS:
  * visual language EP+OCL
* To describe Templates: FTL
* Both languages have formal semantics

EP

* EP structures models around classes
* Classes comprise events and properties
  * Details of events described in OCL
* EP models divided into domains
  * Domains represent different subject matters
* Domains are linked using bridges

FTL (1)

* Expresses templates of any target language
* Generation is based on substitutions
* We used a java implementation of FTL theory (integrated in Democles)

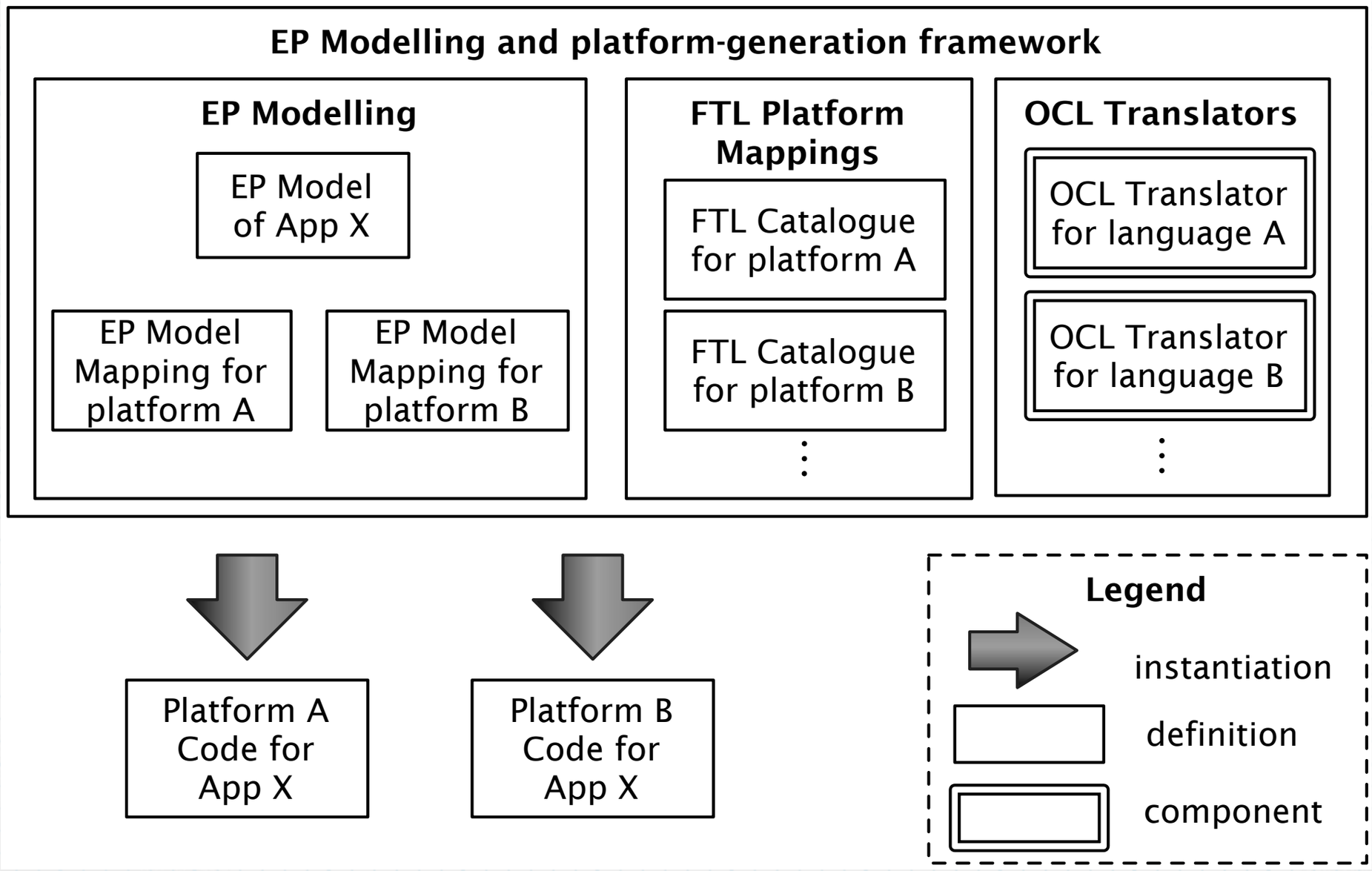
FTL (II)

```
module JavaClassCat
Class ==
  public class 《ClName》{
      ⟦private 《propName》 : 《propTy》;⟧
      ⟦ClassMethod⟧
}■
```

