Service-Oriented Product Lines: Towards a Development Process and Feature Management Model for Web Services

SOAPL 2008
Introduction

§ What is software development?
  • Usage of a software development process
  • Transform requirements into different artifacts (architectural descriptions, interface descriptions, source code…)

§ How to manage artifacts?
  • Apply changes to existing artifacts
  • Reduce coupling of source code

§ What about reuse?
  • Commonality and variability

§ Combination of Software Product Lines and Service-Oriented Architectures provides solutions to many common software problems
Structure

- Introduction
- Definitorial Background
- Development Process for Software Product Lines
- Service-Oriented Product Lines
- Example
- Conclusions
Definitional Background: Software Product Lines

- Withey: „Product Lines is a group of products sharing a common, managed set of features“ [1]
- Specifically, manage variability among features that represent requirements
- Goal: Structure and reuse software development artifacts
Definitional Background: Service-Oriented Architectures

- Loosely coupled and autonomous services
- Properties according to Josuttis: self-containment, coarse-grained interfaces, reusability and composability [2]
- Implementation: Web Services or Enterprise Service Bus
Definitional Background: Web Services

§ „Software applications that can be discovered, described and accessed based on XML and standard Web protocols“ [3]

§ Described by a WSDL
  • Abstract definition describes interface, operations and messages
  • Concrete definition describes bindings to operations

§ Distinguish into service broker, provider and consumer
Structure

- Introduction
- Definitorial Background
- Development Process for Software Product Lines
- Service-Oriented Product Lines
- Example
- Conclusions
Domain Engineering

- **Develop a software family**

- **Analysis**
  - Capture domain specific knowledge
  - Develop a domain model
  - Represent domain concepts and requirements in a central feature model
  - Identify variants with their distinguishing features

- **Design**
  - From architectural description to software entities
  - Decide used frameworks, libraries and programming languages
  - Form technological foundation for implementation of variants

- **Implementation**
  - Make or buy decision for software entities
Application Engineering

Develop individual member (of the software family)

Five steps

- Problem Analysis (overall problem specification)
- Product Specification (concrete set of selected features)
- Collateral Development (Documentation)
- Product Implementation (Executables and test cases)
- Deployment
Structure

- Introduction
- Definitorial Background
- Development Process for Software Product Lines
- Service-Oriented Product Lines
- Example
- Conclusions
Service-Oriented Product Lines

- Implement SPL with an SOA

- Different impacts on development phases
  - Analysis:
    - Select SOA-specific modeling languages
    - Software requirements can be modeled as features or part of the ESB
  - Design
    - ESB as routing and messaging backbone, and also implements e.g. compliance requirements
    - ESB mostly forms common part of SPL
    - Web Service abstracts whole applications, databases or fine granular software entities
  - Implementation
    - Careful choice of purchased ESB
    - Wrap existing software with Web Services or use web service repositories
    - Full SOPL process (design interface and implementation) vs. light SOPL process (design only interfaces)
Structure

- Introduction
- Definitorial Background
- Development Process for Software Product Lines
- Service-Oriented Product Lines
- Example
- Conclusions
Domain Engineering for a Web Store (base taken from [4])

Web store contains 7 modules:
Web Store: A Feature Model

- Abstract representation of the overall product line
- Uses set-like notations for features
  - Base = \{Acq, Chk, Crd, Ord, Shp, Bil, Pay\}
- Detail out features
  - Credit Ranking: Use an independent agency (Agc) or explanation of the bank (Bak)
    - Crd = \{Agc, Bak\}
  - Shipment via surface (Sur) or airmail (Air)
    - Shp = \{Sur, Air\}
  - Surface shipment with standard (Std) or Express (Exp) Mail
    - Sur = \{Std, Exp\}

- Individual member is a composition of specific features
  - Store1 = Base
  - Acq • Chk • Agc • Bak • Ord • Std • Exp • Air • Bil • Pay
Customers demand new features
- Discounting for bigger quantities of ordered goods
- Traceability of features

