Families of Formal Requirements in Telephone Switching

Jan Bredereke

University of Bremen, TZI
Germany

Titel Page

talk: for 30 min., including discussion

Motivation: Maintenance Problems in Telephone Switching

our concern: formal requirements

- requirements documents complete
  → all FI inherently present

problems in requirements maintenance

- many versions, many revisions
- scope of consequences of a change?
- in state automata:
  which parts of a trace must remain?
  which parts may be changed?

our approach

- view as $\boxed{\textbf{family} \text{ of (formal) requirements}}$

Motivation: Maintenance Problems in Telephone Switching

many versions, many revisions

- **no configuration control support**
  in many formal techniques

⇒ **Family Idea for Requirements**

Family Idea for Requirements

general idea

- **maintain** all variants/features together (**one document**)
  - **generate** specific requirements **family member** automatically as necessary
- **document information** necessary **for changes**:
  - **dependence** of requirements
  - what is the **core of a feature**

Family Idea for Requirements

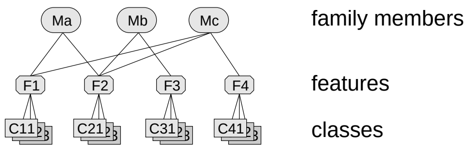
one document

- at least **logically**

⇒ **Structuring the Specification**

Structuring the Specification

- **constraint-oriented** specification style: one class per property
- **group** classes **into features**



- **composition by conjunction**
  - restricted form of **inheritance**

Structuring the Specification

one class per property

- get relatively **small**, but **self-contained units**

group classes

- for **incremental specification**
- figure: **each** class **associated to one** feature
- can **include/exclude entire features**

composition by conjunction

- suitable **for constraint-oriented** style
- **keeps** properties **independent**

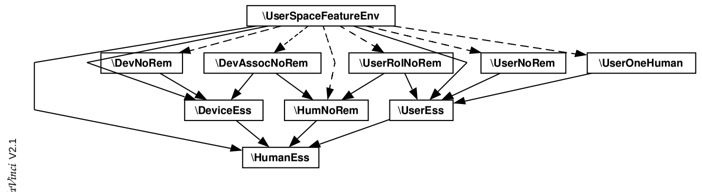
restricted inheritance

- **no overwriting** of things, just **logical "and"**

⇒ **Documenting Dependences by Inheritance**

Documenting Dependences by Inheritance Relation

- **inheritance** (=dependence) **if**
  - **use of** previous **definition**
  - **rely on** previous **constraint**
- **no inheritance**
  → **no access** to definitions
  → **no interference**
- **example:**
  classes of **one feature** of case study:



- example:
  classes of **all features** of one family member:
  . . .

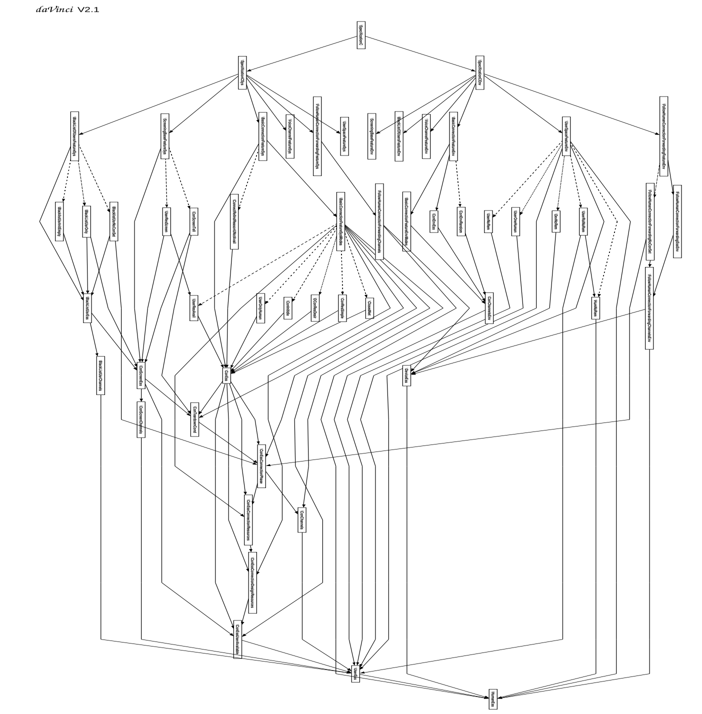
Documenting Dependences by Inheritance Relation

no interference

- **what, if** use **common definition?**
  → conjunction is **monotonic**,
  further restrictions **don't invalidate**
  previous constraints

⇒ **Uses-Hierarchy of Classes for one Member**

Uses-Hierarchy of Classes for one Family Member of Case Study



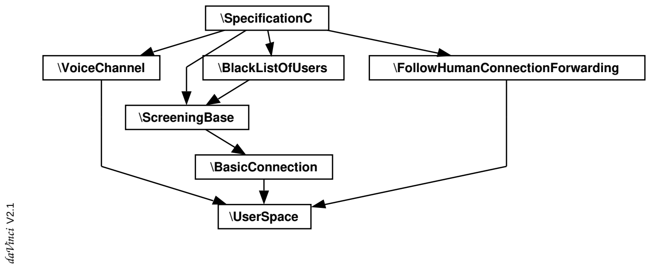
Uses-Hierarchy of Classes for one Family Member of Case Study

