

**Prioritizing Features
Through
Categorization:**

An Approach to Resolving
Feature Interactions

by

P. Ann Zimmer

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2007

© P. Ann Zimmer 2007

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

P. Ann Zimmer

Abstract

Feature interactions occur when one feature interferes with the intended operation of another feature. To detect such interactions, each new feature must be tested against existing features. The detected interactions must then be resolved; many existing approaches to resolving interactions require the feature set be prioritized. Unfortunately, the cost to determine a priority ordering for a feature set increases dramatically as the number of features increases. This thesis explores strategies to decrease the cost of prioritizing features, and thus facilitates priority-based solutions to resolving feature interactions.

Specifically, this thesis introduces a categorization approach that reduces the complexity of determining priorities for a large set of features by decomposing the prioritization problem. Our categorization approach reduces this cost by using abstraction to divide the system's features into categories based on their main goal or functionality (e.g., block unwanted calls, present call information). Next, in order to detect and resolve the interactions that occur between these seemingly unrelated categories, we identify a set of principles for proper system behaviour that define acceptable behaviour in the global system. For example, a call that should be blocked by a call-screening feature should never result in a voice connection. The categories are then ordered, such that adherence to the principles is optimized. We show that using category priorities, to order a large feature set, correctly resolves interactions between individual features and significantly reduces the cost to determine priority orderings.

The four significant contributions that this thesis makes are: 1) the categorization of features, 2) the principles of proper system behaviour, 3) automatic generation of priority orderings for categories, and 4) devising several optimizations that reduce the search space when exploring call simulations during the automatic generation of the priority orderings. These contributions are examined with respect to the telephony domain and result in the identification of 12 feature categories and 9 principles of proper system behaviour. A Prolog model was also created to run call simulations on the categories, using the identified principles as correctness criteria. Our case studies showed the reduced cost of our categorization approach is approximately $1/10^{55}\%$ of the cost of a traditional approach. Given this significant reduction in the cost and the ability of our model to accurately reproduce the manually identified priority orderings, we can confidently argue that our categorization approach was successful.

The three main limitations of our categorization approach are: 1) not all features (e.g., 911 features in telephony) can be categorized or some categories will contain a small number of features, 2) the generated priority ordering may still need to be analyzed by a human expert, and 3) the run time for our automatic generation of priority orderings remains factorial with respect to the size of the number of categories. However, these limitations are small in comparison to the savings generated by the categorization approach.

Acknowledgements

I would like to express my gratitude to the many people, who either directly and indirectly, helped me with the writing of this thesis.

I would like to start by thanking my supervisor, Professor Joanne Atlee, for her guidance and support through my entire PhD journey. Jo, without your help and sound advice my research would have been much less focused. You helped me develop into a better researcher and writer; your dedication to your work is amazing.

Many thanks to my thesis committee, Jean-Charles Grégoire, Michael Godfrey, Richard Treffer, and John Thistle, for their thoughtful insights and ideas. Their many comments and suggestions helped to improve this thesis.

I also enjoyed my discussions with Pamela Zave and thank her for the valuable insights into industry-related research in the telephony domain. Her insights and responses to questions helped me understand some of the finer points of telephony.

I would also like to thank members of the WatForm lab at the University of Waterloo, their presence made day-to-day life more interesting. Thank you for supporting me, giving feedback, questioning everything, and offering me encouragement.

Thanks to all my friends whose support and understanding help to make me a better person. Special thanks to Mike Neame and Jenn Annis for volunteering to proofread my thesis. Mike, your feedback allowed me to identify areas that needed further explanation. Jenn, your editing expertise made this thesis clearer and easier to read. Thank you both.

Finally, I must thank my family. To my parents, Patsy and Cyril Meade, thank you for always believing that I could do anything I wanted, for supporting me and loving me no matter what. To my husband, Adam Zimmer, I could not have done this without you! You are my rock, you pushed me when I needed it, proofread my work so very many times, and took over running our home to give me the extra time I needed to focus on my writing. You are amazing and I love you!

