Maintaining Telephone Switching Software Requirements*

Jan Bredereke
Universität Bremen, TZi
P.O. box 33 04 40, D-28334 Bremen, Germany
brederek@tzi.de, www.tzi.de/~brederek

Abstract

We discuss how telephony software requirements should be structured to reduce maintenance costs. "Feature interaction" problems have become a serious obstacle to add more features to telephone switches. We show why existing public switched telephone networks are hard to modify. We then argue that a more modular, layered requirements architecture can help. Finally we survey approaches for such an improved telephone switching architecture and discuss them.

1 Introduction

During the lifetime of a successful software product, maintenance costs more than anything else. Maintenance means to revise and enhance software systems, maintaining (or improving) their conceptual integrity and keeping documents complete and accurate.

Currently, telephone switching software must be changed substantially. Reasons are the opening of the world telecommunications markets (which follows a similar development in the USA), cheaper broadband transmission hardware, faster switch processing hardware, the advent of wireless mobility, and the probable advent of mass Internet telephony. Competition forces the providers to add more and more new services into the switches. The intervals of changes become shorter and shorter.

Changing telephone switching systems is difficult because of their complexity. They already comprise hundreds of features, and they rank among the largest software systems in the world. Recently, so-called *feature and service interaction problems* have become a serious obstacle to adding more features and services to these systems [1]. The awareness of these problems arose starting from the end of the

*This work was partially funded by the Leibniz Program of the German Research Council (DFG) under grant Ol 98/1-1.

1980s. Since 1992 there has been a series of six Feature Interaction Workshops [2, 3], where industry and academia together tried to tackle the problems.

According to a commonly used definition, "a *feature interaction* occurs when the behaviour of one feature is altered by the use of another", in particular in an undesired way. In this definition, a "feature" can be, more or less, about every change to the system imaginable. This can range from the introduction of Answer Call over Number Portability to just increasing the number of trunks that the switch can handle. In this article, we use "service" as a synonym for "feature".

Several classes of causes for service interactions have been identified, and service interaction categorisations have been proposed. Among others, Cameron et al. [4] present a categorisation by *causes* of interactions. Their top-level categories are: violation of service assumptions, limitations on network support, and intrinsic problems in distributed systems.

We claim that the service interaction problems are a collection of separate problems from different domains; one of these domains is the topic of our article. Even though many people use the term "service interaction *problem*", we therefore prefer the term in the plural, "problem*s*".

The domain we will discuss is a software requirements structure suitable for maintenance. Service interactions often already arise in the requirements documents. When these documents are a *complete* description of the behaviours and of other interesting properties, then *all* service interaction problems are present in the requirements documents at least inherently. Therefore, they should be tackled already there. (Of course, we still have to ensure that any implementation conforms to its requirements. But this is a different question from that of the interaction of services which work perfectly well on their own.) The information in a requirements document must be structured suitably to make it manageable. Having a complete requirements document alone is not sufficient by far to tackle service interaction problems.

In Section 2, we show why existing telephone switching software requirements are hard to modify; and in Section 3, we discuss solutions. In Section 4, we survey approaches for improved telephone switching architectures. Section 5 summarizes the survey and discusses these approaches.

2 Requirements Structuring Problems

In this section, we look at the structure of telephone switching requirements and identify three problems which make the requirements hard to change consistently.

2.1 Monolithic Requirements or Single Layer of Extension

Existing PSTNs (Public Switched Telephone Networks) are hard to modify, because at the upper, application oriented layers, one monolithic service provides lots of separate functionalities. The IN (Intelligent Network) [5] is an attempt to solve

Table 1: some new concepts in telephone switching.

conditional call setup blocking
dialled number translation
multi-party call/session
service session without communication session
distinction user – terminal device
distinction user – subscriber
mobility of users and of terminals
multiple service providers, billing separately

this problem, by defining an extension interface for POTS (Plain Old Telephone Service).

Unfortunately the IN does not allow services to be created on top of other services. Services must be independent of each other. This is because all new services must be built on the POIs (Points of Invocation) and PORs (Points of Return) of the Basic Call Process.