Impacts existing services of the Web Store
§ Discounting concepts refines four basic features
  
  \[
  \text{Disc} = \{\text{Crd} = \{\Delta \text{Agc}, \Delta \text{Bak}\}, \Delta \text{Bil}, \Delta \text{Pay}\}
  \]

§ Build a new member
  
  • Include discounting feature
  • Limit shipment to standard surface mail
  • \(\text{Store}_2 = \{\text{Base} - \{\text{Exp, Air}\}\} \cdot \text{Disc}\)
  
  • \(\text{Store}_2 = \text{Acq} \cdot \text{Chk} \cdot \text{Agc} \cdot \text{Bak} \cdot \text{Ord} \cdot \text{Std} \cdot \text{Bil} \cdot \text{Pay} \cdot \text{Disc}\)
  
  • \(\text{Store}_2 = \text{Acq} \cdot \text{Chk} \cdot \text{Agc} \cdot \text{Bak} \cdot \text{Ord} \cdot \text{Std} \cdot \text{Bil} \cdot \text{Pay} \cdot \Delta \text{Agc} \cdot \Delta \text{Bak} \cdot \Delta \text{Bil} \cdot \Delta \text{Pay}\)

§ Combination of a basic and refined feature leads to the final representation
  
  • \(\text{Store}_2 = \text{Acq} \cdot \text{Chk} \cdot \text{Agc'} \cdot \text{Bak'} \cdot \text{Ord} \cdot \text{Std} \cdot \text{Bil'} \cdot \text{Pay'}\)
Description of Web Services with WSDL gives a high level view

Feature granularity must manage WSDL descriptions

Example: WSDL for Billing

- `<element name="CalcBillOutput">
- <!-- Other definitions ommitted -->
- `<xsd:sequence>
- `<xsd:element name="customerName" type="xsd:string"/>
- `<xsd:element name="customerAddress" type="xsd:string"/>
- `<xsd:element name="items" type="ItemOrder" minOccurs="1" maxOccurs="unbound"/>
- `<xsd:element name="totalPrice" type="xsd:integer"/>
- </xsd:sequence>
- <!-- Other definitions ommitted -->
- </element>

Variability Management with XAK [5]

- `<element name="CalcBillOutput" xak:artifact="STOREbillOutput">
- <!-- Other definitions ommitted -->
- `<xsd:sequence xak:module="billOutput">
- `<xsd:element name="customerName" type="xsd:string"/>
- <!-- Other definitions ommitted -->

§ WSDL Refinement

- `<xak:refines xak:artifact="STOREbillOutput">`
- `<xak:extends xak:module="billOutput">`
- `<xak:super xak:module="billOutput"/>
- `<xsd:element name="discount" type="xsd:integer"/>
- `<xsd:element name="discountedPrice" type="xsd:integer"/>
  - `<xak:extends>
  - `<xak:refines>

§ Combined WSDL

- `<element name="CalcBillOutput">`
- `<xsd:sequence>`
  - `<xsd:element name="customerName" type="xsd:string"/>
  - `<xsd:element name="customerAddress" type="xsd:string"/>
  - `<xsd:element name="items" type="ItemOrder" minOccurs="1" maxOccurs="unbound"/>
  - `<xsd:element name="totalPrice" type="xsd:integer"/>
  - `<xsd:element name="discount" type="xsd:integer"/>
  - `<xsd:element name="discountedPrice" type="xsd:integer"/>
  - `<xsd:sequence>`
  - `<xsd:sequence>`
- `<element>`
- `<xsd:sequence>`
- `<xsd:sequence>`
- `<xsd:sequence>`
- `<xsd:sequence>`
Conclusions

- Introduction
- Definitorial Background
- Development Process for Software Product Lines
- Service-Oriented Product Lines
- Example
- Conclusions
Conclusions

- Feature models and variability management models can be used for Service-Oriented Product Lines as well

- XML refinements allow practical solution to feature management

- Focus on models leads to a high-level view

- Promising
  - If existing code base can be reused efficiently: focus on light SOPL process (only define interfaces)
  - Introduce Domain Specific Languages for domain modeling and SPL configuration, allowing participation of end-users
Thanks for your attention!
References


