Designing Evolutionary Architecture-centric Component-based Software Product Lines

Hassan Gomaa  
Dept. of Computer Science  
George Mason University  
Fairfax, Virginia, USA  
hgomaa@gmu.edu

Third International Conference on Software Engineering Advances (ICSEA 2008),  
Keynote Presentation  
October 27, 2008

Overview

- Software Modeling and Design  
- Software Modeling and Design Methods  
- Designing Evolutionary Systems and Product Lines  
- Designing and Evolving Systems from Architectural Design Patterns  
- Executable Models of Software Designs  
- Dynamic Software Reconfiguration
Software Modeling and Design

- Origins of Modeling
  - Small scale plans in art and architecture
- Modeling in science and engineering
  - Abstraction of system at some level of precision and detail
  - Analyze model to get better understanding of system
- Computer-Aided Design (CAD)
  - Computer models of product before product manufacturing
- Software Modeling
  - OMG: Modeling is designing of software applications before coding

Overview

- Software Modeling and Design
- **Software Modeling and Design Methods**
- Designing Evolutionary Systems and Product Lines
- Designing and Evolving Systems from Architectural Design Patterns
- Executable Models of Software Designs
- Dynamic Software Reconfiguration
RT Software Design Methods
1984 - 1993

• 1984 – Support for concurrent task structuring
  – Design Approach for Real-Time Systems (DARTS)
• 1986 - Support for distributed applications
  – DARTS/DA
• 1989 - Support for real-time Ada design
  – ADARTS
• 1993 - Support for concurrent, real-time, & distributed OO design
  – CODARTS

UML and COMET

• Unified Modeling Language (UML)
  – OMG Standardized notation for describing design
  – Methodology independent
• Concurrent Object Modeling and architectural design mETHod (COMET)
  – Object Oriented Analysis and Design Method
  – Targeted for concurrent, distributed, and real-time applications
  – Uses UML notation
• COMET = UML + Method
• H. Gomaa, “Designing Concurrent, Distributed, and Real-Time Applications with UML”, Addison Wesley Object Technology Series, 2000
Designing Software Product Lines with UML

- **Software Product Line (SPL)**
  - Family of products / systems (Parnas, Weiss, SEI)
- **OO Analysis and Design for Product Lines (PLUS)**
  - Extends COMET, other methods for single systems
  - Model commonality and variability among members of software product line
- Apply standard UML extension mechanisms
  - Stereotypes, constraints, tagged values
- **UML 2.0**
  - Notation for depicting software architectures and components

Evolutionary Process Model for Software Product Lines
Overview

- Software Modeling and Design
- Software Modeling and Design Methods
  - Designing Evolutionary Systems and Product Lines
  - Designing and Evolving Systems from Architectural Design Patterns
- Executable Models of Software Designs
- Dynamic Software Reconfiguration
Designing Evolutionary Systems and Product Lines

- Software Product Line
  - Model commonality and variability
  - Feature modeling used to explicitly capture variability
- Evolutionary System
  - Software system evolves from version to version
  - Can model as software product line
  - Each version of system is member of SPL
  - Model different features as system evolves
- Evolutionary Software Design
  - Evolution built into design method
  - Architecture-Centric Evolution

Evolutionary Architecture-Centric Development for Systems and SPL

- Kernel First Approach
  - Develop initial version of system or kernel of SPL
  - Kernel is similar to single system
- Evolutionary Development Approach
  - Consider evolution as an iteration in software development
- Model-based evolution
  - Feature-based Impact Analysis
  - Consider impact of optional and alternative features on kernel
  - Emphasis on dynamic modeling
Requirements Modeling

What should Evolutionary Design Method provide?

- For evolutionary systems and product lines
  - Support variability and evolution in use case modeling
  - Integrate feature modeling with other UML views

Use Case Modeling for Evolutionary Systems and Product Lines

- Categorize use cases using UML stereotypes
- Model commonality
  - «kernel» use cases
- Model variability
  - «optional» use cases
  - «alternative» use cases
  - Model variation points in use cases
    - Specify variability within use case
- Model use case evolution
  - Additional optional and alternative use cases
  - Additional variation points in existing use cases
Kernel Use Case Model for Microwave Oven SPL

Categorize use cases using UML Stereotypes
- Kernel use case

Evolution of Use Case Model: Microwave Oven SPL
- Add optional use cases
- Add variation points
  - E.g., optional Light,
  - Alternative One-line or Multi-line Display
Feature Modeling for Evolutionary Systems and Product Lines

