Feature Orientation Considered Harmful?

Jan Bredereke

University of Bremen

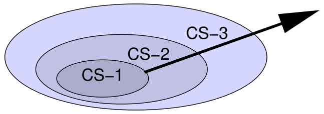
Overview

1. What are features?
2. What problems do we have with features?
3. How can we solve the problems we have with features?
4. Do we still have feature orientation afterwards?

1. What Are Features?

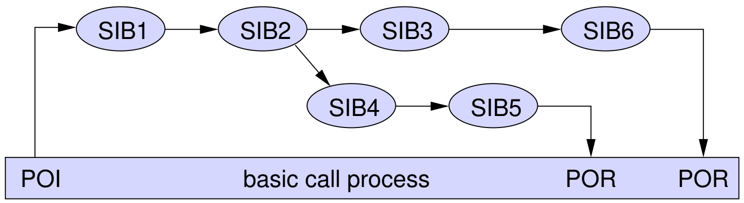
Intelligent Network (IN)

- extension of telephone switching systems
- general goals:
  - rapid introduction of new services
  - broaden range of services
  - multi-vendor environment
  - evolve from (all) existing networks
- standardized by ITU-T
- approach: base service & additional services/features
- new services step by step:



Global Functional Plane

- service independent building blocks (SIBs)
- service logic ("glue" for SIBs)
- basic call process
  - is special SIB
  - POI: point of initiation (of service)
  - POR: point of return



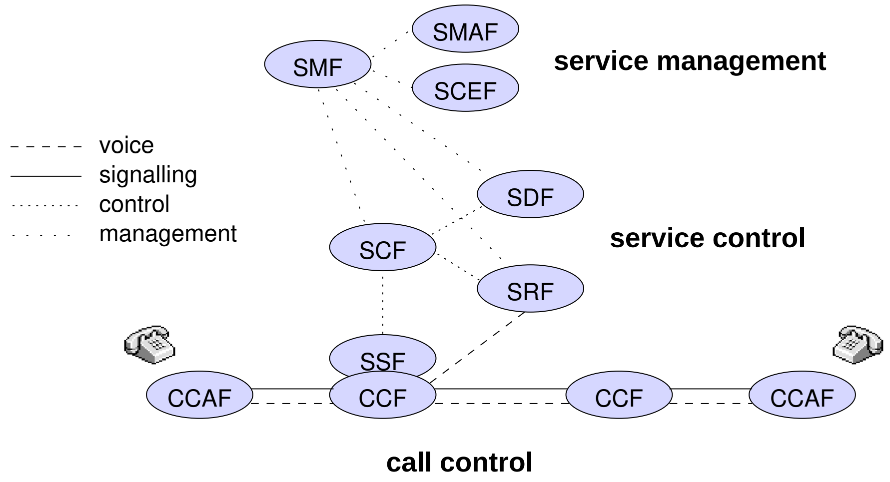
Features in IN CS-1

- Abbreviated dialling
- Attendant
- Authentication
- Authorization code
- Automatic call back
- Call distribution
- Call forwarding
- Call forwarding on BY/DA
- Call gapping
- Call hold with announcement
- Call limiter
- Call logging
- Call queueing
- Call transfer
- Call waiting
- Closed user group
- Consulation calling
- Customer profile management
- Customized recorded announcement
- Customized ringing
- Destinating user prompter
- Follow-me diversion
- Mass calling
- Meet-me conference
- Multi-way calling
- Off net access
- Off net calling
- One number