- rather **large**
- **only interested in dependences** of features

⇒ **Hierarchy of Features**

Hierarchy of Features

- **extract hierarchy of features**
  from uses-hierarchy of classes
  - example:



- consequence:
  dependence of features explicit, too
- (automatically) exploit dependence info

Hierarchy of Features

(automatically) exploit

- supporting info **for specifiers**
- **automated type check**

⇒ **Non-Monotonicity of New Features**

Non-Monotonicity of New Features

problem

- **most new features** *change* **behaviour**
  of base system or of other features
- change: **consequences?**

our approach

- constraint-oriented specification:
  one property per class
- **add+remove entire properties/classes**
- **document what is essential part** of feature
  - formally:
    mark essential/changeable classes
    by special inheritance operator
- (automatically) exploit info on “essential”

Non-Monotonicity of New Features

change

- lots of **theory available if**:
  - **adding** traces only, **or**
  - **removing** traces only
- **change:** adding **and** removing traces

(automatically) exploit

- **type check**
- **warning** by **heuristics**

⇒ **Detecting FI by Type-Checks**

Detecting FI by Type-Checks

syntactic rules

syntactic **errors**:

- “remove” an “essential” class
- feature of needed class not included
- feature of “removed” class not included
- another class still needs “removed” class

heuristic syntactic rules

syntactic **warnings**:

- class is marked both essential and changeable
- class is “removed” twice

Detecting FI by Type-Checks

⇒ **Formal Technique: (extension of) CSP-OZ**

Formal Technique: (extension of) CSP-OZ

general

- CSP: **control and communication**
- Object-Z: **data**
- developed at **Univ. of Oldenburg**

specific language features used

- **encapsulation of data and behaviour** into a class
- **plain inheritance** (no overwriting used)

language extensions

- **grouping** into features and family members
- **removing** entire classes
- specifying a **feature list**

Formal Technique: (extension of) CSP-OZ

encapsulation

- CSP-OZ is **object-based**, not object-oriented

plain inheritance

- **restricted use** of CSP-OZ

⇒ **Tool Support**

Tool Support

- cspozTC:
  - **type checker** for CSP-OZ
- genFamMem:
  - **generates** a family member
  - **type checker for extended CSP-OZ**
  - extracts **uses-hierarchy**
- daVinci:
  - **visualizes** uses-hierarchy graph
- FDR:
  - **model checker** for CSP (no ...-OZ yet)

Tool Support

cspozTC

- developed at Univ. of **Oldenburg**

genFamMem

- **own work**

daVinci

- developed at Univ. of **Bremen**

FDR

- **commercial** tool
- by **Formal Systems**, Inc., Oxford, UK

no ...-OZ yet

- will have a **translation to FDR** for a **subset**
  - by Univ. of Oldenburg

⇒ **Example of FI Detection**

Example of FI Detection

- from case study
- **two screening features**:
  - BlackListOfDevices
  - BlackListOfUsers
- **both** features **"remove"** the **same class** of an underlying feature
- → **warning** from **type checker**
- manual **analysis shows**:
  1. each feature provides a replacement class
  2. these two classes are *not* contradictory
  3. but each feature has **another class** that states it's the **only screening feature** → **contradiction!**
  4. **both** contradicting classes are **changeable**
  5. **resolution:** specify **another feature** which removes both and which provides a suitable replacement

type checker here worked as **FI filter**

Example of FI Detection

BlackListOfDevices

- **prevents connections** originating from **devices** on a **list**

BlackListOfUsers

- **prevents** connections originating from **users** on a list
- use when the **system can distinguish** the different **users at** a **device**

contradiction

- problem in constraint-oriented style: there may not be any correct implementation
- approach: find valid initial state
- we restrict ourselves to invariance properties, no required progress (not compositional in the way we need it)

restricted progress checks

- no deadlock: check by model checking tools

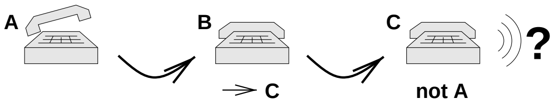
FI filter

- pointed us to critical spots

⇒ **Avoiding FI**

Avoiding FI

standard example: TCS–CF



our case study

- **distinguish concepts** of device/human/user
  already in base system
  before any feature is defined
- 2 forwarding features (device/user)
- 2 black list features (device/user)
- no FI between forwarding/screening

family planning is important!

Avoiding FI

avoiding FI

- **even better** than detecting

already in base system

- but:
  changeable property that **equates**
  these **concepts**

before any feature is defined

- necessary:
  - **analyze what may come**
  - **prepare up-front**

(• comparable approach:
SCV analysis for families of programs
(they generate programs from families))

family planning

- **introduce concepts early** and **explicitly**
- **separate** introduction of **concept**
  and **its first use** (into different “features”)
  - we separated: **connection / voice channel**

⇒ **Summary**

Summary

maintenance of formal requirements

1. **maintain** all variants/features together
   - **generate** specific requirements **family member**
     automatically as necessary
2. **document information** necessary **for changes**:
   - **dependence** of requirements:
     by inheritance and its class uses-hierarchy
   - what is the **core of a feature**:
     by explicitly essential/changeable classes
3. **exploit info**
   - supporting **info for specifiers**
   - **automated type check**
   - **heuristic** warnings (FI **filter**)
4. **avoid FI:**
   family planning

Summary

⇒ **End**