Independent services have the advantage that they can be developed independently. But if two services, in some sense, overlap, undesirable interactions may occur. CF (Call Forwarding) allows calls to one telephone to be forwarded to another. OCS (Originating Call Screening) allows particular numbers to be blocked; for example parents can restrict the phone access of their children. When both services are available, an unexpected service interaction can happen; a youth might circumvent OCS by programming the desired number as the call forwarding target, and then calling his own number. The call will be forwarded to the target and will defeat the screening if OCS is processed before and independently of CF.

The interaction problem cannot be solved by simply reversing the order of processing. The user might have blocked all long-distance calls using OCS. Later, he attempts to forward all his calls to a friend which he will visit and who lives in a neighbour town. When CF is processed before and independently of OCS, all incoming calls will be blocked instead of forwarded.

The notion of a called user is extended and thus changed by CF; all services that rely on this notion must be defined on top of CF in order to solve the interaction problem. They must use the extended notion of a called user.

2.2 New Services Depend Implicitly on New Concepts

Undesired service interactions can happen, because fundamentally new concepts often are introduced only implicitly. Table 1 lists some new concepts. It is difficult to understand a new concept fully while it is used by only one service. This makes it hard to express the concept explicitly. Also, a new concept may need a new layer in the architecture, which cannot be added explicitly to a non-layered architecture such as the IN.

One such new, implicit concept is the n-party session. It is a generalisation of the "call", which always has two end points. It is necessary for such services as Consultation Call, Conference Call, Call Forwarding, and Universal Personal Telecommunications. The latter allows its users to register with any terminal device and then have all their calls and services available there. This new concept is not explicit in the terminology or architecture of current switching requirements. In the IN, the Call Forwarding service is expressed by two conventional calls glued together in the middle. The above interactions between OCS and CF occur because the CF service changes the semantics of a call implicitly, which invalidates assumptions of the OCS service about a call.

Another such new, implicit concept is terminal mobility. More than 50% of the terminal devices are now mobile in some countries. The requirements architecture of any current telephone switching network does not allow to add mobility easily. This is so even though mobility doesn't change the set of requirements much from a strictly user-oriented point of view. The user-oriented view has one big node which represents the network, with many terminal devices attached. Mobility means that there is no piece of wire anymore. Any current requirements architecture represents the network by a set of interconnected nodes. Each represents a switch. Terminal devices can be associated to one such switch only. A description of terminal mobility therefore needs complex hand-over procedures between nodes. The description of the switching network as a set of interconnected nodes had good reasons. But it now prevents us from describing terminal mobility requirements "naturally" from the user-oriented point of view. It prevents us from encapsulating mobility in the design architecture into a separate description of how mobile terminal devices and base stations inter-work.

2.3 Concerns of the Users' Interface Are Spread Out

The requirements documents are difficult to maintain if the requirements for one concern are spread out far through the documents; the users' interface is such a concern. There is no systematic coordination among the services of the IN about the use of the physical signals at a terminal device. A terminal device often has only few syntactic signals available: twelve buttons, a hook switch, and a few signal tones. Many services are available currently. Together they require a large number of signals, many more than are available physically on most terminal devices. Physical signals must be reused in different modes of operation. But the definitions of several services implicitly assume exclusive access to the user's terminal device. When this assumption is violated, undesired service interactions can, and frequently do, occur.

The service interaction between a Credit Card Call (CCC) service and a Voice Mail service is an example. A Credit Card Call begins with an authorisation phase. The user must enter the card's number and PIN. For convenience, the service often allows to make another call without entering this long sequence again. She just presses the "#" button at the end of the first call. The user of a Voice Mail service can access her voice mail messages. If she does it from a phone other than her own, she must authorize herself, too. She calls her own phone number and then

presses the "#" button followed by some identification. She may want to access the voice messages using the CCC service. But the services cannot work together. The telephone system has no way to determine whether the caller presses the "#" button because she wants to terminate the call without leaving a message, or whether she is the owner of the voice mail box and wants to authenticate herself.

This service interaction can be resolved by adding another physical signal. In the calling card service to which the author once subscribed, the "#" button must be pressed at least two seconds to take effect. Unfortunately, the existing terminal devices do not allow additional physical signals in a number sufficient for all services.

3 Solution: A More Modular Requirements Structure

The use of modularization in the information hiding sense is one way of preparing for future changes. This means that we identify different concerns and separate them as much as possible. The idea is well known for the design phase, but it can be applied to software requirements, too. In the design phase, one defines an information hiding module for each concern with narrow, precisely documented interfaces between them. Each module hides a "secret", i.e., one implementation decision. Whenever a secret changes, only one module is affected. When we look at telephone switching software *requirements*, we claim that the principle is not applied sufficiently.

