A Tool for Generating Specifications from a Family of Formal Requirements

Jan Bredereke

Universität Bremen
Germany

Overview

- **domain:** telephone switching
  - feature interaction problems
- **methodology**
  - families of formal requirements
  - non-monotonous changes
- **application**
  - case study
  - tool

Feature Interaction (FI) Problems in Telephone Switching

- features **work separately**, but **not together**
- **hundreds** of features
- **combinations** cannot be checked anymore

feature

- about **any increment** of functionality
  - worse than with components

telephone switching

- users' **expectation high**
  - similar to dependable systems

FI Example: Calling Card + Voice Mail

#-button

- (Bell) calling card:
  start new call without re-authorization
- (Meridian) voice mail:
  end of mailbox number, end of password, . . .

call voice mailbox using calling card??

either:

- early disconnect, or
- calling card feature crippled

resolution by Bell

- introduce new signal:
  "#-button pressed at least 2 sec."

FI Example: Call Waiting + Call Forward on Busy

both activated

- in busy state
- when another call arrives

only one can get control

- no resolution, except restrictions on features

FI Example: Originating Call Screening + Area Number Calling

OCS

- aborts calls to numbers in list
- query Service Data Point (SDP) for list

ANC

- dialled number + area(calling number) $\rightarrow$ called number
- example: Domino's Pizza
- query SDP for called number

FI Example:
Originating Call Screening + Area Number Calling (2)

switch may restrict no. of queries

- protection against infinite loops
- e.g., one query per call
- $\rightarrow$ OCS subscription prevents orders for pizza

solution: one more query??

FI Example: Calling Number Delivery + Unlisted Number

conflict of goals

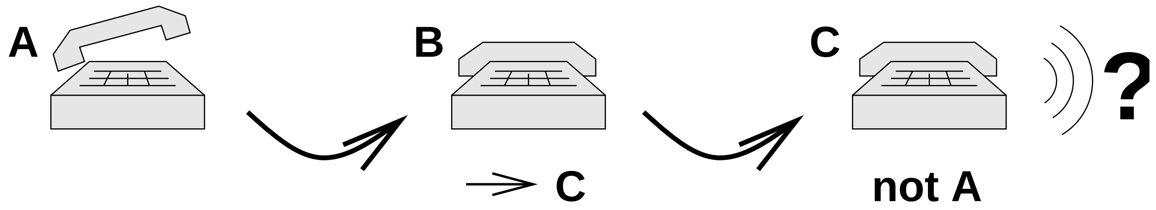
- CND reveals caller
- UN prevents revealing caller

resolution

- weaken one feature
- e.g.: CND delivers only 1-111-1111-1111 for unlisted number

FI Example: Call Forwarding + Terminating Call Screening

Will phone C ring?



- **Terminating Call Screening** when A is caller, C blocks him

  Call Forwarding B forwards all calls to C

  feature interaction can/should A reach C?

- **notion of “caller”** is crucial

Formal Analysis of Feature Interaction Problems?

yes, but. . .

- formalization is **huge task**
- **complexity** not amenable to tools

Maintenance of Formal Requirements

goal: make FI analysis feasible

- our **focus**: formal **requirements**
- **modular** requirements specification

Families of Formal Requirements

- **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**

Maintenance of Formal Requirements

our focus: formal requirements

- requirements **documents complete**
  $\rightarrow$ all feature interactions **inherently present**

modular requirements specification

- **complete** requirements are **not sufficient**,
  need also some **structure**
- modularity in the sense of **Parnas**, i.e., **information hiding**

family idea

- transferred **from SW engineering** of programs to **requirements** of telephone systems

$\Rightarrow$ **Incremental Specification**

Incremental Specification

- standard means: **refinement**

  step by step $\left\{ \begin{array}{c} \text{extend behaviour} \\ \text{or} \\ \text{impose constraints} \end{array} \right\}$ while preserving interesting properties

- but . . .

  in telephone switching: **new features change the behaviour**

- a telephony feature is $\boxed{\textbf{non-monotonous}}$

Incremental Specification

new features change the behaviour

- . . . **of** the **base** system **or** of other **features**
- **example: Call Forwarding**
  - stops to connect to the dialled number (**restricts** base system behaviour),
  - starts connecting to the forwarded-to number (**extends** the base system behaviour)

