

# Configuring Members of a Family of Requirements Using Features

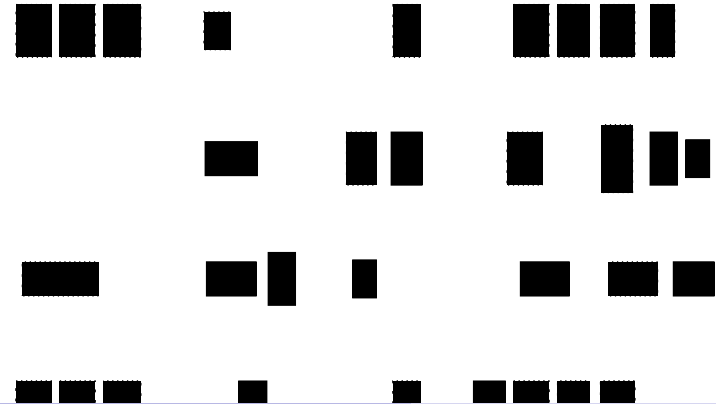
Jan Bredereke

Universität Bremen, Germany

June 29, 2005

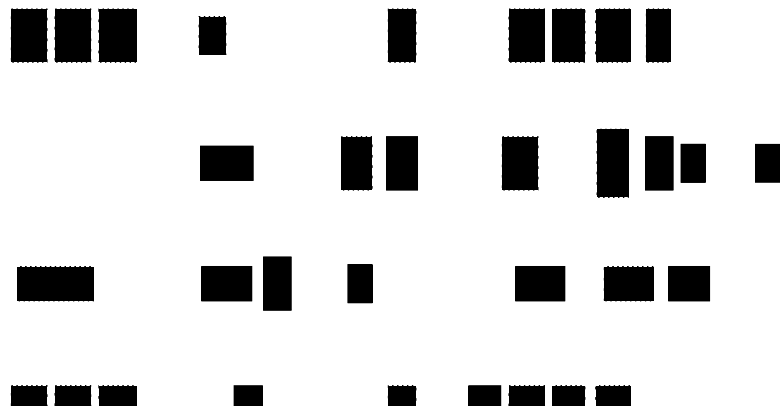
## Motivation: Family of Systems

first system:



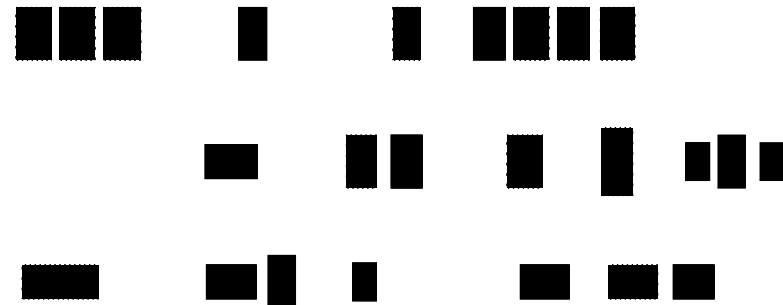
## Motivation: Family of Systems

second system:



## Motivation: Family of Systems

third system:



## Outline

The Problem: Feature != Requirements Module

Solution: Configuring Requirements Modules in Z

Example: A Family of LAN Message Services

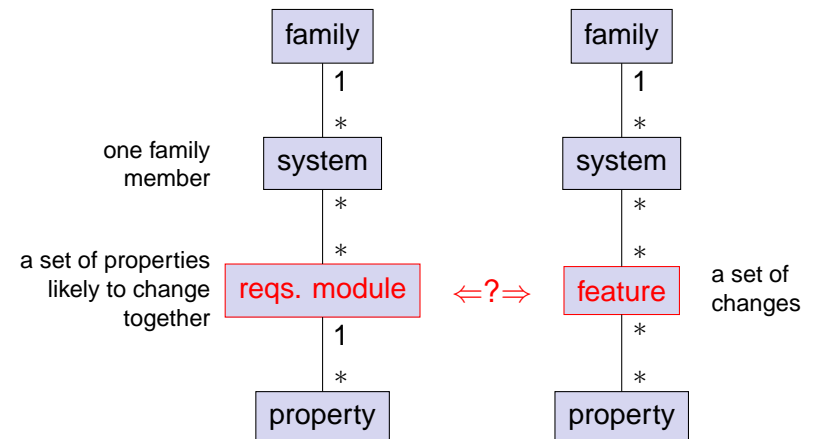
## (Naive) Feature Orientation

- ▶ **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**

## Concentrate on Requirements

- ▶ all feature interaction problems:  
inherently present in requirements

## Which Structure for Requirements?



## Observation: Feature $\neq$ 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**

## 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

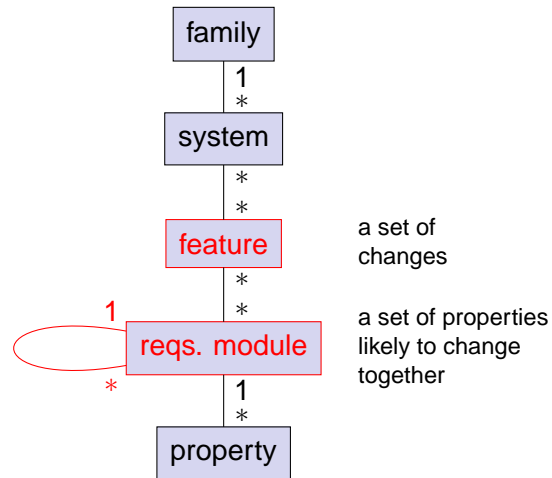
likeliness to change together

- ▶ 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

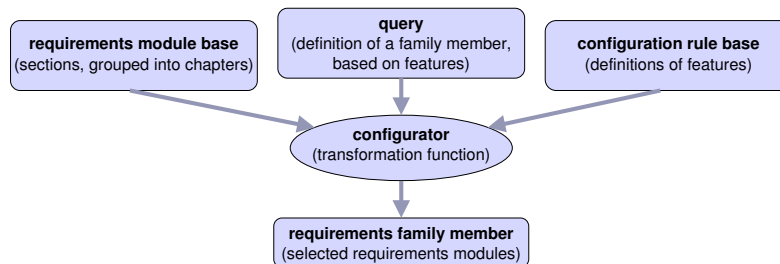
## Features as Configuration Rules for Req. Modules



## Z<sub>F</sub>: 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<sub>F</sub>



## Formal Definition of Z<sub>F</sub>

- ▶ 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

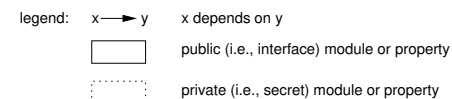
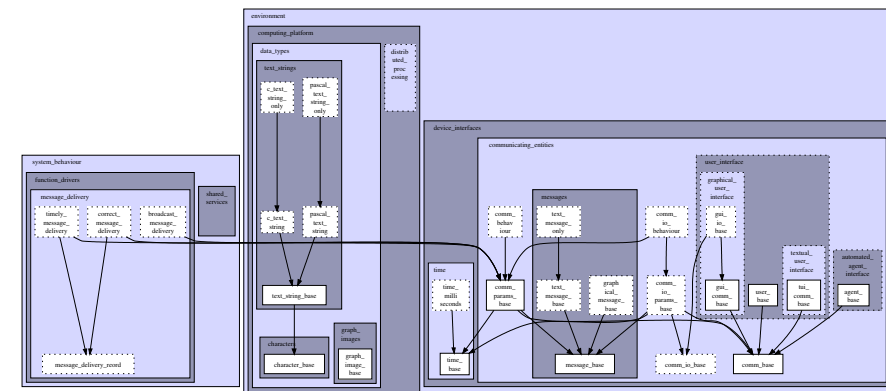
...

###### 1.1.1.1.2.2 private section gui\_io\_base

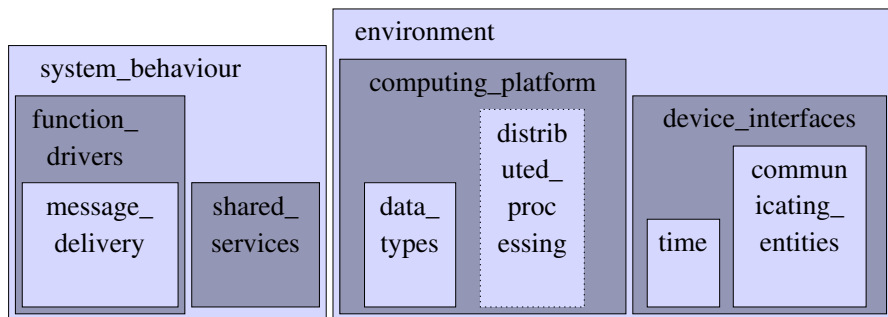
parents gui\_comm\_base, comm\_io\_base

...