Contents

1	Introduction	1
1.1	The Feature-Interaction Problem	2
1.2	Prioritization	3
1.3	Hypothesis and Thesis Overview	3
1.4	Thesis Contributions	5
1.5	Thesis Organization	7
2	Background and Terminology	9
2.1	Basic Call Structure	9
2.2	Address Zones	14
2.3	Multiple-User Connections	16
2.4	Feature Modules	18
2.5	Feature Interactions in Telephony	23
2.6	Comparison with the DFC Architectural Model	23
3	The Feature-Interaction Problem	27
3.1	Feature Interactions	27
3.2	Filtering	30
3.2.1	Use Case Maps	31
3.3	Detection	31
3.3.1	Requirement-Level Techniques	32
3.3.2	Formal-Methods Techniques	32
3.3.2.1	Finite State Machines	33
3.3.2.2	LOTOS	33
3.4	Prevention	33
3.4.1	Distributed Feature Composition	34
3.4.2	AIN	35
3.4.3	SIP	36
3.5	Resolution	36

3.5.1	Agent-Based Architecture Detection and Resolution	37
3.5.2	Fuzzy Logic	38
3.5.3	Negotiation	38
3.5.4	Priority Schemes	39
3.5.5	Rollback	42
3.6	Related Work	42
3.7	Summary of the Feature-Interaction Problem	45
4	Prioritization Using Categories and Principles	47
4.1	Classification of Features	48
4.1.1	Multiple Purpose Features	56
4.2	Principles of Ideal Feature and System Behaviour in Telephony	57
4.3	Example Interactions	59
4.3.1	Three-Way Calling versus Personal Directory	59
4.3.2	Call Return versus Make Set Busy	59
4.4	Correctness of Prioritization Using Categories	59
4.5	Manual Analysis Results	63
4.5.1	Observations	64
4.5.2	Principles and Category Ordering Results	65
4.6	Combining Categorization with Address Translation	69
5	Using Prolog to Automatically Generate the Partial Order	91
5.1	General Design Overview	92
5.2	Modelling Abstractions	95
5.2.1	Call Stage Routing Information	96
5.2.2	Call Stage (CS)	97
5.2.3	Segment Stage (SegStg)	99
5.2.4	Call Representation in Prolog (CallID)	101
5.2.5	Other Interesting Data Structures and Terminology of Interest	102
5.2.6	Category Representation	103
5.3	Execution Model	103
5.4	Feature-Transition Rules	104
5.4.1	Transitions Rules	105
5.5	Updating the Call State	107
5.6	Principle Assertions	108
5.7	Understanding the Output	109
5.8	Optimizations	119
5.8.1	Constraint Optimizations	119

5.8.2	Criterion Optimizations	120
5.8.3	Pairwise Optimizations	122
5.9	Chapter Summary	123
6	Evaluation	125
6.1	A Case Study in Telephony	125
6.2	An Advanced Case Study in Telephony	129
6.3	Cost Analysis: the Number of Feature Orderings	132
6.3.1	Pairwise Optimization	133
6.4	Cost Analysis: the Number of Call Scenarios	134
6.5	Analysis Results	137
6.6	Model Reusability	139
7	Conclusions and Future Work	141
7.1	Future Directions	145
7.1.1	Intra-category Prioritization Cost Reduction	145
7.1.2	Other Feature Rich Domains	147
7.1.3	User Defined Priority Schemes	147
A	Feature Names and Definitions	149
B	Continuation of Correctness Property Validation 4.6.1	153
	Bibliography	175
	Glossary	179
	Index	185

List of Tables

5.1	Signal Table	95
6.1	Total Feature Count	127
6.2	Feature data options	136
6.3	Run Time Results for Different Category Sets	138

List of Figures

2.1	Basic Call	10
2.2	Call containing multiple source and target address zones	15
2.3	Multiple Active Calls	16
2.4	Communicating Finite State Machine for Outgoing Call Screening	19
4.1	Example of an active Multiplex feature responding to an incoming call.	52
4.2	Example of a Remote Control Invoking feature	55
4.3	Partial Ordering of Feature Categories.	64
4.4	Multiple address zone call	69
4.5	Composition of address zones	72
4.6	Logging Principle: Blocking in Address Zone A_q	75
4.7	Logging Principle: Blocking in Address Zone A_p	75
4.8	Logging Principle: Multiplex in Address Zone A_q	77
4.9	Logging Principle: Multiplex in Address Zone A_p	78
4.10	Logging Principle: Set Outcome in Address Zone A_q	79
4.11	Logging Principle: Set Outcome in Address Zone A_p	79
4.12	Abortion Principle: Alias in Address Zone A_q	80
4.13	Abortion Principle: Alias in Address Zone A_p	82
4.14	Abortion Principle: GenCall in Address Zone A_q	83
4.15	Abortion Principle: GenCall in Address Zone A_p	84
4.16	Abortion Principle: Blocking in Address Zone A_q	85
4.17	Abortion Principle: Blocking in Address Zone A_p	86
4.18	Failure Principle: Set Outcome in Address Zone A_q	87
4.19	Failure Principle: Set Outcome in Address Zone A_p	88
4.20	Failure Principle: FailCall in Address Zone A_p and A_q	89
5.1	Feature Composition	92
5.2	Call Tree call1aBbS2	94
5.3	Call Stage Progression	98
5.4	BlockT Transrule (Initialization Source Zone)	105