- Feature (Kang, SEI)
  - Function or characteristic that differentiates between members of the software product line
- Feature modeling
  - PLUS integrates feature modeling with other UML modeling views
- Feature modeling in PLUS
  - Determine features from use cases and variation points
  - Concentrate on modeling variability
- Evolutionary Systems and Product Lines
  - Feature modeling guides evolution

Feature Modeling with UML

- Use static modeling meta-class notation
  - Meta-classes depict features and feature groups
- Features are categorized using UML stereotypes
  - «common feature»
  - «optional feature»
  - «alternative feature»
  - «default feature»
  - «parameterized feature»
- Model Feature Dependencies
  - Requires
  - Mutually includes
Features and Feature Groups in UML

Feature Relationships: Constraint on using features in group
E.g., «exactly-one-of feature group»,
«zero-or-one-of feature group»

Analysis Modeling
What should Evolutionary Design Method provide?
• For evolutionary systems and product lines
  - Dynamic modeling is very important
  - Evolution of interaction models
  - Evolution of state machines
Dynamic Interaction Modeling
Feature Based Impact Analysis

- Kernel First Approach
  - Develop kernel interaction diagrams to realize kernel use cases
  - Realized by kernel and default objects
- Product Line evolution approach
  - Consider impact of optional and alternative features on SPL kernel objects
  - Results in optional and variant objects
- Example of Kernel First Approach
  - Microwave Oven SPL
  - Kernel use case: Cook Food
- Product Line Evolution – impact analysis

Copyright 2008 H. Gomaa

Kernel Communication diagram for Cook Food use case
Impact Analysis of Beeper and Light Features

• 2 optional objects added
• Impact on control object
  • Feature dependent messages

Evolution of State Machines
Feature Based Impact Analysis

• Kernel First Approach
  – Develop kernel state machines for state dependent control objects
• Product Line evolution approach
  – Consider impact of optional and alternative features on kernel state machines
• Feature dependent state transition
  – Use feature condition as guard on state transition
    • Event [Feature Condition]
• Feature dependent action
  – Action is only executed if Feature Condition is True
  – Action [Feature Condition]
Kernel statechart for Microwave Oven Control

- Incoming message to object -> input event on statechart
- Output event on statechart -> outgoing message from object

Impact on Statechart for Microwave Oven Control

Feature dependent actions:
- Action is only executed if Feature Condition is TRUE
- Action [Feature Condition] e.g., [light], [beeper]
Evolution of State Machine Models

- Model state machine variability
  - Inherited vs Parameterized State Machines
- Inherited State Machine
  - Different state machine for each alternative or optional feature
- Disadvantage:
  - Each feature & feature combination needs an inherited state machine
  - Could lead to combinatorial explosion of inherited state machines
- Often better to design parameterized state machine
  - Feature dependent state transitions and actions

Design Modeling
What should Evolutionary Design Method provide?

- Software Architectural Patterns
- Evolutionary Component-Based Software Architectures
Component-based Distributed Software Architecture

- Distributed component
  - Logical unit of distribution and deployment
  - Well-defined provided and required interfaces
- Modeling Components in UML 2.0
  - Components modeled as UML 2.0 structured classes
  - Depicted on UML 2.0 composite structure diagrams
  - Provides support for
    - Composite and simple components
    - Interfaces, ports, connectors
- PLUS component categorization using stereotypes
  - Application role category
  - Reuse category

Software Architecture for Microwave Oven – Before Evolution


Copyright 2008 H. Gomaa
Design of “Plug Compatible” Components

- Connector
  - Joins **required** port of one component to **provided** port of another component
- “Plug compatible” components
  - One interface
  - Realized by different components
- E.g., Microwave Control can be connected to one-line or multi-line version of Microwave Display
- If interface needs to be extended
  - Use component inheritance

Design of individual components

<table>
<thead>
<tr>
<th>«interface»</th>
<th>IDisplay</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayPrompt (in promptId)</td>
<td></td>
</tr>
<tr>
<td>displayTime (in time)</td>
<td></td>
</tr>
<tr>
<td>clearScreen ()</td>
<td></td>
</tr>
<tr>
<td>displayTOD (in TOD) [feature = TODClock]</td>
<td></td>
</tr>
</tbody>
</table>

Key
- ◯ port
- □ provided interface
- ▲ required interface
- ◇ concurrent component

Copyright 2008 H. Gomaa
Design of Optional Components

- Feature dependent interfaces & components

```
interface IBeeper
    {feature = Beeper}
initialize ()
beep ()
```

```
optional output component
BeeperComponent
```

```
interface ILamp
    {feature = Lamp}
initialize ()
switchOn ()
switchOff ()
```

```
optional output component
LampComponent
```

