Component-Based Product Line Engineering

Software Product Line Basics

The Product Line Practice (PLP),
Carnegie Mellon Software Engineering Institute, Pittsburgh, PA, USA

http://www.sei.cmu.edu/plp/

Component-Based Product Line Development: The KobrA Approach

Colin Atkinson, Joachim Bayer, Dirk Muthig
Fraunhofer Institut Experimentalles Software Engineering,
Kaiserslautern, Germany

http://www.iese.fhg.de/KobrA/
Product Line Basics - Key Concepts

Product Line (PL) Definition

A group of products sharing a common, managed set of features that satisfy specific needs of a selected market or mission [Clements 99]

- Product lines → Architecture-based development within application domain
- Software product lines provide systematic reuse of the core assets
Product Line Practice (PLP) Definition

The systematic use of software assets to modify, assemble, instantiate, or generate the multiple products that constitute a product line.
Product Line Basics - Product Line Acquisition

Terminology

Core asset development/acquisition → Domain Engineering

Product development/acquisition → Application Engineering
(from core assets)

Product Line Acquisition
The process of obtaining products and services through contract

Contract
Binding agreement between two or more parties that establishes the requirements for the products and services to be acquired

Acquisition ⇔ Contracting
Product Line Basics - Acquisition Strategy

Acquisition Strategy

A plan of action for achieving a specific goal or result through contracting for products and services.

Core Assets Acquisition Strategy

- to develop a software architecture, a production plan, other core assets
- mine legacy assets to extract core assets
- manage, upgrade, and enhance the asset base and support product developers
- purchase or license commercial off-the-shelf (COTS) components

Product Acquisition Strategy

- to develop set of products from core assets according to the production plan
- maintain, upgrade, or enhance a product or set of products
- provide new assets for evaluation as candidate assets
Product Line Basics - PL Acquisition Program

Software Assets

System 1  
System 2
System 3
System 4
Domain Engineering

⇒ covers all the activities for building software core assets within Application Domain (AD)

• identifying one or more domains
  
  *(Domain Analysis)*

• capturing the variation within a domain

  *(Domain Design)*

• defining the mechanisms for translating requirements into systems created from reusable components

  *(Domain Implementation)*
Application Engineering (AE)

⇒ develops software products from software assets created by a domain engineering process

Typical AE activities include using:

• domain model
• generic design (design model)
• partitioning strategy and coordination model (Architecture Style)
• application generators
• and software components
Product Line Basics - Observations

Observations about Concepts and Terminology

- Product line $\neq$ group of products produced by a single business unit
- PLP $\neq$ the same as single-system development with reuse
- PLP $\neq$ another name for domain engineering
- PLP $\neq$ another name for component-based development
- PLP does not depend on the size of particular corporation
- PLP $\Rightarrow$ unbounded variability $\neq$ loss of flexibility
- Process Improvement $\subset$ Product Line Practice
- Products share either the same architecture or slight variations of it
Important Concepts and Terminology

- Ad hoc reengineering ↔ Preplanned reusable software core
- Components → “reuse in the small” ↔ PL → “reuse in the large”
- Product Lines approach is not a “panacea”
- Software System (“everything from wires to GUI”) = Product within AD
Product Line Basics - Differences

Product Lines ≠ multiple releases of the same product

<table>
<thead>
<tr>
<th>Area</th>
<th>Multiple releases</th>
<th>Product Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>added incrementaly</td>
<td>usually are not a subset of another product</td>
</tr>
<tr>
<td>Architecture</td>
<td>usually identical</td>
<td>instance of PL has its specific variations</td>
</tr>
<tr>
<td>Compatibility</td>
<td>backward compatibility</td>
<td>no compatibility (why ?)</td>
</tr>
<tr>
<td>Sequence</td>
<td>sequentially</td>
<td>independently</td>
</tr>
<tr>
<td>Organization structure</td>
<td>one team for each release</td>
<td>structured organization</td>
</tr>
<tr>
<td>Lifetime</td>
<td>old release → new release</td>
<td>indefinite lifetime</td>
</tr>
<tr>
<td>Qualities</td>
<td>same qualities</td>
<td>each product has different qualities</td>
</tr>
<tr>
<td>User Interface</td>
<td>stays the same</td>
<td>each product → different user interface</td>
</tr>
</tbody>
</table>
The KobrA Approach - Introduction

Concepts

- KobrA → synergy between the component-based and PL approaches
- Components: construction elements for the application of KobrA PL Development
- Components → flexible and rapid configurability needed for PL development
- PLP → methodological foundation for the component-based frameworks
- “hard wiring” of the frameworks → “plugging in” of the components

PuLSE (Product Line Software Engineering)

- provides technical components for the different framework deployment phases
- KobrA → “ready-to-use” customization of PuLSE
The KobrA Approach - The KobrA Concepts

The KobrA Approach Concepts

- **Processes**: create products ↔ **Products**: describe components
- **Products**: models, documents, code modules, test cases, etc.
- **Components**: can be separated from the environment = reused independently
- **Framework Engineering Activities** ↔ **Application Engineering Activities**

Result of both Engineering Activities:

Mixture of textual and
UML-based models
Framework (“Instance of Domain”) Engineering Concepts

- Framework = reuse infrastructure for creating software systems within AD
- KobrA frameworks = Static tree of Components
- Component: Specification ↔ Realization (tightly coupled arrangement)