The responsibility for the users' interface should be centralised. A few, dedicated requirements documents should describe it. Their maintenance must be coordinated by a single organisational unit. There must be only one such unit within a provider. For some aspects, the unit should be attached even to a standardisation body. In the area of human-computer interfaces (HCI), a centralized user interface is already standard operating procedure. There are good libraries for graphical user interfaces (GUIs). Application requirements are written on the semantic level only ("select file"), never on the syntactic level ("mouse click"). In contrast, the requirements for most telephone services contain users' interface concerns. They are written in such terms as "on-hook" and "flash-hook". Examples are the assignments of the first and the second feature interaction detection contest at FIW'98 [3] and FIW'00 [2]. These assignments are accessible, of manageable size, and written by people from industry.

In [6], we propose such a centralized approach. It encapsulates the syntactic details of the signal-poor current users' interface and provides a sufficient number of semantic signals to other modules. We also sketch an application to the Intelligent Network.

A prerequisite for encapsulating the users' interface is a requirements architecture that supports information hiding modules, such as a layered architecture. In a layered architecture, each layer solves a partial problem and hides its details from the other layers. Computer communication systems today are usually designed

in this way. The upper layers of the Internet protocols provide many specialised services, such as HTTP, SMTP, FTP, Telnet, and many more. Changes to one of these services do not affect the others. Nevertheless, some application protocols, which implement their respective service, build on other application services. The PSTN has a similar structure for the lower communication layers; an example is the architecture of the widely used Signalling System No. 7. In contrast, the upper, application oriented part of the PSTN is a single monolithic block. We think that the Internet way of adding services will reduce service interactions, compared to the PSTN and Intelligent Network way.

But first the idea of modularizing the requirements must be accepted more widely; only then the architectural models can be improved. The structure of the assignments for the feature interaction detection contest shows this.

A modular, layered architecture helps to avoid implicit, undocumented assumptions. Layers of information hiding modules are connected by explicit interfaces. Explicit interfaces document the assumptions on modules. Implicit, undocumented assumptions can become invalid by new services. This is one of the main causes of service interaction problems.

An architecture of fully layered services reduces the danger that the same new concept will be introduced by different services. Such a parallel introduction can lead to inconsistency. A layered architecture encourages to define a new concept much earlier. It is cheaper to add another layer than to restructure a monolithic system.

Nevertheless, adding or extending concepts still costs. When we extend the two-ended "call" to a n-party "session", and when we have a dedicated "connection resource" module, then the syntactic definition of its interface will show clearly which other modules use this module and which don't. Unfortunately, at least most of the connection-oriented services must be inspected and many of them must be revised. After this work is done, we can add more services like Call Forwarding, without further work or fear of interactions.

Integrating a new concept needs sufficient confidence in its quality and future success. Changing a software infrastructure of this size is extremely expensive. Nevertheless, integrating some new concepts eventually is inevitable. We think that the session concept now is understood sufficiently well. It therefore should become a first class member of switching requirements. In the following, we discuss new architectures that go this way.

4 Survey of New Architectures

We now sketch some approaches that propose an improved telephone switching architecture. These approaches are not concerned with software requirements only, but with a comprehensive software architecture. Nevertheless, here we are interested mostly in the requirements aspect.

4.1 Current: the Intelligent Network

The Intelligent Network (IN) [5] currently has the largest impact on implementations. This international standard by the ITU-T defines an interface to PSTNs where new services can be added, potentially created by third parties. The IN models POTS, possibly enhanced by any non-IN services, in an abstract way as a Basic Call Process (BCP). A number of processing states is identified at which the basic processing can be suspended in order to process an IN service. It is not intended to capture the structure of the entire basic call processing in the definition of the BCP. An IN service has one POI and one or more PORs, where basic call processing resumes. The invocation can depend on several kinds of trigger conditions. The structure of the Service Switching Function (SSF), which hands off control from the basic call processing to the IN call processing, also comprises a Feature Interaction Manager (FIM). Its task is to handle interactions among IN services and between IN and non-IN services. The ways in which this can be achieved are not part of the IN standard and are left to the implementers.

The IN architecture builds on POTS and therefore it is based on a call-oriented, two-party, narrow-band voice communication link between non-mobile terminal devices, billed by a single network provider. Many of the IN services independently lift one of these limitations.