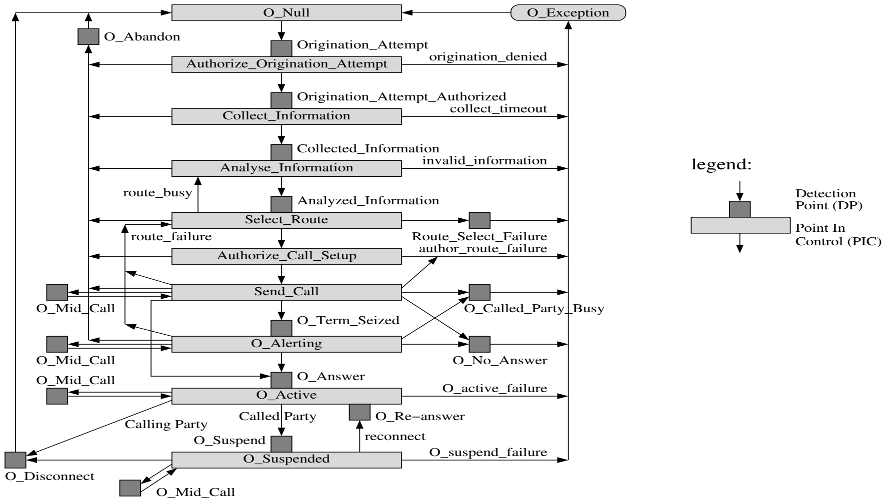
- Origin dependent routing
- Originating call screening
- Originating user prompter
- Personal numbering
- Premium charging
- Private numbering plan
- Reverse charging
- Split charging
- Terminating call screening
- Time dependent routing

- 38 features

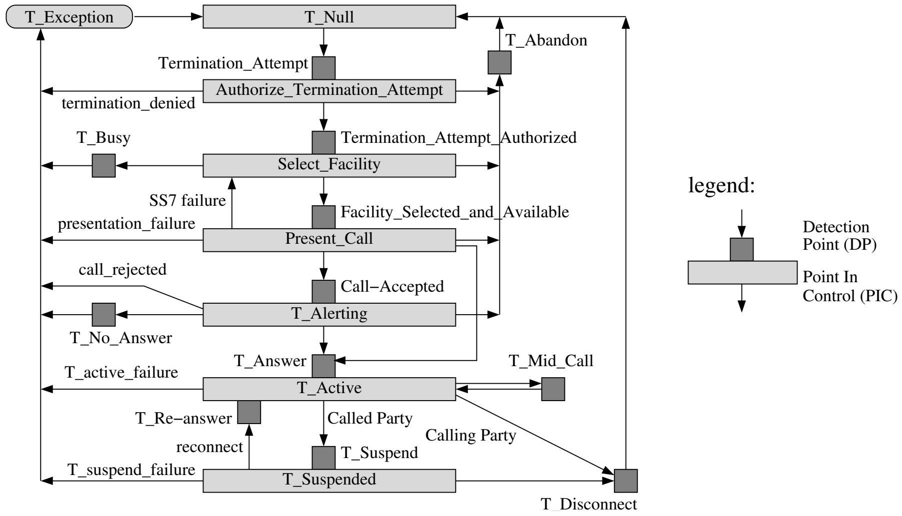
Architecture of Distributed Functional Plane



Originating Basic Call State Model of IN-CS2



Terminating Basic Call State Model of IN-CS2



Feature-Oriented Description in Telephone Switching

- base description plus separate feature descriptions
- attraction: behavioural "modularity"
  - easy change of system behaviour
  - make *any* change by just adding a new feature description
  - never change existing descriptions
- emphasizes individual features
  - makes them explicit
- de-emphasizes feature interactions
  - makes them implicit in the feature composition operator

2. What Problems Do We Have With Features?

Feature Interaction Problems in Telephone Switching

- *features work separately, but not together*
  - hundreds of (proprietary) features
  - combinations cannot be checked anymore
- telephone switching
  - users' expectation high
- feature
  - about any increment of functionality

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."

Call Waiting & Call Forward on Busy

- both activated simultaneously
  - in busy state
  - when another call arrives
- only one can get control
  - no resolution, except restrictions on features

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

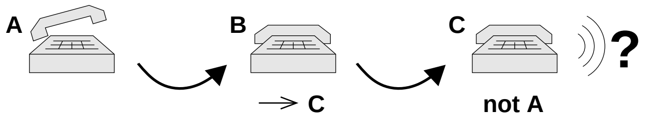
- switch may restrict no. of queries
  - protection against infinite loops
  - e.g., one query per call
  - → OCS subscription prevents orders for pizza
- solution: one more query??