## Complete Module Hierarchy and Dependencies



## Top-Level Requirements Modules



## Features of the LAN Messages Family, in Z<sub>F</sub> 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<sub>F</sub>

The "Lunch Phone" system:

lunch\_alarm  
 deskPhoneXY\_hardware

} one input for configurator

The "Classic PC" edition:

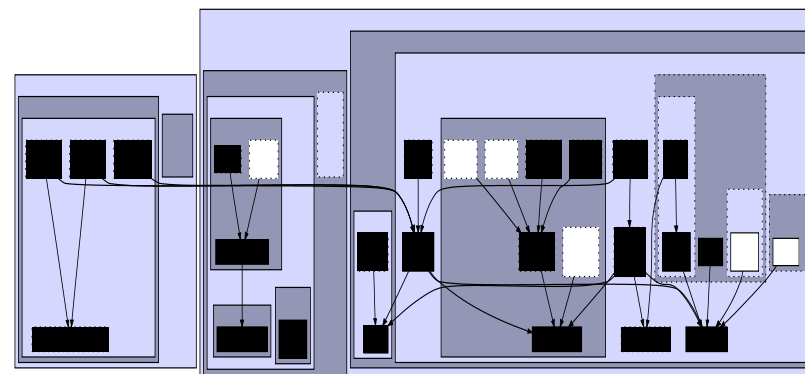
note\_to\_all  
 multi\_line\_text\_message  
 standardPC\_hardware

The "Deluxe PC" edition:

lunch\_alarm  
 birthday\_cake\_picture  
 note\_to\_all  
 multi\_line\_text\_message  
 scroll\_text\_message  
 standardPC\_hardware

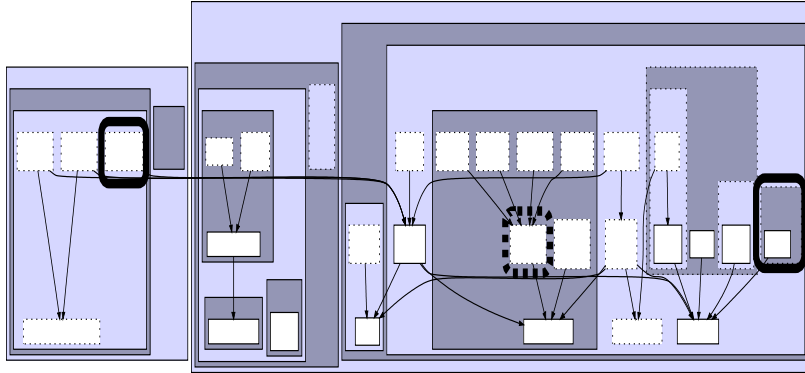
## "Lunch Phone": Base System + Two Features

base system:



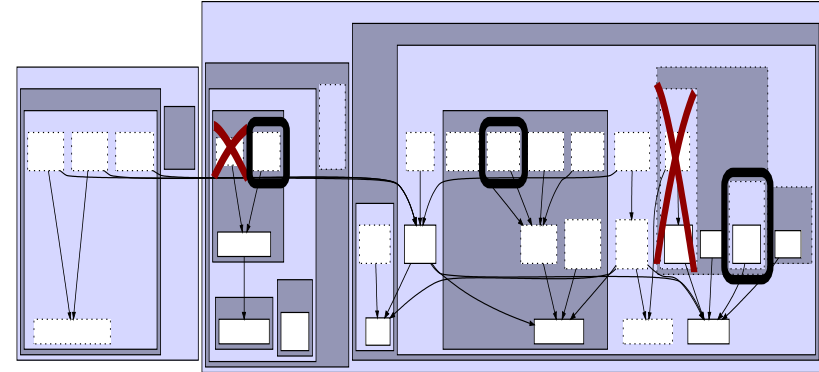
## “Lunch Phone”: Base System + Two Features

feature lunch\_alarm:



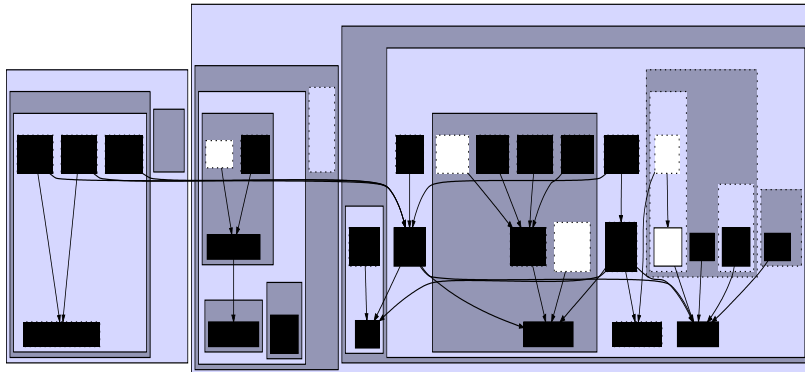
## “Lunch Phone”: Base System + Two Features

feature deskphoneXY\_hardware:



