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

some change:
Motivation: Family of Systems

second system:
Motivation: Family of Systems

another change:
Motivation: Family of Systems

third system:
Outline

The Problem: Feature != Requirements Module

Solution: Configuring Requirements Modules in Z

Example: A Family of LAN Message Services
(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
(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
(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?

```
+---------+  +---------+  +---------+
| family  |  | family  |  | family  |
| system  |  | system  |  | system  |
| 1       |  | 1        |  | 1        |
| *       |  | *        |  | *        |
| property|  | property |  | property |
```

Jan Bredereke

Configuring Requirements Using Features
Which Structure for Requirements?

![Diagram showing relationships between family, system, and property]
Which Structure for Requirements?

- **family**
  - 1
  - *
  - *
- **system**
  - *
- **reqs. module**
  - 1
  - *
- **property**
Which Structure for Requirements?

Naive Feature Orientation

- Feature != Requirements Module

Example: A Family of LAN Message Services

<table>
<thead>
<tr>
<th>Family</th>
<th>System</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

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

Jan Bredereke
Configuring Requirements Using Features
Which Structure for Requirements?

- **Naive Feature Orientation**
  - Feature != Requirements Module
  - Example: A Family of LAN Message Services

**Diagram:**

1. **Family**
   - 1
   - *
   - *
   - *
2. **System**
   - *
3. **Requirements Module**
   - 1
   - *
4. **Property**
   - *
   - *

- **One family member**
- **A set of properties likely to change together**
- **A set of changes**

Jan Bredereke
Configuring Requirements Using Features
Which Structure for Requirements?

Naive Feature Orientation
Feature != Requirements Module

Example: A Family of LAN Message Services

one family member

a set of properties likely to change together

Jan Bredereke
Configuring Requirements Using Features
Observation: Feature $\neq$ Requirements Module

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

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

The Problem: Feature ≠ Requirements Module

Solution: Configuring Requirements Modules in Z

Example: A Family of LAN Message Services
Definition: Requirements Module

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

likeness 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 Requirements Modules
Features as Configuration Rules for Req. Modules

- **family**
  - 1
  - *
- **system**
  - *
  - *
- **reqs. module**
  - 1
  - *
  - *
- **property**
  - *
  - *
  - *
  - *

Jan Bredereke
Configuring Requirements Using Features
Features as Configuration Rules for Req. Modules

Diagram:

- **family**
  - 1
  - *
  - **system**
    - *
    - **reqs. module**
      - 1
      - *
      - **property**
        - *
        - **feature**

Jan Bredereke

Configuring Requirements Using Features
Features as Configuration Rules for Req. Modules

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

Jan Bredereke
Configuring Requirements Using Features
Features as Configuration Rules for Req. Modules

- **family**
  - 1
  - *
- **system**
  - *
  - *
  - *
- **feature**
  - *
  - *
  - *
- **reqs. module**
  - 1
  - *
  - *
  - *
- **property**
  - *
  - *
  - *

Jan Bredereke
Configuring Requirements Using Features
Features as Configuration Rules for Req. Modules

- family
  - 1
  - *
- system
  - *
  - *
- feature
  - *
  - *
- reqs. module
  - 1
  - *
  - 1
- property
  - *

- a set of changes
- a set of properties likely to change together
Z_F: 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 $Z_F$
Formal Definition of $\mathcal{ZF}$

- 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
- ...

Jan Bredereke

Configuring Requirements Using Features
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
      ...
   1.1.1.1.2.2 private section gui_io_base
      parents gui_comm_base, comm_io_base
      ...

Jan Bredereke
Configuring Requirements Using Features
The Problem: Feature ≠ Requirements Module
Solution: Configuring Requirements Modules in Z
Example: A Family of LAN Message Services

Complete Module Hierarchy and Dependencies

Legend:
- x \implies y: x depends on y
- \text{public (i.e., interface) module or property}
- \text{private (i.e., secret) module or property}

Jan Bredereke  Configuring Requirements Using Features
Top-Level Requirements Modules
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 System + Two Features

feature deskphoneXY_hardware:
“Lunch Phone”: Base System + Two Features

\[ \text{lunch phone} = \text{base} + \text{lunch\_alarm} + \text{deskphoneXY\_hardware} \]
An Inconsistent Configuration: Type Error in \( Z_F \)

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

feature birthday_cake_picture:
An Inconsistent Configuration: Type Error in $Z_F$

```
feature deskphoneXY_hardware:
```
An Inconsistent Configuration: Type Error in $Z_F$

base + birthday_cake_picture + deskphoneXY_hardware:
Detecting Inconsistent Configuration Rules / Features

- some inconsistencies are made type errors
- important case: include & exclude same property
- detect automatically
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
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
Reserve Slides

More Examples for Type Rules and Semantics of ZF

Resolving Inconsistent Configuration Rules

Abstract Interfaces
More Examples for Type Rules and Semantics of ZF

Resolving Inconsistent Configuration Rules

Abstract Interfaces

More Examples for Type Rules and Semantics of ZF

Jan Bredereke

Configuring Requirements Using Features
Resolving Inconsistent Configuration Rules

- reduce number of “hard” conflicts: differentiate the strictness of rules
  - essential property
  - changeable property
- classification by original specifier
- priority is per feature
Interfaces Restrict Access
Generating One Family Member
The Access Rules for Modules in $\mathbb{Z}_F$

- 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:
- $x \rightarrow y$: $x$ depends on $y$
- public (i.e., interface) module or property
- private (i.e., secret) module or property