FTL (III)

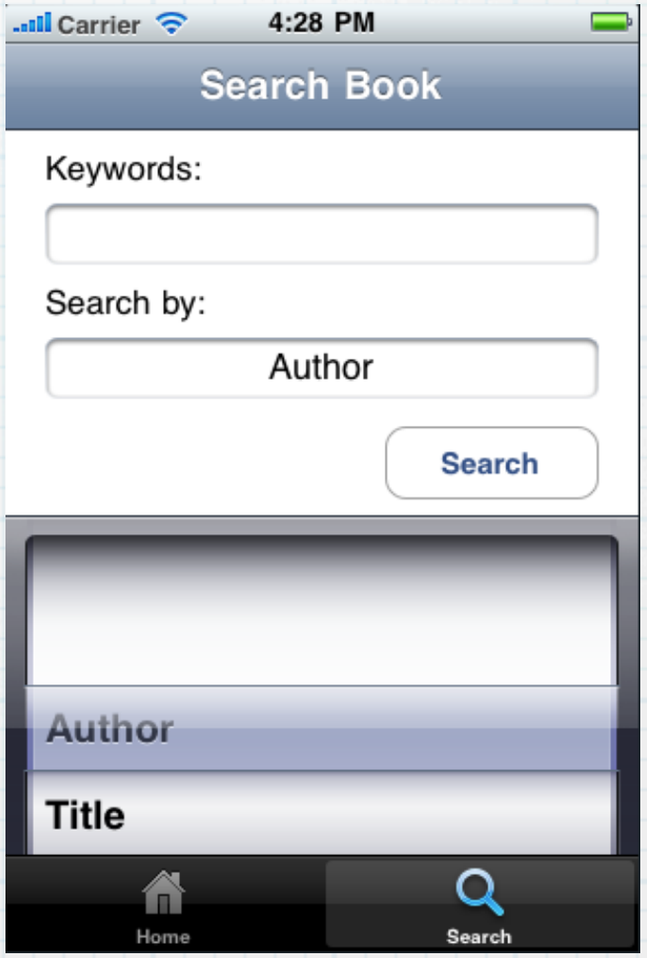
```
public class Book {
   private title : String;
   private isbn : String;
}
```