Evolution of Microwave Oven Architecture

- Feature dependent components, connectors and messages
- Messages correspond to feature dependent actions on state machine

- Microwave Control
- Feature dependent actions on state machine: Switch On[light], Beep [beeper]
Evolution of Microwave Oven Architecture

- Feature dependent components, connectors and messages
- Messages correspond to feature dependent actions on state machine

Overview

- Software Modeling and Design
- Software Modeling and Design Methods
- Designing Evolutionary Systems and Product Lines
- Designing and Evolving Systems from Architectural Design Patterns
- Executable Models of Software Designs
- Dynamic Software Reconfiguration
Software Architectural Patterns

- Software Architectural Patterns [Buschmann, Shaw]
  - Recurring architectures used in various software applications
- Goal: Design Software Architecture from
  - Software Architectural Patterns
- Architectural Structure Patterns
  - Address structure of major subsystems
- Architectural Communication Patterns
  - Reusable interaction sequences between components

Architectural Structure Patterns for Software Product Lines

- Layered patterns very important for evolution
  - Layers of Abstraction
  - Kernel
- Client/Server patterns
  - Basic Client/Server
  - Client/Broker/Server
  - Client/Agent/Server
- Control Patterns very important in RT Design
  - Centralized Control
  - Distributed Control
  - Hierarchical Control
Layers of Abstraction Pattern

- Structure system into hierarchical layers [Parnas]
  - Each layer provides services for higher layers
- Layers of Abstraction in Product Lines
  - Allows design of variable and extensible software
  - Kernel components at lowest layer
  - Optional and variant components at higher layers
- Software Evolution
  - Add components at higher layers
  - Depend on services provided at lower layers

Flexible Layers of Abstraction Pattern in Factory Automation SPL
Centralized Control Pattern
E.g., Microwave Oven Control Architecture

- Centralized Control Pattern
  - One control component
    - Executes statechart
  - Receives sensor input from input components
  - Controls external environment via output components

Key

Copyright 2008 H. Gomaa

Distributed Control Pattern

- Several control components
- Control is distributed among the components
- Each component controls part of system
  - Executes statechart
- Control components communicate with each other to provide overall control

Copyright 2008 H. Gomaa
Hierarchical Control Pattern

- Hierarchical Controller
  - Coordinator component
    - Provides high level control
    - Sends commands to each control component

Architectural Communication Patterns for Software Product Lines

- Asynchronous communication patterns
- Synchronous communication patterns

- Very important for evolutionary design:
  - Broker Communication Patterns
    - Broker forwarding
    - Broker handle
    - Discovery
  - Group Communication Patterns
    - Broadcast
    - Subscription/notification
Software Architecture for Microwave Oven - asynchronous communication Pattern

• Subscription/Notification Pattern
  - Client subscribes to join group
  - Receives messages sent to all members of group

Copyright 2008 H. Gomaa
Building Systems and SPL from Software Architectural Patterns

- Consider architectural structure patterns
  - Different patterns can be combined
- Start with layers of abstractions pattern
  - Incorporate client/server patterns
  - Incorporate control patterns
- Apply architectural communication patterns

Building Emergency Monitoring System From Software Architectural Patterns

Architectural Structure Patterns
- Layered pattern
- Client/Server patterns

Architectural Communication Patterns
- Synchronous
- Asynchronous
- Broker
- Subscription/Notification
• Evolve **Emergency Monitoring** System

![Diagram of Emergency Monitoring System]

- **User Interface Component**: OperatorInterface
- **Data Collection Component**: EventMonitor
- **Server Component**: EventHandlingServer
- **Control Component**: DistributedController

**Event Notification**

**Input Component**: SensorNode
**Output Component**: ActuatorNode

Copyright 2008 H. Gomaa
Overview

- Software Modeling and Design
- Software Modeling and Design Methods
- Designing Evolutionary Systems and Product Lines
- Designing and Evolving Systems from Architectural Design Patterns
- **Executable Models of Software Designs (with R. Pettit)**
- Dynamic Software Reconfiguration

---

Modeling Executable Software Architectures

- Design and analyze concurrent software architecture
- Behavioral design patterns
  - Concurrent component
  - Connector
  - Mapped to Colored Petri Net template
- Map concurrent software architecture to CPN model
  - Select and Interconnect CPN templates for components and connectors
- Analyze CPN model
  - Application behavior
  - Application performance

Concurrent Software Architecture

- Uses component / connector paradigm
  - Component (application object)
    - Concurrent object with single thread of control
    - Use COMET concurrent object (task) structuring criteria
    - Define behavioral design pattern for component
  - Connector
    - Provides message communication between components
- CPN behavioral design template provided for each
  - Component
  - Connector