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

Call Forwarding & Terminating Call Screening

- CF
  - B forwards all calls to C
- TCS
  - when A is caller, C blocks him
- A calls B: can/should A reach C?



- notion of "caller" is crucial

Informal Feature Interaction Definition in Literature

- *FI:*
  the behaviour of a feature is changed by another feature
- not precisely clear what a feature actually is
- not all interactions are undesired

Categorization of Causes

according to Cameron et. al. [CGL+94]:

- violation of feature assumptions
  - naming
  - data availability
  - administrative domain
  - call control
  - signalling protocol
- limitations on network support
  - limited CPE signalling capabilities
  - limited functionalities for communications among network components

- intrinsic problems in distributed systems
  - resource contention
  - personalized instantiation
  - timing and race conditions
  - distributed support of features
  - non-atomic operations

Approaches for Tackling FI

- ignore
- informal
  - filtering
  - heuristics
  - . . .
- formal methods
  - validation of:
    - ▷ specified properties of the features
    - ▷ general properties of the system (free of non-determinism, . . .)

- new architectures
  - IN
  - Tina, Race, Acts
  - DFC, agents
- better software engineering processes

- in practice: ignore / informal / processes / (architectures)
- formal analysis?
  yes, but. . .
  - formalization is huge task
  - complexity not amenable to tools
    - ▷ "spaghetti code" dependences

Feature Interactions in the Requirements

- if requirements complete,
  all FI are (inherently) present in the requirements

Requirements Structuring Problems

- monolithic requirements or single layer of extension
  - ISDN: monolithic
  - IN: no features on top of features
  - CF & TCS: resolution needs extended, common notion of caller
  - CF & OCS: resolution needs extended, common notion of called user

- new services depend implicitly on new concepts
  - some new concepts:
    - ▷ conditional call setup blocking
    - ▷ dialled number translation
    - ▷ multi-party call/session
      - required for CF & TCS and for CF & OCS
    - ▷ service session without communication session
    - ▷ distinction user – terminal device
    - ▷ distinction user – subscriber
    - ▷ mobility of users and of terminals
      - difficult to specify with network of distributed switches
    - ▷ multiple service providers, billing separately

- concerns of the users' interface are spread out
  - several features assume exclusive access to the user's terminal device (12 buttons + hook)
  - example: calling card & voice mail

3. How Can We Solve the Problems We Have With Features?

Needed: a More Modular Requirements Structure

- centralize responsibility for the users' interface
- a layered architecture
  - like in computer communication systems

New Architectures

- current: IN
  - currently largest impact on implementations
    - ▷ see above
  - Jain
    - ▷ enhanced IN-like architecture
    - ▷ developed currently
    - ▷ in Java
    - ▷ allows multi-party, multi-media calls
    - ▷ Java Call Control (JCC):
      call state machine similar to that of the IN
    - ▷ JCC does not handle feature interactions

- future: Tina, Race, and Acts
  - Tina
    - ▷ radical approach: entirely new architecture
    - ▷ strongly based on Open Distributed Processing (ODP) and Corba
    - ▷ migration difficult
  - Race project
    - ▷ Cassiopeia
      - developed open services architectural framework (Osa)
      - many commonalities with Tina
      - focuses on requirements engineering of services
      - tries to take legacy services into account
    - ▷ Score
      - concerned with the methodological aspects of service creation
      - detection of undesired service interactions:
        formal methods, exhaustive simulation
        applied to small example

- Acts project
  - ▷ followed Race project
  - ▷ application and on evaluation of service architectures
  - ▷ result: a modified architecture