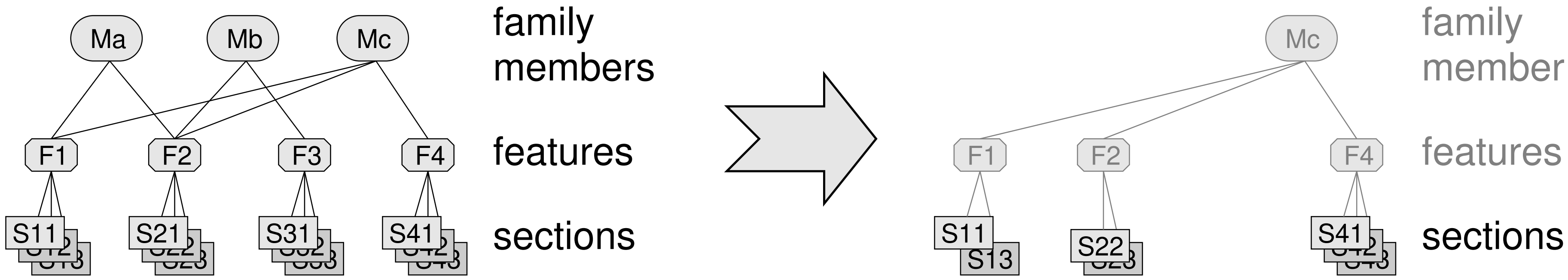
non-monotonous

- **various** feature **operators** have been **proposed**
- analytical **complexity** too big **for tools**, too
- **reduce** complexity by **modularization** in the information hiding sense
- our **feature construct**: allows non-monotonous changes, but **not arbitrary changes**

⇒ **Generating Family Members from a Family Document**

Generating Family Members from a Family Document

family of requirements **requirements specification**



extension of CSP–OZ

plain CSP–OZ

formal language

- extension of **CSP-OZ**

Generating Family Members from a Family Document

CSP-OZ

- combination of **CSP** and **Object-Z**
- "`feature`" and "`familymember`" are already two of our extensions

requirements specification

- the generator **selects** the sections of **one family member**
  - **grey** part: extensions are **comments** in plain CSP-OZ
- the generator also **removes** some sections
  new language construct for this
- this **allows non-monotonous** increments

⇒ **Case Study**

Case Study

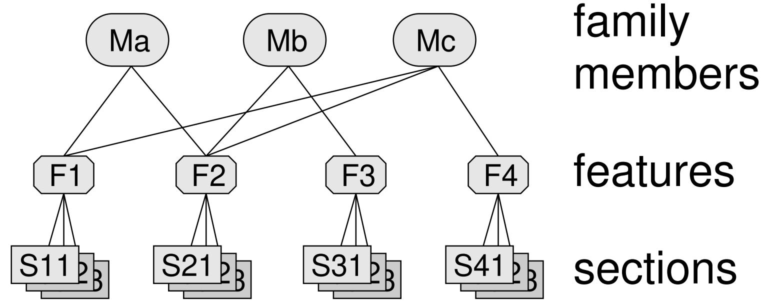
the chapters of the requirements document

1. Introduction
2. `feature` UserSpace
3. `feature` BasicConnection
4. `feature` VoiceChannel
5. `familymember` SpecificationA
6. `feature` ScreeningBase
7. `feature` BlackListOfDevices
8. `familymember` SpecificationB
9. `feature` BlackListOfUsers
10. `feature` FollowHumanConnectionForwarding
11. `familymember` SpecificationC
12. `feature` TransferUserRoleToAnotherHuman
13. `familymember` SpecificationD

⋮ ⋮

Indices / Bibliography

structure of a family



Case Study

“`feature`”, “`familymember`”

- these are **formal keywords** in extension

⇒ **The Base Telephone Switching System**

The Base Telephone Switching System

`section` **HumanEss** **[Humans]**

parents

. . .

HumanEss

. . .

$humans : \mathbb{F}\, humanIds$

`section` **UserEss** **[User Roles]**

parents HumanEss

. . .

UserEss

`inherit` HumanEss

. . .

$users : \mathbb{F}\, userIds$

$userRoles : humanIds \rightarrow \mathbb{F}\, userIds$

`default_section` **DevAssocNoRem** **[Telephone Device Associations neither Added nor Removed]**

DevAssocNoRem

`inherit` HumanEss, DeviceEss, . . .

$\forall\, h : humans \bullet$

$\quad possessedDevices(h) = initialPossessedDevices(h)$

The Base Telephone Switching System

“HumanEss”	introduces the set of humans
“UserEss”	introduces the set of user roles
“DeviceEss” (not shown)	introduces the set of telephone devices
“DevAssocNoRem”	**relation** humans–devices is **static** (POTS)

“parents”

- from **Z standard**
- documents the **dependences of sections** **(important!)**

“`default_section`”

- new formal keyword denotes: section is **changeable**
- keyword “`section`” denotes: section is **essential**:

  distinguish **core of feature** **(important!)**

⇒ **A Black List Feature**

A Black List Feature

`feature` **BlackListOfUsers**

...