Asynchronous I/O Pattern

- I/O component
  - Handles external input/output on demand
- CPN pattern
  - Thread of control maintained by control token
  - Each component has its own control token
- CPN Transition executes function
  - Processing time associated with transition
- Colored tokens to differentiate role of tokens
  - Control token
  - Input event
  - Output message
Asynchronous I/O Pattern

(a) I/O component

CPN I/O component template

(b) Connectors

- Connector
  - Provides message communication between concurrent components
    - Queue - Asynchronous communication
    - Buffer - Synchronous communication
  - Interface to connector uses CPN places
    - Facilitates interconnection between concurrent component templates and connector templates
Synchronous Communication Pattern

- Synchronous buffer models synchronous communication
- Producer sends message and waits for reply
- One message at a time allowed in the buffer
- Producer and consumer are blocked until message has been passed

synchronous message

(a)

(b)

CPN buffer connector template
Constructing CPN Model from UML Concurrent Design Model

1. Develop COMET/UML design model
   – COMET structuring criteria
2. Construct Architecture-Level CPN Model
   – Represent each component & connector by CPN template
   – Templates developed using DesignCPN
   – Interconnect CPN templates
3. Model characteristics of individual component
   – Customize CPN templates for application
4. Exercise model in DesignCPN simulator
   – Analyze functional behavior
     • Detect and correct design problems
   – Analyze performance characteristics
     • Does software architecture meets timing constraints?

Connecting CPN Templates to form CPN Architecture

I/O component async message

CPN I/O component template

CPN queue connector template
Overview

- Software Modeling and Design
- Software Modeling and Design Methods
- Designing Evolutionary Systems and Product Lines
- Designing and Evolving Systems from Architectural Design Patterns
- Executable Models of Software Designs
- Dynamic Software Reconfiguration (with M. Hussein)

Dynamic Software Reconfiguration in Safety Critical Systems

- Safety Critical Systems
  - Highly available, Time critical
- Examples: air traffic control systems, spacecraft, automotive and aircraft control systems
- Challenge
  - Evolve the configuration of software application
    - At run-time
  - Application must be operational during dynamic reconfiguration
Approach

- Most product line research aimed at deriving different family members from
  - Product line architecture + implementation
  - At Configuration Time
  - NOT at Run Time
- Research approach
  - Model all configurations of safety critical system as product line members
  - Dynamically change from one family member to a different family member at Run Time
  - Develop Software Reconfiguration Patterns

Evolutionary Product Line Life Cycle
- Build, then Deploy

Copyright 2008 H. Gomaa
Reconfigurable Evolutionary Product Line Life Cycle
Reconfigure after Deployment

Software Reconfiguration Patterns

- **Concept**
  - Develop software reconfiguration patterns for well-known software architectural patterns
  - Reconfiguration Pattern
    - Specifies how set of components cooperate to dynamically change system configuration
- **Software Reconfiguration Patterns developed**
  - Master-Slave pattern
  - Centralized Control pattern
  - Client / Server pattern
  - Decentralized Control pattern
Decentralized Control Reconfiguration Pattern

- Decentralized Control components communicate with each other
  - Components must notify each other if going quiescent
  - Component can cease to communicate with neighbor but can continue with other processing

Reconfiguration State Machine

- Reconfiguration state machine model
- Component transitions to a state where it can be reconfigured
  - Active State
    - Component is operational
  - Quiescent State
    - Component Idle
    - Can be removed from configuration
Example of Reconfiguration Scenario

- Reconfigurable factory automation SPL architecture
- Uses: Master-Slave, Client / Server, & Decentralized Control patterns

a) Before reconfiguration

- Line Workstation Controller M - 1
- Pick & Place Robot M
- Manufacturing Robot M

b) After reconfiguration

- Line Workstation Controller M - 1
- Pick & Place Robot N
- Manufacturing Robot N

Summary and Conclusions

- Software Modeling and Design
- Designing Evolutionary Systems and Product Lines
  - Evolution built into software design method
  - Architecture-Centric Evolution
- Designing and Evolving Systems from Architectural Patterns
- Executable Models of Software Designs
- Dynamic Software Reconfiguration
- Other related research
  - Software performance modeling (with D. Menasce)
  - Multiple-view meta-modeling (with M. Shin)
  - Tool support for SPL development and product derivation (many)
  - SPL model-based testing (with E. Olimpiew)
  - Separation of concerns in multiple-view models (with Saleh & Shin)
  - Software process modeling (with L. Kerschberg)
  - Design of Service-Oriented Architectures (with J. Street)