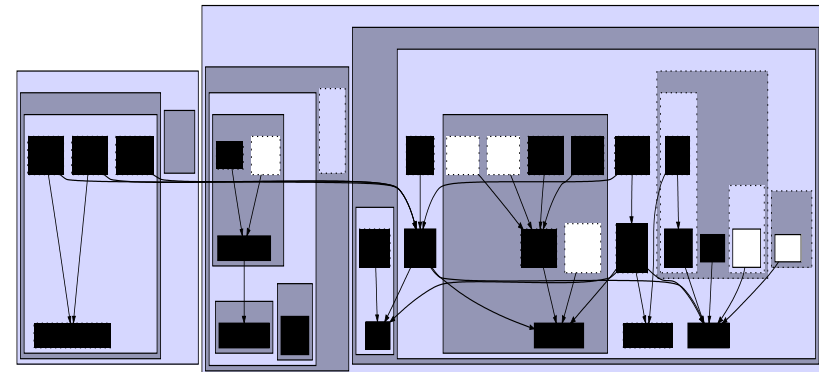
## “Lunch Phone”: Base System + Two Features

lunch phone = base + lunch\_alarm + deskphoneXY\_hardware:



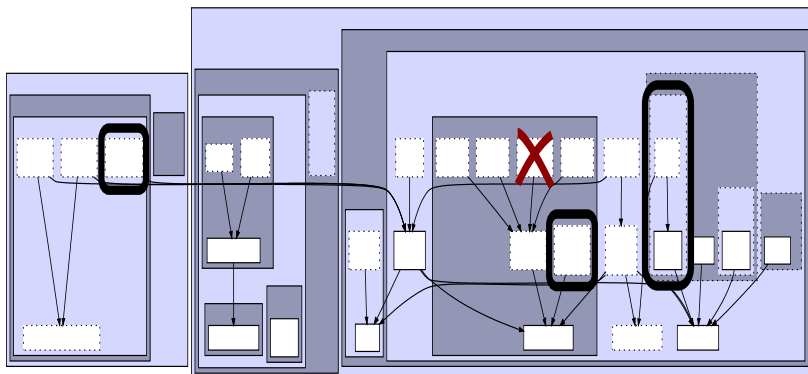
## An Inconsistent Configuration: Type Error in $Z_F$

base system:



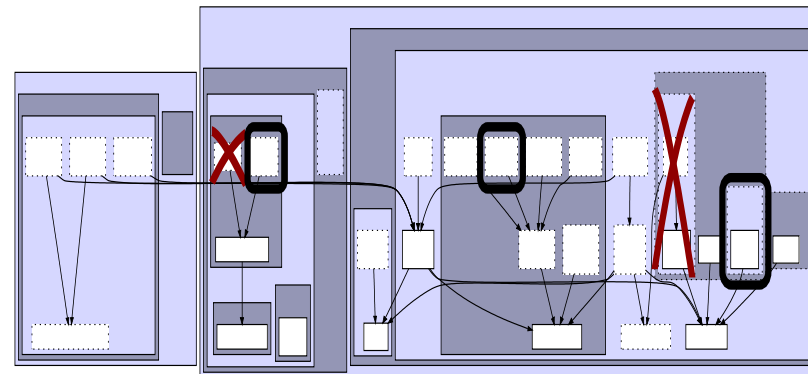
## An Inconsistent Configuration: Type Error in $Z_F$

feature birthday\_cake\_picture:



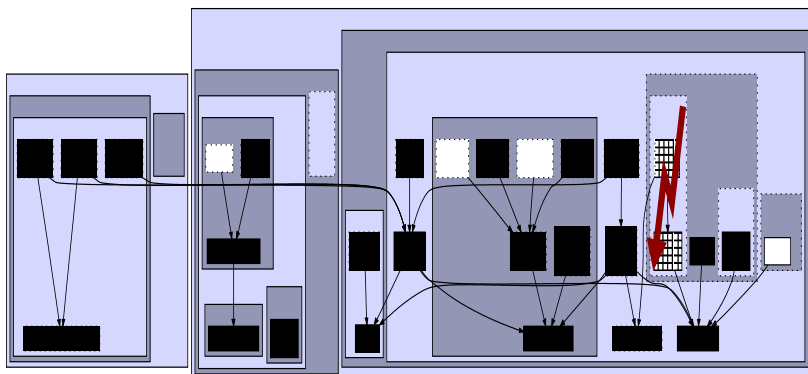
## An Inconsistent Configuration: Type Error in $Z_F$