Jain [7, 8] has an enhanced IN-like architecture. It is a standard developed currently. Jain offers a portable network interface to application services. This interface can be added on top of any kind of network (PSTN, wireless, Internet). It is written in the Java language. Its call model allows multi-party, multi-media calls. Jain's Java Call Control (JCC) has a call state machine similar to that of the IN, and also uses a similar trigger mechanism. JCC does not handle feature interactions. These must be managed at the application level, or by provisioning and management functions.

4.2 Future: Tina, Race and Acts

The Tina initiative (Telecommunication Information Network Architecture) [9, 10] follows a more radical approach. It was conducted by Tina-C, a world-wide consortium of network operators and telecommunications and computer suppliers. It has a long-term scope and defines an entirely new architecture, called Tina. The initiative took up concepts from Open Distributed Processing (ODP) and the Common Object Request Broker Architecture (Corba), and it applied and adapted them to the telecommunications domain.

The Race project (Research and technology development in Advanced Communications technologies in Europe) is a related large research initiative and has been conducted by the European Community. Cassiopeia was one of its many projects. It developed an open services architectural framework for integrated service engineering, called Osa [11]. Both the thrust of the project and the resulting architecture are quite close to Tina-C. Both have a service driven approach, and both emphasize the separation of concerns. Differences are that Tina-C explicitly aims to define a

software architecture, while Cassiopeia focuses more on requirements engineering of services. Tina defers the use of legacy services and their access to a later stage, while Cassiopeia sees them as an important part of the resource infrastructure, since they are always there and must be taken into account.

The commonalities are not surprising. Both Cassiopeia and Tina have been influenced by the previous Race project Rosa (Race Open Service Architecture), and both had some partners in common.

Another Race project worth mentioning here is Score (Service Creation in an Object-oriented Reuse Environment). It is not directly concerned with software architecture. But it is concerned with the methodological aspects of service creation. Part of that is the detection of undesired service interactions. Score applies formal methods to IN-like systems. The goals are to detect conflicting service requirements which are not satisfiable and to detect specified requirements which are not satisfied on an explicit behavioural model. The approach uses exhaustive simulation for these validations. It has been applied to small demonstration examples.

The Acts project (Advanced Communications Technologies & Services) [12] of the European Union follows the Race project. Five of its projects are concerned with service architectures. The emphasis is now on their application and on their evaluation in trials. The Difference project relates different service and service management architectures to each other. The Dolmen project extended and refined the Tina architecture (and also Cassiopeia's Osa), in particular in the area of mobility, since the earlier projects had assumed an underlying fixed network. The result is called Open Service Architecture for an integrated fixed and Mobile environment (Osam). The Vital project implemented and tested a specification of an Open Distributed Telecommunications Architecture (Odta), which is based on Tina. The Acts Insignia project added an IN interface to broadband ISDN [13]. This approach was extended by the Ibis project of the CoRiTeL laboratory. The Ibis project ran a Tina architecture over an underlying broadband ISDN by using the IN interface of the Insignia project [14]. The Acts Avanti project investigated user interfaces to a multi-media system that can adapt to different terminal hardware, network bandwidth, and user abilities (able-bodied/blind/motor-impaired). The user interface module of Avanti encapsulates the lexical and syntactic levels of user interaction; it makes them exchangeable for each user group.

Two Acts projects are concerned with service creation. The Screen project identified and documented what is known about tool supported methodology for service creation. It created a document on engineering practices for component-based service creation (Corba, Tina). Contrary to original plans, its service interaction work remained relatively general-purpose. The Tosca project is concerned with the integration of IN-based systems into Tina, It developed a service creation platform for both Tina and IN, with particular emphasis on IN–Tina inter-working. In the Tosca project, Kolberg and Magill also evaluated Tina with respect to service interactions. We will discuss the result below.

4.3 Research: the DFC and the Agent Architecture

More novel architectures have been proposed in the research literature.

The DFC (Distributed Feature Composition) virtual architecture is proposed by Jackson and Zave [15]. Its specific goal is to avoid and to detect service interaction problems. It allows to compose features in a pipe-and-filter network. Each feature is represented by one or more boxes, or filters. These are connected by simple two-way voice and signalling connections, or pipes. The features and the routing among them are separated. Feature boxes are composed dynamically by the DFC router when a *usage* demands it. The concept of the usage is related to the concept of the session.