Framework Development Process

- Context Realization = Elicitation of the environment properties and determining the scope of the framework (from root component in top-down manner)

- Output: Context Realization Model (CRM) - (Class diagrams, Decision Models)
- CRM : Specification Models (SM) → Realization Models (RM) = (refinement)
- RM of “upper” component → SM of “lower” component (recursion)
The KobrA Approach - Components’ UML Models
The KobrA Approach - Context Class Diagram

Library System Example - Context Class Diagram

```
Figure 5. Context Class Diagram for Library Systems
```
The KobrA Approach - Library Business Processes

Library System Example - Business Processes

![Diagram of library business processes](image-url)
### The KobrA Approach - Decision Model

#### Library System Example - Decision Model

<table>
<thead>
<tr>
<th>Id</th>
<th>Customer Interaction</th>
<th>Resolution</th>
<th>Diagram</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Does the loan of items cost anything?</td>
<td>y</td>
<td>Activity Diagram</td>
<td>Activity „Loan Cost Determination“ is present</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>„Item Check In“</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>n</td>
<td>Activity Diagram</td>
<td>Activities „Loan Cost Determination“ and „Loan Cost Collection“ are not present.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>„Item Check In“</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>What is the type of payment?</td>
<td>per item</td>
<td>Activity Diagram</td>
<td>Activity „Loan Cost Collection“ is present</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>„Item Check In“</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>membership fee</td>
<td>Activity Diagram</td>
<td>Activity „Loan Cost Collection“ is not present.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>„Item Check In“</td>
<td></td>
</tr>
</tbody>
</table>
The KobrA Approach - Specification Class Diagram

Library System Example - Specification Class Diagram

```
LoanManager
Account currentAccount
startCustomerInteraction()
endCustomerInteraction()
getAccount(userData)
startCheckIn()
endCheckIn()
startCheckOut()
endCheckOut()
startReservation() (opt.)
endReservation() (opt.)

Customer
name
personalData

Account
id

Item
id
itemData

LoanInfo
checkOutDate
returnDate (opt.)

0..1
```
The KobrA Approach - Statechart Diagram

Library System Example - Statechart Diagram
The KobrA Approach - Operation Schema

Library System Example - Operation Schema \((\text{startCheckIn})\)

<table>
<thead>
<tr>
<th>Operation</th>
<th>startCheckIn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>All returned items are identified, removed from the account, registered as present in the stock,</td>
</tr>
<tr>
<td></td>
<td>\begin{center}loan costs are determined, {and collected}.\end{center}</td>
</tr>
<tr>
<td>Reads</td>
<td>supplied itemData of returned items, currentAccount</td>
</tr>
<tr>
<td>Changes</td>
<td>currentAccount</td>
</tr>
<tr>
<td>Sends</td>
<td>ItemManager.return(list of returned Items)</td>
</tr>
<tr>
<td>Assumes</td>
<td>customer interaction is open but no transaction is active.</td>
</tr>
<tr>
<td>Result</td>
<td>Returned items that were registered for currentAccount are back in stock and have been removed from currentAccount.</td>
</tr>
<tr>
<td></td>
<td>\begin{center}Loan costs have been determined {and collected}.\end{center}</td>
</tr>
</tbody>
</table>
The KobrA Approach - Collaboration Diagram

Library System Ex. - Collaboration Diagram (startCheckIn)
The KobrA Approach - Realization Class Diagram

Library System - Realization Class Diagram (LoanManager)
Application Engineering Concepts

- Constructing specific applications in the domain covered by the framework
- Tightly coupled with the models developed during framework engineering
- Driven by the framework’s decision models (*traversing in a top-down manner*)
- Framework models $\rightarrow$ Specific models

Reasons for Specific models (Product Instances)

- Exactly corresponding models due to maintenance of particular application
- Specific user requirements (*e.g.*, *too many conflicting constraints*)
The KobrA Approach - App. Context Realization

Application Context Realization (ACR) Concepts

**Start:** Establishing an initial contact with potential customer

**Stop:** Customer accepts the realization of the application context

**Output:**
1) Context decisions (*choices made by the customer*)
2) Requirements for the application to be developed (*a set of models*)

Requirements elicitation (driven by decision sequence)
- a set of possible alternatives (*no change in system → framework = template*)
- customer-specific requirements (*changes in application creation lifecycle*)

Customer-specific requirements problems
(a) conflicts with other requirements  (b) realization problems in the framework
The KobrA Approach - Framework Instantiation

Framework Instantiation Concepts

**Start:** ACR is (partially) created = context decisions (partially) exist

**Stop:** When all decisions are resolved, modeled and integrated into Application

**Output:**
1) Application realization (*choices made by the designers*)
2) Application tree (*a hierarchy of components*)

Instantiation of the component hierarchy = connection of:

- Application Context Realization Decision Models
- Decision Models at the Framework’s Component Realization level

Frequent Customer-specifics → Framework-specifics
The KobrA Approach - AE Process Overview

Overview of the Application Engineering Process
Product Line Engineering - References

Publications related to the Product Lines Engineering:

Product Line Practice Initiative (*product line community*):

http://www.sei.cmu.edu/plp/

Fraunhofer Institute of Experimental Software Engineering:

http://www.iese.fhg.de

Interesting PL approach from the developer’s perspective:

http://www.cs.utexas.edu/users/schwartz/pub.htm

Aspect Oriented Programming (AOP) - homepage: