The Koala Component Model for Consumer Electronics Software
by: Ommering, Linden, Kramer, Magee

Presented by: Bridget Flaherty

Overview

• Koala
  – Motivation
  – The Koala Model
  – Handling Diversity
  – Coping with Evolution

• Software Component Model Lifecycle
• Opportunities for Aspect-Orientation
• Discussion
Consumer Electronics Software

• Challenge:
  + Increase in size and complexity of software
  + Increase in diversity of products and therefore the diversity of the software
  - Decrease in development time

A Solution to the Challenge

• Component-based approaches
  – Component
    • encapsulated piece of software
    • has an explicit interface to its environment
    • can be used in many configurations
  – The component based approach allows for multiple configurations with variation in both structure and content
The Perfect Marriage
Components + Architectural Description Language

• Components without an ADL
  – Difficult to manage components
  – Difficult to maintain consistency between original
    component designs and their code
    (limitations with reverse engineering)

• Architectural Description Language:
  – Is an explicit description of configuration structure
  – Improves component management by helping users
    visualize:
    • Diversity of product family
    • Complexity of individual products

CE Domain Requirements

• Static Binding is Preferable
  – Limits runtime overhead
  – Possible because component connections are
    constant and known at configuration time

• Dynamically Bound components for
  upgrading high end products

• “requires” interfaces necessary
  – Components require a certain interface
    without having explicit binding knowledge
Darwin Evolved = Koala

- Darwin:
  - An ADL originally developed for distributed systems
- Like Darwin it offers:
  - Explicit hierarchical structure
  - Components have *requires* and *provides* interfaces
  - Bindings
- In addition to Darwin, Koala has:
  - Glue code between components
  - Diversity parameter mechanism
    - Parameters are defined
    - Code is optimized based on parameters

**THE KOALA MODEL**

Separation of component development and configuration development
Koala Graphical Notation

HANDLING DIVERSITY
Koala’s Features for Efficiently Handling Product Diversity
Diversity Features

1. Interface Compatibility:
   • Interfaces cannot be changed to fit only 1 implementation
   • Interfaces must provide at least all the required functions

2. Function Binding:
   • Used to bind functions with different names without adding runtime overhead
   • Koala actually generates a macro based on developed C code

3. Partial Evaluation:
   • Koala understands subset of C

4. Diversity Interfaces:
   • Components have no configuration-specific information
   • Instead it requires properties through the standard interface mechanism

5. Diversity Spreadsheets:
   • Object oriented spreadsheet of diversity parameters

6. Switches:
   • Function binding with condition expressions to route function calls to desired component
Diversity Features

7. Optional Interfaces:
   • Providing extra interfaces through an *optional requires* interface

8. Connected Interfaces:
   • Determine if components are connected and save time-consuming initialization

COPING WITH EVOLUTION

Koala’s Features
Coping with Evolution

1. Interface Repository:
   - where interface definitions are stored
   - Evolution constraining rules:
     1. existing interface types cannot be changed*
     2. new interface types can be added

2. Component Repository:
   - where components are stored
   - Evolution constraining rules:
     1. new components can be added
     2. provides interfaces can be added but not deleted from existing components
     3. new requires interfaces must be optional, existing requires interfaces can be converted to optional

3. Configuration Management
   - manages component history

SOFTWARE COMPONENT MODEL LIFECYCLE
Generic Component Lifecycle

• Design Phase:
  – Components are constructed, catalogued and stored in a repository
  – Components can be composed into a composite component and stored in repository

• Deployment:
  – Components are retrieved and compiled to binary code

• Run-Time Phase:
  – Components are instantiated with data and executed

Koala Component Lifecycle

• Design Phase = Generic Design Phase

• Deployment Phase:
  – No new composition is possible in the deployment phase
  – Components cannot be changed outside of the design phase
Evolving Embedded Product Lines: Opportunities for Aspects

Product Line Evolution

- Evolving to incorporate new features
- Increased code and complexity within the product line
- New features have an increasingly more crosscutting nature
Crosscutting Issues

- Components provide separation of functionalities, but entangle multiple features
- These features may not be captured in Architectural Description Language
- Adding a new feature with a crosscutting nature will cause integration problems

Opportunities for Aspect-Orientation

- Requirements
  - Aspect-oriented requirements engineering (AORE)
    - Area of research to modularize and reason about crosscutting concerns during requirements
- Architectural level
  - Aspects can help with:
    - Crosscutting in an ADL
- Code level
  - Encapsulate the code of new features into aspects and automatically weave them into several components
DISCUSSION

Versatility of Koala?

• No documented cases of Koala being applied to more domains
• Certainly a studied case of an applied software product line model for embedded systems
• Suggestions that it could be extended to be more “CORBA-like” to be used in distributed systems