Therefore, multi-party sessions are supported, even with a dynamic structure.

Tool support for detecting feature interaction problems is possible. The behaviour of the feature boxes is specified by a combination of formal description techniques. The features are encapsulated into loosely coupled boxes to make this analysis easier.

The architecture is called *virtual* because it does not assume any particular underlying physical architecture. Nevertheless, the authors consciously avoided choices that are likely to be expensive to implement.

There is no layered architecture; all features are specified independently. Features cannot build onto each other in an organised way.

The architecture is prepared for some new concepts of call related features, but not for others. It supports the multi-party call. But in the DFC's concept of the telephone number, there is no distinction among users, the different roles they play, and the different terminal devices they may use. The architecture has no specific provisions for billing or management. These issues need to be handled by the (call-related) feature boxes, too.

The users' interface is another concern which is spread out over the feature boxes. The signals are on a syntactic level, for example "flash hook".

The DFC architecture must have many of the problems that current switching architectures have because of one of its strong points: it was designed to be implementable on a conventional switch.

The DFC architecture is interesting because of its mechanisms for handling the problems when combining features. Recently, the DFC architecture has been implemented in AT&T's Eclipse project, which additionally incorporates Voice Over IP.

An Agent Architecture is outlined by Zibman et. al. [16]. It separates several concerns explicitly. There are four distinct types of agents: user agents, connection agents, resource agents, and service agents. This separates user and terminal concerns. The terminal resource agent encapsulates the user interface details, such as the signal syntax. The distinct user and connection agents separate call and connection concerns. The user agents bring the session concept with them. The connection agents coordinate multiple resource agents. The resource agent separates resource management from both session control and from the services.

The approach allows to detect service interaction problems only late, at run-time.

Table 2: some new concepts and how architectures cover them.

explicit concept available in basic architecture	IN	Tina, Race, Acts	DFC	Agent Architecture
conditional call setup blocking	✓	✓	✓	✓
dialled number translation	✓	✓	✓	✓
multi-party call/session	–	✓	✓	✓
service session without communication session	–	✓	✓	✓
distinction user – terminal device	–	✓	–	✓
distinction user – subscriber	–	✓	–	–
mobility of users and of terminals	–	✓	–	–
multiple service providers, billing separately	–	✓	–	–

Table 3: ideas for maintainability and how architectures cover them.

	IN	Tina, Race, Acts	DFC	Agent Architecture
multiple layers of services	–	✓	–	✓
most current concepts explicitly included (Table 2)	–	✓	–	–
encapsulated users' interface	–	✓	–	✓
mechanisms to detect service interactions off-line	–	(–)	✓	–
close to legacy systems	✓	–	✓	✓

Ambiguities are logged, for example ambiguities in precedence rules.

This architecture has some problems of the currently implemented architectures, too, because it was designed to work with them. It restricts itself to narrow-band telephony over a fixed network. And it was a design goal that the introduction of new services should not require modifications of existing software. Therefore POTS is represented by a single service agent even though POTS really comprises several distinct concerns.

5 Summary and Discussion

Table 2 summarizes how the new architectures cover our selection of new concepts from Table 1 above. Table 3 lists how the architectures cover the ideas for maintainability discussed.

The *IN* is an important step towards an open and extensible architecture, compared to the proprietary and monolithic architectures that existed before. But it can be only one step towards a maintainable architecture. Such an architecture must provide already more of the current concepts explicitly, and it must be more modular to allow adding further concepts.

The substantial research efforts of the *Tina, Race*, and *Acts* projects have added most of the interesting concepts to the resulting architecture. The drawback is that it is quite far away from the structure of current systems, and a transition would be expensive.

The PSTN providers have the choice of either making high investments into the architecture, or of postponing the introduction of many interesting services, or of saving money up front by introducing the few general-purpose "loop holes" at the price of increased sub-sequential costs for more service interaction problems.

Most mechanisms are still research work that detect service interaction problems off-line, before the services are installed. This is true for the *DFC* approach as well as for other approaches not presented here.

Undesired service interactions can still happen in the new architectures, such as Tina. This is so despite that Tina avoids several kinds of service interactions which can occur in the IN, for example in the users' interface and due to limited network support. *Violated assumptions* or *conflicting goals* can still cause undesired service interactions.

