Configuring Members of a Family of Requirements Using Features

Jan Bredereke
Universität Bremen, Germany
June 29, 2005

Motivation: Family of Systems
first system:

Motivation: Family of Systems
second system:

Motivation: Family of Systems
third system:
The Problem: Feature $\neq$ Requirements Module

Solution: Configuring Requirements Modules in Z

Example: A Family of LAN Message Services

Outline

(Naive) Feature Orientation

- base system plus separate features as needed
- arbitrary increments
  - chosen from marketing perspective
  - marketing cannot care about structure of software or organization of requirements
- attractive!
- feature interaction problems
  - needed: organize requirements for change

Concentrate on Requirements

- all feature interaction problems: inherently present in requirements

Which Structure for Requirements?

- a set of properties likely to change together
- one family member
- a set of features

Jan Bredereke  Configuring Requirements Using Features
**Observation: Feature ≠ Requirements Module**

1. **type mismatch:**
   - requirements module: a set of properties = 1 set
   - feature: a set of changes = added & removed props. = 2 sets

2. **different grouping criterion** for properties:
   - requirements module: likeliness of change, averaged over entire family
   - feature: marketing needs of single situation

**Definition: Requirements Module**

requirements module
a set of properties that are likely to change together

likeliness to change together
- properties hold / don’t hold for how many family members?

**Hierarchy of Requirements Modules**

- handle really huge number of properties?
  - configure many requirements conveniently?
  - find requirement in large document?

- group them again and again: recursive structure!
  - modules inside modules
  - top-level modules: most stable
  - leaf modules: most likely to change
### Features as Configuration Rules for Req. Modules

- **family**
  - **system**
    - **feature**
      - **reqs. module**
        - **property**

### ZF: A Requirements Module Construct and a Feature Construct for Z

- well-known formal language Z
  - explicit hierarchical modules
    - feature construct
    - type rules, for consistency
    - [explicit interfaces between requirements modules]

### Configuring Requirements Modules Using Features in ZF

- **requirements module base** (sections, grouped into chapters)
- **configuration rule base** (definitions of features)
- **configurator** (transformation function)
- **requirements family member** (selected requirements modules)

### Formal Definition of ZF

- brief: in ICFI'05 paper
- in detail: in my book
  (is on my Web page: Habilitation thesis)
Outline

The Problem: Feature != Requirements Module
Solution: Configuring Requirements Modules in Z
Example: A Family of LAN Message Services

Example: A Family of LAN Message Services

idea

users on a LAN can send each other short messages
▶ example: “I cut birthday cake in 5 minutes"

less complex than full telephony

variabilities
▶ individual addressing
▶ message blocking
▶ message re-routing
▶ output on text console
▶ delayed messages
▶ . . .

The LAN Message Family Specification

1. chapter environment
   1.1 chapter device_interfaces
   1.1.1 chapter communicating_entities
      1.1.1.1 private chapter user_interface
      1.1.1.1.1 section user_base
         parents comm_base
   . . .
      1.1.1.1.2 private chapter graphical_user_interface
      1.1.1.1.2.1 section gui.comm_base
         parents comm_base
      . . .
   1.1.1.2 private section gui_io_base
      parents gui.comm_base, comm_io_base
   . . .

Complete Module Hierarchy and Dependencies
The Problem: Feature != Requirements Module
Solution: Configuring Requirements Modules in Z
Example: A Family of LAN Message Services

The LAN Message Services Specification
Features of the Family

**Top-Level Requirements Modules**

- **system Behaviour**
  - function drivers
  - message delivery
- **Environment**
  - computing platform
  - data types
  - distributed processing
- **Device Interfaces**
  - device interfaces
  - time
  - communicating entities

**Features of the LAN Messages Family, in Z_F Syntax**

- **feature note_to_all:**
  - broadcast_message_delivery
  - text_message_base
  - (+) one_line_message
- **feature scroll_text_message:**
  - multi_line_message
  - (+) one_line_message
  - max_lines1000_message
  - graphical_user_interface
  - textual_user_interface
- **feature birthday_cake_picture:**
  - broadcast_message_delivery
  - graphical_message_base
  - text_message_only
  - graphical_user_interface

- **feature lunch_alarm:**
  - automated_agent_interface
  - broadcast_message_delivery
  - text_message_base
- **feature deskPhoneXY_hardware:**
  - graphical_user_interface
  - textual_user_interface
  - max_lines2_message
  - pascal_text_string
  - pascal_text_string_only
  - c_text_string

**Family Members of the LAN Messages Family, in Z_F**

The “Lunch Phone” system: `lunch_alarm
deskPhoneXY_hardware`

} one input for configurator

The “Classic PC” edition:
- note_to_all
- multi_line_text_message
- standardPC_hardware

The “Deluxe PC” edition:
- lunch_alarm
- birthday_cake_picture
- note_to_all
- multi_line_text_message
- scroll_text_message
- standardPC_hardware

“Lunch Phone”: Base System + Two Features

base system:
“Lunch Phone”: Base System + Two Features

feature lunch_alarm:

```
lunch phone = base + lunch_alarm + deskphoneXY_hardware:
```

An Inconsistent Configuration: Type Error in $Z_F$

base system:
An Inconsistent Configuration: Type Error in $Z_F$

**feature** birthday
cake
deskphoneXY

Detecting Inconsistent Configuration Rules / Features

- some inconsistencies are made type errors
- important case: include & exclude same property
- detect automatically
The Problem: Feature ≠ Requirements Module
Solution: Configuring Requirements Modules in Z
Example: A Family of LAN Message Services

The LAN Message Services Specification
Features of the Family

Summary

- feature ≠ requirements module

<table>
<thead>
<tr>
<th>requirements module</th>
<th>feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>a set of properties</td>
<td>a set of changes</td>
</tr>
<tr>
<td>for long-lived family</td>
<td>for single situation (marketing)</td>
</tr>
<tr>
<td>provides an abstraction</td>
<td>a configuration rule</td>
</tr>
</tbody>
</table>

- applied to formalism Z
  - configure specifications in Z
  - detect inconsistent configurations as type errors

- Outlook
  - associate code fragments to requirements
  - policies and families
  - application to other formalisms

- reduce number of “hard” conflicts:
  - differentiate the strictness of rules
  - essential property
  - changeable property
  - classification by original specifier
  - priority is per feature

Resolving Inconsistent Configuration Rules

Abstract Interfaces

More Examples for Type Rules and Semantics of Z

Resolving Inconsistent Configuration Rules

Abstract Interfaces

More Examples for Type Rules and Semantics of ZF

Resolving Inconsistent Configuration Rules

Abstract Interfaces
Interfaces Restrict Access

The Access Rules for Modules in $\mathcal{ZF}$

- Anything can depend on an interface.
- An interface never depends on a secret.
- A secret can depend on a secret only if they are siblings.

Legend:
- $\rightarrow$ dependency
- private
- public
- $x \rightarrow y$ means $x$ depends on $y$.

Jan Bredereke | Configuring Requirements Using Features