5.5	BlockS Transrule (Initialization Source Zone)	106
5.6	Abortion Constraint Principle Assertion	110
5.7	Personalization criterion principle Assertion	111
5.8	Call Tree callaBbSrS4	113
5.9	Call Tree call1bSrS3	115
5.10	Call Tree call1bSrS3-r	116
5.11	A call scenario involving BlockS and Alias	117
5.12	A call scenarion involving Present and Alias	118
5.13	Criterion Optimization Method	121
6.1	Single call-simulation results from Prolog	128
6.2	Partial ordering results from single call-simulation analysis	129
6.3	Multiple call-simulation results from Prolog	130
6.4	Partial ordering results from multiple call-simulation analysis	131
6.5	Variable and Formulae Reference	133
6.6	Results generated by Case Study 1	135
B.1	Accessibility Principle: Redirect in Address Zone A_p	154
B.2	Accessibility Principle: Redirect in Address Zone A_q	155
B.3	Authorization Principle: Authentication in Address Zone A_p	156
B.4	Authorization Principle: Authentication in Address Zone A_q	157
B.5	Concretization Principle: Alias in Address Zone A_q	158
B.6	Concretization Principle: Alias in Address Zone A_p	159
B.7	Concretization Principle: Source Redial in Address Zone A_p	160
B.8	Concretization Principle: Source Redial in Address Zone A_q	161
B.9	Network Principle: Multiplex in Address Zone A_q	162
B.10	Network Principle: Multiplex in Address Zone A_p	163
B.11	Personalization Principle: Presentation in Address Zone A_p	165
B.12	Personalization Principle: Presentation in Address Zone A_q	166
B.13	Personalization Principle: Redial in Address Zone A_p	167
B.14	Personalization Principle: Redial in Address Zone A_q	168
B.15	Presentation Principle: Presentation in Address Zone A_q	170
B.16	Presentation Principle: Presentation in Address Zone A_p	170
B.17	Presentation Principle: TermCall in Address Zone A_p	171
B.18	Presentation Principle: TermCall in Address Zone A_q	172
B.19	Presentation Principle: Multiplex in Address Zone A_q	172
B.20	Presentation Principle: Multiples in Address Zone A_p	173

Chapter 1

Introduction

As a system grows in size and complexity, it becomes necessary to decompose the system into a collection of subsystems, or features, that are considered separately. This separation of concerns allows the actions and reactions of individual subsystems to be examined in isolation. Such **modular development** simplifies feature design and implementation, in that each feature is treated as a separate concern that overrides and extends the system's basic service. Modular feature development also decreases the time-to-market for new features, because, ideally, features can be developed in parallel or contracted out to third-party programmers and then added to the system without the feature developer needing to know about the presence of other features in the system. However, in practice, the addition of a new feature can interfere with the expected execution of those already in the system. This interference may occur because multiple features can assign conflicting values to shared data variables and can react inconsistently to the same input. Interference caused by the addition of a new feature is called a **feature interaction**.

Feature interactions pose a problem for service developers, as each interaction must be detected and correctly resolved to ensure proper functioning of the extended system. Several approaches to resolving feature interactions in modularly-developed systems require that the feature set be **prioritized**. Unfortunately, the cost to determine a priority ordering for a feature set increases dramatically as the number of features increases. This thesis explores strategies to decrease the cost of prioritizing features, and thus facilitates priority-based solutions to resolving feature interactions. Specifically, this thesis introduces a **categorization approach** that reduces the complexity of determining priorities for a large set of features by decomposing the prioritization problem, as discussed in Section 1.3. To promote modular feature development and facilitate the rapid development of new features, we restrict our work to features implemented as independently developed modules. We assume that features are composed together in a system with an architecture that coordinates the features' execution and communication.