Kolberg and Magill report in [3] that many undesired service interactions known from the IN world can still happen between Tina services. Calling Number Delivery (CND) could be implemented in Tina. It allows its users to see the identification of the inviting party. Independently, Calling Number Delivery Blocking (CNDB) could be implemented. It allows its users to block the delivery of the identification information. When a user with CNDB invites a user with CND, it cannot be decided who has priority, and whether the identification information should be revealed. Tina does not provide any mechanisms to prevent such interactions, where user goals conflict.

Research projects need to restrict themselves in order to succeed in their focus area. Covering the concepts relevant for telephony is a huge task. We have seen how the DFC approach still does not include many important concepts. The *Agent Architecture* of Zibman et. al. also misses important telephony concepts such as mobility and separated stakeholders for billing. It is complementary to the DFC in its research focus. It allows to detect service interaction problems only late, at run-time. Instead, it stresses the modularization of the architecture.

Important Software Engineering problems remain unsolved. We don't know how to prepare for unanticipated changes, like mobile telephony or Internet telephony have been. The information hiding principle must assume that it is known which information is likely to change, and what will be stable. We only can alleviate the problem by taking great care and effort when making predictions.

We have better chances to confine a future maintenance task to one module only if we use an architecture that supports modularization and layers of services, and if we define carefully the known telephony concepts.

References

[1] Dirk O. Keck and Paul J. Kühn. The feature and service interaction problem in telecommunications systems: a survey. *IEEE Trans. Softw. Eng.* **24**(10), 779–796 (October 1998).

[2] Muffy Calder and Evan Magill, editors. "Feature Interactions in Telecommunications and Software Systems VI". IOS Press, Amsterdam (May 2000).

[3] Kristofer Kimbler and L. G. Bouma, editors. "Feature Interactions in Telecommunications and Software Systems V". IOS Press, Amsterdam (September 1998).

[4] E. Jane Cameron, Nancy D. Griffeth, Yow-Jian Lin, et al.. A feature interaction benchmark in IN and beyond. In L. G. Bouma and Hugo Velthuijsen, editors, "Feature Interactions in Telecommunications Systems", pages 1–23, Amsterdam (1994). IOS Press.

[5] ITU-T. "Q.12xx-Series Intelligent Network Recommendations" (2001).

[6] Jan Bredereke. Avoiding feature interactions in the users' interface. In Kimbler and Bouma [3], pages 305–317.

[7] John de Keijzer, Douglas Tait, and Rob Goedman. JAIN: A new approach to services in communication networks. *IEEE Commun. Mag.* **38**(1), 94–99 (January 2000).

[8] Ravi Jain, Farooq M. Anjum, Paolo Missier, and Subramanya Shastry. Java call control, coordination, and transactions. *IEEE Commun. Mag.* **38**(1), 108–114 (January 2000).

[9] Marcel Mampaey and Alban Couturier. Using TINA concepts for IN evolution. *IEEE Commun. Mag.* **38**(6), 94–99 (June 2000).

[10] C. Abarca et al.. Service architecture. Deliverable, TINA-Consortium, URL http://www.tinac.com/ (16 June 1997). Version 5.0.

[11] S. Trigila (ed.). "Open Services Architectural Framework for Integrated Service Engineering". Deliverable R2049/FUB/SAR/DS/P/023/b1, Version 4, RACE Project 2049 (Cassiopeia) (24 March 1995).

[12] The ACTS information window. http://www.infowin.org/acts (2000).

[13] George N. Prezerakos, Stefano Salsano, Alexander W. van der Vekens, and Fabrizio Zizza. INSIGNIA: a pan-European trial for the intelligent broadband network architecture. *IEEE Commun. Mag.* **36**(6), 68–76 (June 1998).

[14] Marco Listanti and Stefano Salsano. IBIS: A testbed for the evolution of intelligent broadband networks toward TINA. *IEEE Commun. Mag.* **36**(6), 78–91 (June 1998).

[15] Michael Jackson and Pamela Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Trans. Softw. Eng.* **24**(10), 831–847 (October 1998).

[16] Israel Zibman et al.. Minimizing feature interactions: an architecture and processing model approach. In Kong Eng Cheng and Tadashi Ohta, editors, "Feature Interactions in Telecommunications III", pages 65–83. IOS Press, Amsterdam (1995).