- research: the DFC and the agent architecture
  - Distributed Feature Composition (DFC)
    - ▷ compose features in a pipe-and-filter network
    - ▷ designed to be implementable on a conventional switch
    - ▷ some new concepts supported, others not
    - ▷ no layered architecture
    - ▷ implemented in AT&T's Eclipse project, which additionally incorporates Voice Over IP
  - Zibman et. al.'s agent architecture [ZWO+96, ZWO+95]
    - ▷ separates several concerns explicitly
    - ▷ restricts itself to narrow-band telephony over a fixed network
    - ▷ Plain Old Telephone Service is represented by a single service agent

Discussion of New Architectures

- IN important step, but not sufficient
- Tina, Race, Acts have most of the interesting concepts, but transition is very expensive
- feature interaction detection is still research

- some undesired service interactions still possible in new architectures
  - Kolberg and Magill checked the FI benchmark for Tina [KoMa98]
  - still possible:
    - ▷ forwarding loop
    - ▷ automatic callback & automatic re-call
    - ▷ calling number delivery & calling number delivery blocking
    - ▷ billing problems for video conference
    - ▷ . . .
  - causes: violated assumptions or conflicting goals
- *how to prepare for unanticipated changes??*
  - at least encapsulate as much as possible

Information Hiding Module

- module:
  - a work assignment
- criteria for designing modules:
  - identify the design decisions that are likely to change
  - have a module for each
- secret of a module:
  - a design decision that might change
- interface between modules:
  - the *assumptions* that they make about each other

Feature-Oriented Descriptions and Common Abstractions

- a module needs a common abstraction/assumption
  - module: now in the information hiding sense
  - common abstraction/assumption: true for all implementations
- a common abstraction/assumption needs a limited domain
- rapid innovation, legacy systems, too many players: hard to limit the domain
- *without domain limits: no common abstraction*

Performing Incremental Specification Formally

- standard means:
  stepwise refinement
- step:
  1. extend behaviour *or* 2. impose constraints
  - example 1.: add another potential event to a state
  - example 2.: specify the order of two events
- interesting properties preserved by step
  - example 1.: all old events remain possible
    - ▷ no deadlock in this state
  - example 2.: no harmful event added
    - ▷ all safety properties preserved

Non-Monotonous Changes

- telephone switching:
  new features change the behaviour
  - of base system, or
  - of other features
- example: call forwarding
  - stops to connect to dialled number
    - ▷ restricts base system behaviour

    and
  - starts connecting to forwarded-to number
    - ▷ extends base system behaviour

Formal Support for Feature Specification

- considerable research effort
  on feature composition operators
- FIREworks project
  (Feature Interactions in Requirements Engineering)
  - various feature operators proposed and investigated
- "feature-oriented programming"
- based on the superimposition idea by Katz
- reflects practice of arbitrary changes successfully
- analytical complexity:
  too big for tools for real systems

Superimposition

- by Katz [Kat93]
- approach:
  - base system
  - textual increments
  - composition operator
- problem:
  - increments have defined interface,
    base system has not
  - increment can invalidate arbitrary assumptions about base system

Families of CSP-OZ Specifications

key ideas:

- *change entire assumptions only*
  - constraint-oriented specification
  - constraint = assumption
- *maintain all variants together*
  - generate specific member automatically as necessary
- *document information needed for changes*
  - dependence of requirements
  - what is the core of a feature

Constraint-Oriented Specification

- features closely interrelated
  - most refer to mode of connection
  - user interface: few, shared lexical events
    - ▷ system cannot be sliced by controlled events
- incrementally impose partial, self-contained constraints
- composition by logical conjunction

Case Study on Telephone Switching Requirements

- black box specification of telephone switching
- attempt to incorporate new concepts