`remove` ConNoScreen

`default_section` **BlackListUsrOnly** **[User Black List is the only Screening Feature]**

BlackListUsrOnly

`inherit` ..., ConScreenEss

$$\forall\, u, v : \mathsf{users} \bullet$$
$$((\mathsf{connectionScreen}(u, v) = \mathsf{connectionScreenBlock})$$
$$\Leftrightarrow ((u, v) \in \mathsf{blackListUsr}))$$

A Black List Feature

- **protects from calls** of users on a black list

methodology

- **split screening** into base and variants
  - ScreeningBase (not shown) + BlackListOfUsers

"`remove`"

- **non-monotonous change**: removes section of another feature

⇒ **A Call Forwarding Feature**

A Call Forwarding Feature

`feature` **FollowHumanConnectionForwarding**

...

 `remove` DevAssocNoRem

A Call Forwarding Feature

- **allows** a human **to move**
  to another telephone device and register himself there
- mainly **just relax** the constraint of DevAssocNoRem above

$\Rightarrow$ **Composition of a Complete System**

Composition of a Complete System

`familymember` **SpecificationC**

`features` $\ldots$, BlackListOfUsers, FollowHumanConnectionForwarding

Composition of a Complete System

- a family member is **just a composition** of features
- **details:** in the paper

$\Rightarrow$ **Avoiding Feature Interactions**

Avoiding Feature Interactions

example

- **introduced** three **notions explicitly**
- **consequences:**
  - black list above:
    screens user roles, not devices
  - another black list feature:
    screens devices, not user roles
  - also two kinds of call forwarding
- **feature interaction** from intro **does not occur** anymore

Avoiding Feature Interactions

introduced notions explicitly

- **“telephone device”**
- **“human”**
- **“user role”**

also two kinds of call forwarding

1. **register human** with another **device**
2. **transfer** a **user role** to another human

feature interaction from intro does not occur anymore

- it is **clear whether** phone should **ring**
- **no ambiguous** notion of **“caller”** anymore

$\Rightarrow$ **Detecting and Resolving Feature Interactions**

Detecting and Resolving Feature Interactions

- case study: no interactions between TCS and CF

example for remaining problems

- both screening features "remove" the same section
- **type rules: warning!**
- manual **inspection:** contradiction
- **resolution: another feature**

Detecting and Resolving Feature Interactions

manual inspection

- **both** features have a **changeable section** that states that **this is the only** screening feature
  $\rightarrow$ contradiction

resolution: another feature

- that **removes both** sections
- that **adds another** section **with** the **correct** statement

$\Rightarrow$ **The Tool genFamMem 2.0**

The Tool genFamMem 2.0

- **extracts specifications** in plain CSP-OZ from a family document,
- **detects feature interactions** by
  - **additional type checks** for families
  - **heuristic warnings**
- helps avoiding feature interactions by
  generating documentation on the structure of the family.

- available freely

The Tool genFamMem 2.0

additional type checks

- like the **constraints on** normal, default, (and collect) **sections,** on “remove”, (**and** on the “default_properties” / “inherit”) **operators**