Our Approach



Running Example

Simple Library Browser



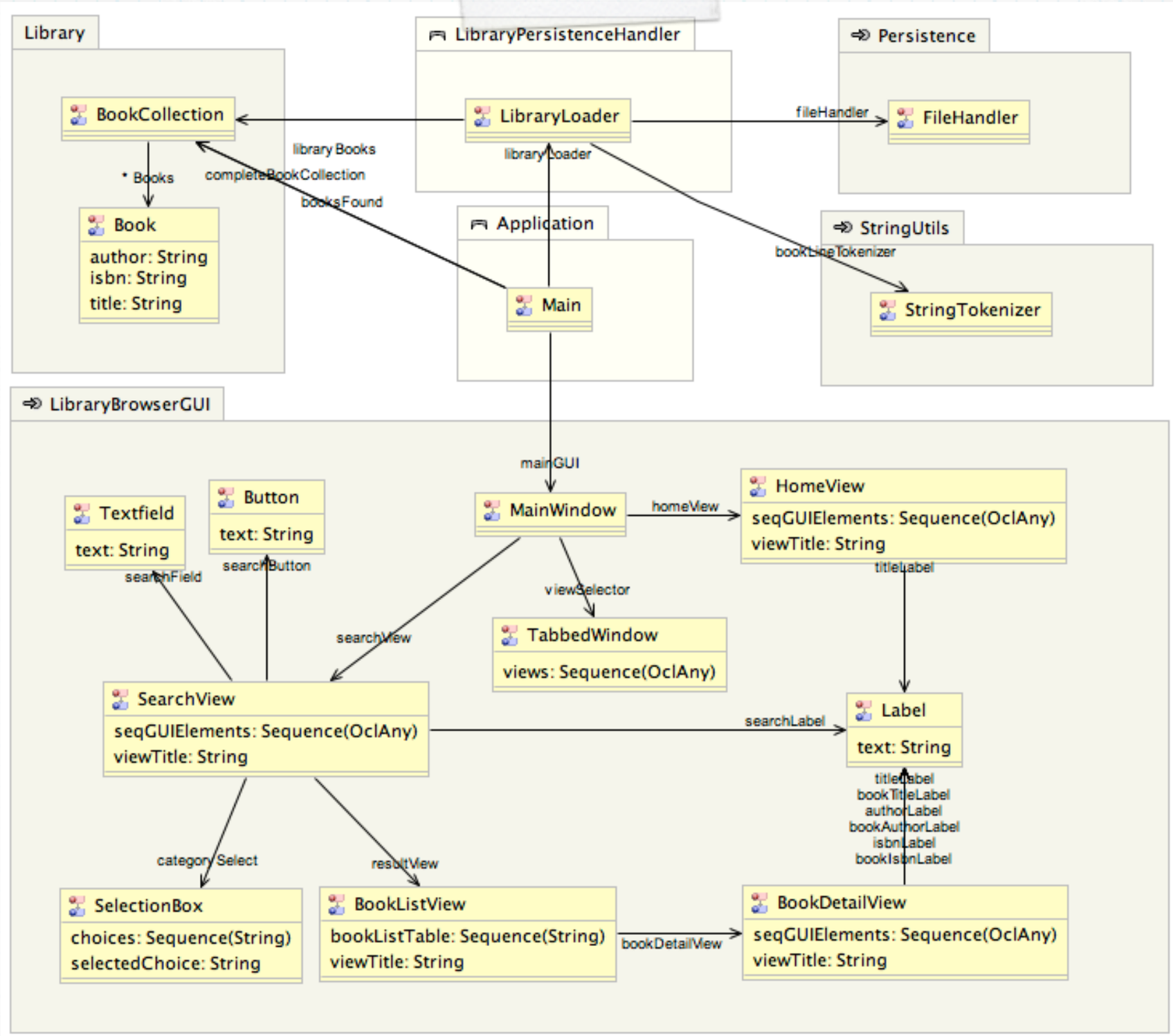
FTL Template Catalogues

// Catalogue of FTL templates
// for the Android platform
module AndroidCat

Templates catalogue for the Android

```
import CoreJava
import AndroidGUI
import AndroidPersistence
import JavaStringUtils
import AndroidSupport
```

EP Model


Library
BookCollection
* Books
Book
author: String
isbn: String
title: String
LibraryPersistenceHandler
LibraryLoader
libraryBooks
completeBookCollection
booksFound
libraryLoader
Persistence
fileHandler
FileHandler
Application
Main
StringUtils
bookLineTokenizer
StringTokenizer
LibraryBrowserGUI
mainGUI
MainWindow
homeView
HomeView
seqGUIElements: Sequence(OclAny)
viewTitle: String
titleLabel
Textfield
text: String
searchField
Button
text: String
searchButton
viewSelector
TabbedWindow
views: Sequence(OclAny)
searchView
SearchView
seqGUIElements: Sequence(OclAny)
viewTitle: String
searchLabel
Label
text: String
titleLabel
bookTitleLabel
authorLabel
bookAuthorLabel
isbnLabel
bookIsbnLabel
categorySelect
resultView
SelectionBox
choices: Sequence(String)
selectedChoice: String
BookListView
bookListTable: Sequence(String)
viewTitle: String
bookDetailView
BookDetailView
seqGUIElements: Sequence(OclAny)
viewTitle: String

Demonstration...

Evaluation (I)

* From same EP PIM: generated code for iPhone and Android

* Platform application described in terms of abstract concepts

* Could be developed by non-platform experts

	iPhone	Android
FTL Catalogue	1745	1661
Generated Code	7119	5702

Conclusions (I)

* EP Model is platform-independent
* Platform-specificity and variability hidden in templates catalogue
* Not everything describable in OCL
* Mapping EP Domain define generic events

Conclusions (II)

* FTL templates provide separate medium over which experts can express their knowledge
  * Use of EP models benefits usability. Templates not so easy to develop
  * Platform catalogues are reusable assets
    * living repository of knowledge

Conclusions (III)

* Use of EP enables formal model analysis
  * event propagation and animation
  * Experimentation of design decisions using generated code

Future Work

* Apply approach to larger case study
* Verification of EP models