Grouping Classes into Features

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

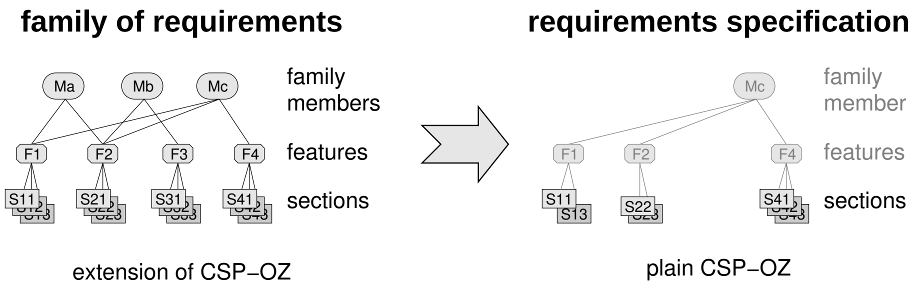
⋮ ⋮

Indices / Bibliography

The Feature Construct

- feature UserSpace
- feature BasicConnection
- familymember SpecificationB

Generating Family Members From a Family Document



Result of Family Member Generation

- family member composition chapter: part replaced

Controlled Non-Monotonous Changes

- feature ScreeningBase
- feature BlackListOfUsers
- feature FollowHumanConnectionForwarding
- familymember SpecificationC

Avoiding Feature Interactions

- introduced three notions explicitly
  - "telephone device"
  - "human"
  - "user role"
- consequences:
  - black list above: screens user roles, not devices
  - another black list feature: screens devices, not user roles
  - also two kinds of call forwarding
- no feature interaction screening–forwarding anymore

Detecting Feature Interactions by Type Checks

- *type rules*: part of the family extension of CSP-OZ
- 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

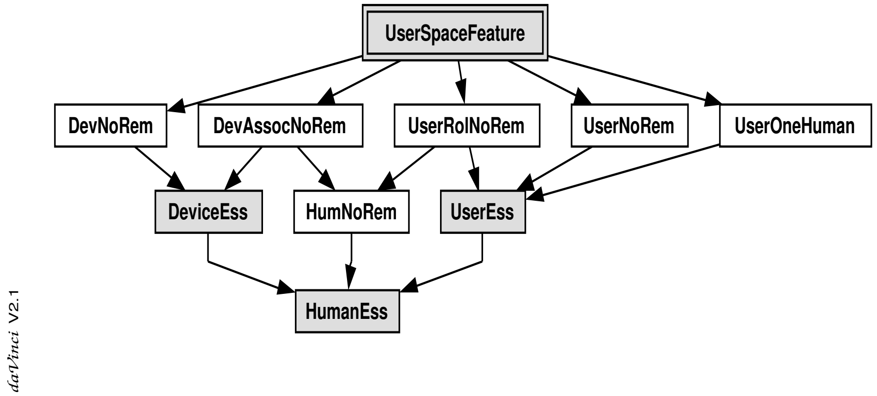
Feature Interactions Detected in Case Study

- no interactions between TCS and CF
  - no type errors detectable
- but other problems problems present:
  - both screening features “remove” the same section
  - type rules: warning!
  - manual inspection: contradiction
- resolution: another feature

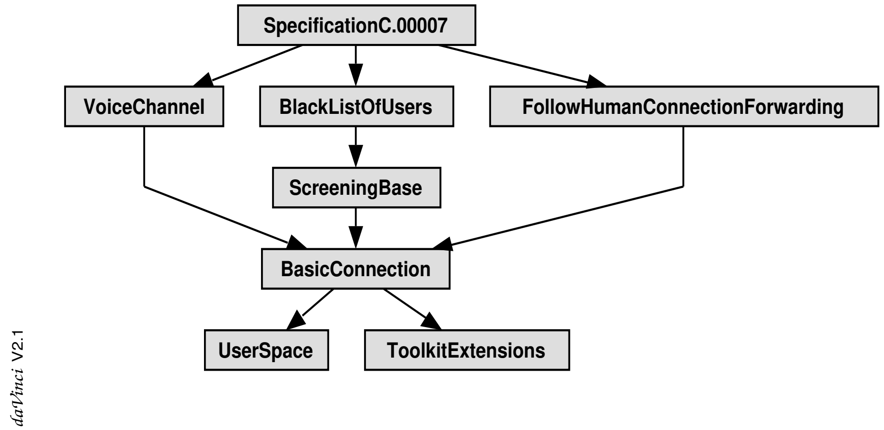
Documenting Dependences

- uses-relation for requirements:
  - use of previous definition
  - reliance on previous constraint
- documented by:
  - Z’s section “parents” construct
  - class inheritance (mapped to Z sections)
- if no relationship: identifiers out of scope