1.1 The Feature-Interaction Problem

Feature interactions are a major problem in incremental software development because the naïve addition of new features often interferes with the correct functioning of existing features in the system. A typical interaction occurs when one feature prevents another feature from executing, or when the combined effect of several features is inconsistent. For example, an interaction occurs when telephone feature Redirect, which redirects a call attempt to another telephone number, fails to execute because its invoking signal is intercepted by another feature; in this case, the call is not redirected as expected. Another telephony example is a Billing feature that records information about a call attempt that is blocked and torndown by a Call-Screening feature before a voice connection is established.

Feature designers spend significant time and effort analyzing how new features interact with existing features. Feature interactions must be identified and assessed as to whether the interactions are desirable or if they need to be resolved. When an undesirable interaction is detected, the designer must modify the new feature, the existing system, or both to resolve the interaction. Thus, the cost of adding a new feature can be high. When only pairs of features are analyzed for interactions, then the analysis of a new feature with respect to existing features is proportional to the number of features already implemented in the system. If larger subsets of features are analyzed for interactions, then the analysis grows exponentially with respect to the number of existing features. Thus, the time to develop a new feature grows as the size of the system grows, affecting not only development costs, but also potential revenue streams and market share, because customers must wait for new features to be analyzed and implemented.

A feature designer can use several different techniques to address the feature-interaction problem, such as filtering, detection, prevention, and resolution. Filtering identifies interaction-prone features, so that a full system analysis need not be performed. This step is followed by a detection algorithm, which identifies actual interactions. Finally, the designer applies a resolution strategy to correctly resolve the identified interactions. An alternative strategy is to use prevention to avoid the occurrence of feature interactions, which can sometimes be accomplished by adjusting the architectural style or modifying protocols for coordinating various features.

A key approach to ensuring interoperability among independently developed features lies in architectures, design rules, and protocols that constrain and control how features interact with each other. For example, AT&T's Distributed Feature Composition (DFC) pipe-and-filter architecture [27], precedence rules for dispatching input events to features [54], call filters [13], and patterns [49] all resolve interactions between independently developed features by prioritizing features' reactions to events. **Prioritization** is an effective method for preventing or resolving conflicts between feature modules: in a call situation where multiple features are enabled, the highest-priority feature reacts first, and then decides whether to preserve the enabling situation (e.g., output the signal unchanged) to allow lower-priority features to also execute.

1.2 Prioritization

Prioritization is extremely important in a system in which features are applied, one after another, in a serial manner. The order in which features are applied affects which features receive and process signals; hence, modifying the order in which features are applied changes the behaviour of the system. Adjusting a feature's priority changes the feature's execution order within the system and can help correctly resolve feature interactions. Consider a user who subscribes to Personal Directory, which translates a code dialled by the user into a corresponding telephone number, and Call Screening, which allows the user to block undesirable call attempts based on telephone numbers entered into a call-screening list. If the user initiates a call by entering one of the Personal Directory codes and the Personal Directory feature is triggered first, then the dialled code is translated into its equivalent telephone number before the *call-setup* signal reaches the Call-Screening feature. The Call Screening feature compares the telephone number to the numbers on the call-screening list, finds the number on the list, and terminates the call attempt. Alternatively, if the priorities are reversed, and the Call Screening feature receives the *call-setup* signal with the dialled code, then the code is not found on the call-screening list, as the call-screening list contains only telephone numbers, so Call Screening allows the call to continue to be setup. This second ordering results in an undesirable interaction, as a call to a telephone number that is on the call-screening list is established.

Previous work on prioritizing features includes the work of Elfe et al. that uses prioritization as part of their technique for testing constraint violations [18] and Tsang and Magill's work that uses prioritization in a run-time feature interaction detection and resolution strategy [48]. These prioritization techniques have several limitations. Only feature pairs are considered in [18]; thus, if a feature interaction occurs only when a larger feature set executes, then this interaction is not detected and addressed when prioritizing pairs of features. Moreover, the designer must provide a default priority ordering that is used whenever an exception arises [18, 48]. As well, these prioritization techniques have scalability issues, as the addition of each new feature results in a significant cost increase to compute a new acceptable ordering. In fact, due to the large number of features found in many domains, it is questionable whether a large feature set can always be prioritized.

