

Object-Oriented Application Frameworks Part 2

Greg Butler

Computer Science, Concordia University, Montreal

gregb@cs.concordia.ca

Development of Frameworks — Issues

Top-down vs Bottom-up

Commonality vs Variability

Identifying “Hotspots”

Classifying flexibility at hotspots

Narrow specialization interfaces

Refactoring

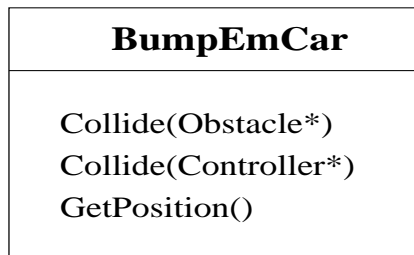
Sidebar on Terminology

Hotspot: a place where flexibility is required.

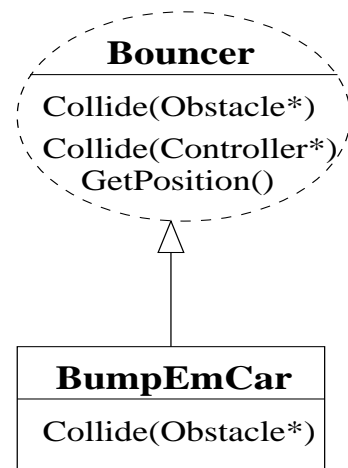
Refactoring: a structural change to a program that does not alter its behaviour.

Usually used in the context of re-distributing responsibility in a class hierarchy.

BEFORE



AFTER



Development of Frameworks — Approaches

Top-down

All applications at once.

Step 1: Domain analysis

Step 2: Develop domain specific software architecture (DSSA)

Step 3: Implement DSSA

Step 4: Populate DSSA as applications required

Bottom-up

One application at a time.

Step 1: Build first application

Step 2: Iterate

1. Change impact analysis for n -th application
2. Refactor existing framework to accommodate changes
3. Build n -th application

Hybrid top-down and bottom-up

Two-three applications initially.

Interleave

1. Partial domain analysis
2. Change impact analysis and refactoring
3. Build n -th application

Documentation of Frameworks — Issues

Kind of Reuse (Kind of Audience)

- **Selecting:** for the intended application
- **Composing:** by selecting from library of concrete subclasses
- **Extending:** hotspot in ways planned for
- **Flexing:** hotspot in a way not planned for
- **Evolving:** to add new hotspots, or new flexibility
- **Mining:** for ideas applicable in other contexts

Learning Curve is too steep

special case of program understanding problem

but worse, since

- design is very *abstract*
to factor out commonality
- design is *incomplete*
needs extra subclasses to create an application
- design provides *(too much) flexibility*
not all needed in the application at hand
- collaborations and dependencies can be *indirect* and *obscure*

Documentation of Frameworks — Approaches

source code of framework

source code of **example applications**

framework overview states domain and scope

cookbook of **recipes** describes typical customizations

(same as *pattern language of patterns*)

also *active cookbooks* as CASE tools

reference manual describing

- purpose of each class
- *interface* of each class
- *specialization interface* of each class
- constraints on customization

design pattern describing flexibility of hotspot

behaviour specification specifying

- *interface contract* of each class
- *interaction contract* of each collaboration

architecture, often as collection of *design patterns*

Development with Frameworks — Issues

Learning the Framework

Planned Customization is fairly routine

Unplanned Customization and Evolution

Development with Frameworks — Approaches

Follow recipes for planned customization

Architecture-based development and evolution

not much written

see ASE in Jacobson&Griss&Jonsson

Framework Development

Classical Bottom-Up Iteration (Johnson, et al)

Stages of framework maturity

- white-box framework
- black-box framework
- component library
- pluggable objects
- visual builder
- language tools

Hotspot Generalization (Schmid, Pree)

Identify hotspots

Classify variability required for each hotspot
choose from *meta-patterns*

Associate a subsystem with each hotspot
select a design pattern for subsystem
subsystem generalizes abstract class/facade

Variation by Demeyer&Meijler&Nierstrasz&Steyaert:
identify axes of variability for each hotspot
introduce one abstract class per axis

Framework Development

Top-Down, End-to-End (Jacobson&Griss&Jonsson)

Component-based frameworks in context of business process engineering

- Application Family Engineering (AFE) — design framework as layered architecture of components
- Component System Engineering (CSE) — design flexibility into each component
- Application System Engineering (ASE) — development with reuse of framework

Overview of CSE:

Capture functionality as use cases

Identify *variation points* in use cases

based on feature-oriented domain analysis (FODA)

Robustness analysis (OOSE) to maximize flexibility

Map to design patterns