Sections of Feature UserSpace



Hierarchy of Features of SpecificationC



Hierarchical Requirements Specification

- a feature can build on other features
- in contrast to the Intelligent Network
- possible to have feature providing a common base

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

Further Tools

- cspozTC
  - type checker for CSP-OZ
- daVinci
  - visualizes uses hierarchy graphs

Semantics of CSP-OZ Extension

- formal definition of language extension in [Bre00b]
  - understand details: need to know Object-Z and CSP

4. Do We Still Have Feature Orientation Afterwards?

Feature Orientation Considered Harmful?

my claims:

- ignoring feature interactions does not work
- formal analysis on "spaghetti" dependences does not scale
- information hiding modules reduce dependences

caveats:

- legacy systems: hard to restructure
- difficult: prediction of change / a limit on the change

5. References

References

[Bre00a] Bredereke, J. *Families of formal requirements in telephone switching*. In Calder, M. and Magill, E., editors, "Feature Interactions in Telecommunications and Software Systems VI", pp. 257–273, Amsterdam (May 2000). IOS Press.

[Bre00b] Bredereke, J. *genFamMem 2.0 Manual – a Specification Generator and Type Checker for Families of Formal Requirements*. University of Bremen (Oct. 2000). URL http://www.tzi.de/~brederek/genFamMem/.

[Bre01] Bredereke, J. *A tool for generating specifications from a family of formal requirements*. In Kim, M., Chin, B., Kang, S., and Lee, D., editors, "Formal Techniques for Networked and Distributed Systems", pp. 319–334. Kluwer Academic Publishers (Aug. 2001).

[Bre02] Bredereke, J. *Maintaining telephone switching software requirements*. IEEE Commun. Mag. **40**(11), 104–109 (Nov. 2002).

[CGL+94] Cameron, E. J., Griffeth, N. D., Lin, Y.-J., et al.. *A feature interaction benchmark in IN and beyond*. In Bouma, L. G. and Velthuijsen, H., editors, "Feature Interactions in Telecommunications Systems", pp. 1–23, Amsterdam (1994). IOS Press.

[Kat93] Katz, S. *A superimposition control construct for distributed systems*. ACM Trans. Prog. Lang. Syst. **15**(2), 337–356 (Apr. 1993).

[KoMa98] Kolberg, M. and Magill, E. H. *Service and feature interactions in TINA*. In Kimbler, K. and

Bouma, L. G., editors, "Feature Interactions in Telecommunications and Software Systems V", pp. 78–84, Amsterdam (Sept. 1998). IOS Press.

[Zav01] Zave, P. *Requirements for evolving systems: A telecommunications perspective*. In "5th IEEE Int'l Symposium on Requirements Engineering", pp. 2–9. IEEE Computer Society Press (2001).

[ZWO+95] Zibman, I., Woolf, C., O'Reilly, P., Strickland, L., Willis, D., and Visser, J. *Minimizing feature interactions: an architecture and processing model approach*. In Cheng, K. E. and Ohta, T., editors, "Feature Interactions in Telecommunications III", pp. 65–83. IOS Press, Amsterdam (1995).

[ZWO+96] Zibman, I., Woolf, C., O'Reilly, P., Strickland, L., Willis, D., and Visser, J. *An architectural approach to minimizing feature interactions in telecommunications*. IEEE/ACM Trans. on Networking **4**(4), 582–596 (Aug. 1996).