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  $\neq$  requirements module

requirements module	feature
a set of properties for long-lived family provides an abstraction	a set of changes for single situation (marketing) a configuration rule

▶ 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

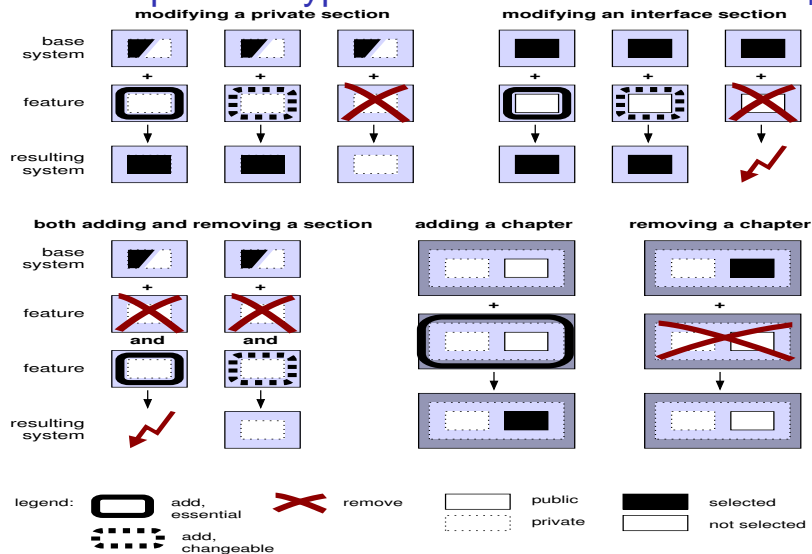
## Reserve Slides

More Examples for Type Rules and Semantics of ZF

Resolving Inconsistent Configuration Rules

Abstract Interfaces

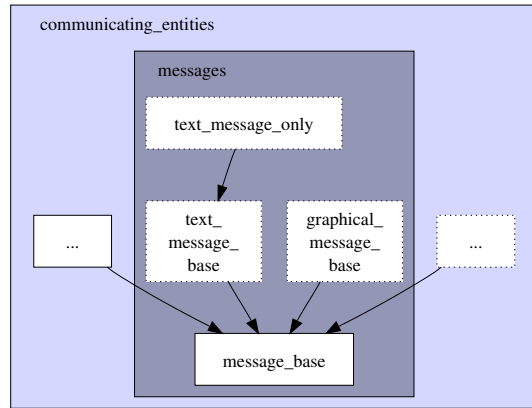
## More Examples for Type Rules and Semantics of Z<sub>F</sub>



## Resolving Inconsistent Configuration Rules

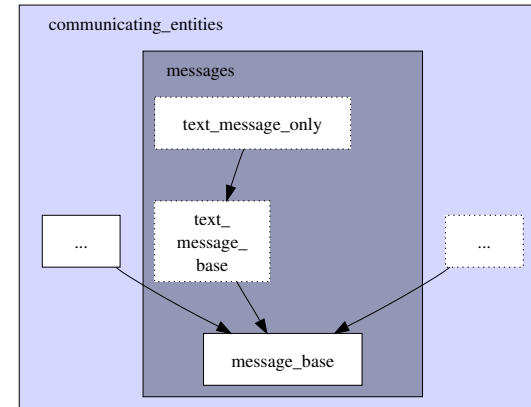
- ▶ reduce number of “hard” conflicts:
  - ▶ differentiate the strictness of rules
    - ▶ essential property
    - ▶ changeable property
- ▶ classification by original specifier
- ▶ priority is per feature

## Interfaces Restrict Access



legend:  public  private  dependency

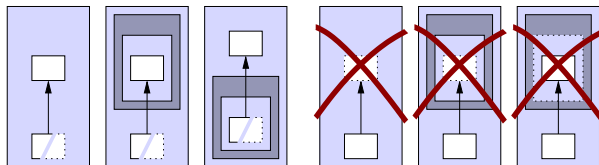
## Generating One Family Member



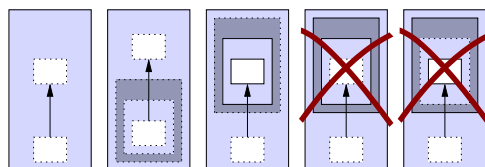
legend:  public  private  dependency

## The Access Rules for Modules in $Z_F$

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