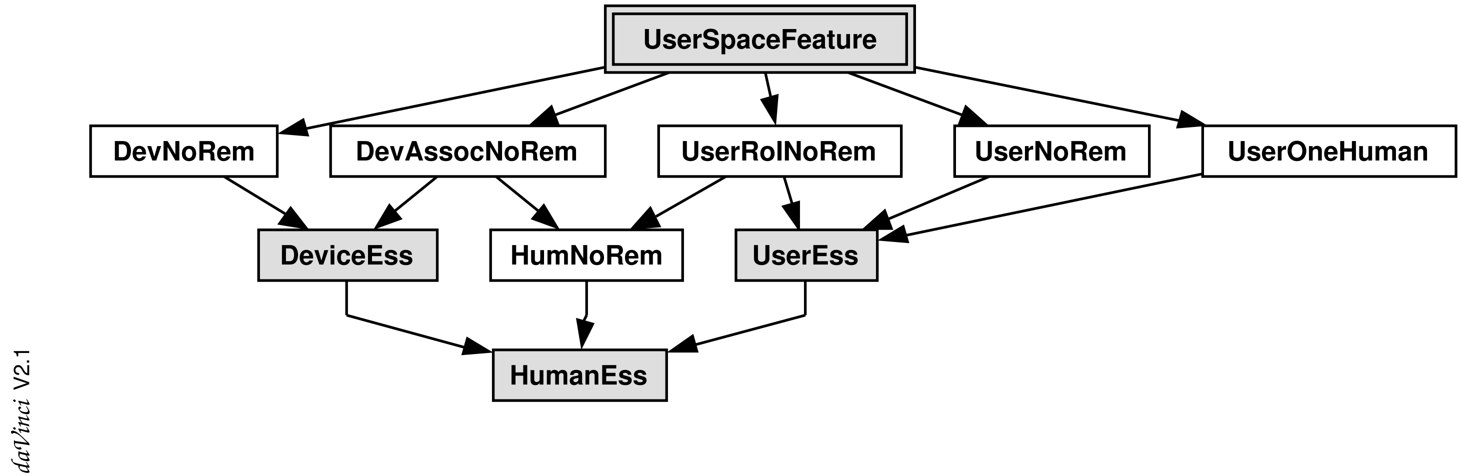
heuristic warnings

- for example when two different remove clauses **remove the same class**

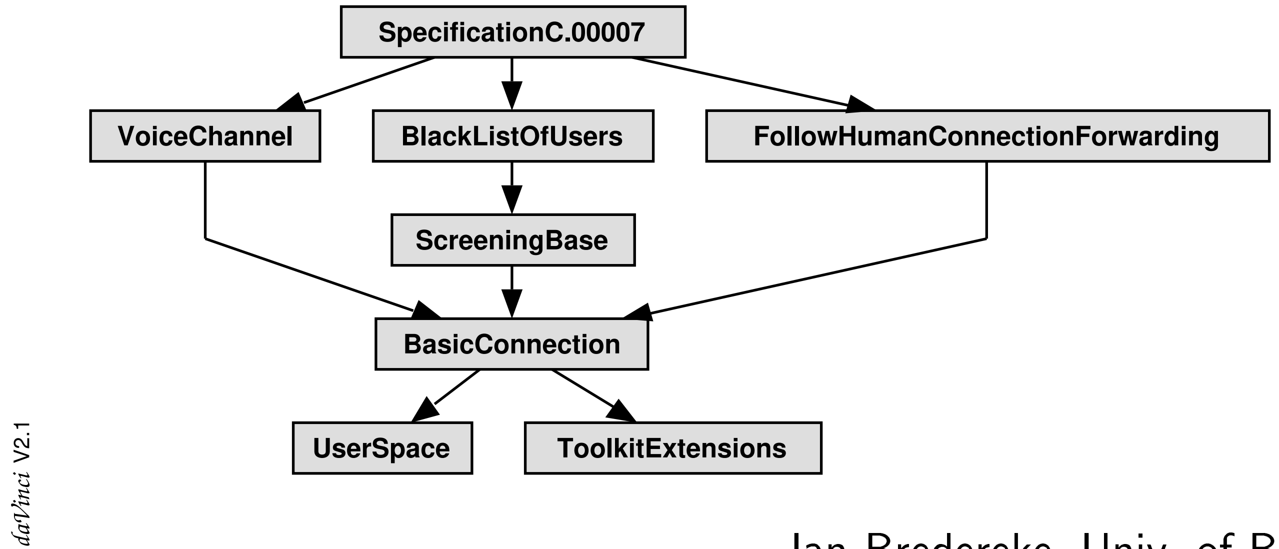
⇒ **Generated Documentation: the Uses-Hierarchy**

Generated Documentation: the Uses-Hierarchy

the sections of feature UserSpace



family member SpecificationC, abstracted to features



Generated Documentation: the Uses-Hierarchy

boxes

plain grey	**essential** section
white	**changeable** section
(double-border)	(collect section)

⇒ **Summary**

Summary

maintenance of a family of formal requirements

- **methodology:** original specifier of a feature formally **documents**
  - **what is core** of the feature
  - **dependences** on other features
- **formal feature combination mechanism** for CSP-OZ
  - some feature interactions (FI) appear as **type errors**
- **tool** genFamMem 2.0
  - **generates requirements** specifications from a family document
  - **detects FI** (type errors; heuristic warnings)
  - generates documentation (helps **avoiding FI**)
- **case study**

Summary

maintenance of a family . . .

- $\boxed{\boxed{\textbf{family}}}$ is important: **keep** all **variants together**

further research

- more experience – **extend case study** further
- apply to **other formalisms** than CSP-OZ
  - **necessary:** constraint-oriented specification style and incremental refinement
  - **already supported:** $\mathrm{CSP_Z}$ and plain Z
- investigate **relationship: families** of requirements – **families** of programs for reuse-oriented implementation process

$\Rightarrow$ **Paper on This Work**

Paper on This Work

[1] Jan Bredereke. A tool for generating specifications from a family of formal requirements. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, "Formal Techniques for Networked and Distributed Systems", pages 319–334. Kluwer Academic Publishers (August 2001).