1.3 Hypothesis and Thesis Overview

This thesis focuses on reducing the cost of prioritizing a set of independently developed features, where the priority ordering is used to resolve feature interactions. Our hypothesis is that **categorization** can be used to reduce the cost of determining a priority ordering for a set of features by decomposing the prioritization problem. Categorization decomposes the prioritization problem into two phases: partitioning features into categories and then sorting separately the

set of categories and the set of features found within each category. To begin, each feature is categorized based on its goal or functionality. For example, the goal of a number of features is to block unwanted incoming calls from reaching the subscriber. An alternative feature goal is to present information (e.g., caller identification, call time) to the user. Although the functionalities of these latter features vary in behaviour (e.g., via ring tones or textual displays), their common goal is to present information.

The criteria for identifying and scoping feature categories are to maximize cohesion and minimize cohesion between the categories. Maximizing cohesion between the features within a category increases the similarity of the goals and behaviour patterns of those features and thus increases the probability that the features manipulate the same shared variable, while minimizing the cohesion between the categories minimizes the dependencies between features and thus reduces the possibility of interactions between features in different categories.

To detect and resolve the interactions that occur between these seemingly unrelated categories, we identify a set of **principles for proper system behaviour** that define acceptable (or unacceptable) behaviour in the global system. An example of a principle in the telephony domain is: a call that should be blocked by a call-screening feature should never result in voice connection. These principles are used to identify optimal orderings among the categories, where an optimal ordering is one that violates the smallest number of principles.¹

Specifically, when we use the principles for proper system behaviour as correctness criteria, we are able to detect the following types of interactions between feature categories: **constraint violations**, which occur when undesirable data is recorded in the database; **global-invariant violations**, which occur when the call attempt enters an undesirable call state; and **data modifications**, which occur when features attempt to record conflicting values to a shared database entry. We do not claim that our **categorization approach** presented in Chapter 4 prevents or resolves every interaction of these types, just those that are expressed as principles to be upheld.

High cohesion among features in the same category means that they have very similar goals and behaviour patterns, making it highly probable that they will attempt to manipulate the same shared data variables. The resolution of these **intra-category** interactions does not usually depend on global correctness criteria, but rather on user preference. For example, if two features react to the same event (e.g., Call Forward on No Answer and Voice Mail both attempt to handle the case where the caller is unavailable), then the human expert must decide which feature has priority and should respond to the event first. By decomposing the prioritization problem, we can automatically generate resolutions to unintended interactions between feature categories using the principles for proper system behaviour, thus freeing up resources (i.e., the human expert) to focus on resolving interactions between intra-category features.

The intra-category and category orderings are then combined to produce optimal orderings of

¹A more formal definition of an optimal category ordering is given in Chapter 4.

features that adhere to principles of proper system behaviour. These orderings can then be used as input to several proposed priority-based resolution strategies (see Section 3.5.4) to aid in the feature-interaction problem.

We hypothesize that our categorization approach is general enough (and the number of categories is small enough) to automatically generate optimal priority orderings for the feature categories. To evaluate this hypothesis, we created a model using Prolog that simulates a telephony environment and that takes as input a set of categories and a set of principles for proper system behaviour. These inputs are used during call simulation to identify and resolve interactions that occur between categories. The model generates as output a formally verified set of optimal priority orderings for the categories, each of which adheres to the principles. In Chapter 6, we generate prioritized orderings for 11 categories, using 9 principles identified by our categorization approach for the telephony domain.

1.4 Thesis Contributions

There are four significant contributions that this thesis provides.

Categorization of Features: The categorization of features and the ordering of these feature categories are design rules presented in this thesis. The categorization constrains the design as each feature component must be placed in one and only one category. The acceptable category orderings also limits the placement of features, such that features are grouped together and implemented in an ordering that correctly resolves feature interactions within the domain. This categorization of features contributes to the field of feature interaction in two ways. First, in order to correctly categorize features, it is necessary to develop an understanding of the motivation and essential functionality of the features in the domain, in our case the telephony domain, so that we can use this knowledge to create a set of meaningful feature categories. A simple set of guidelines for creating the feature categories was developed to help correctly categorize the features. These guidelines increase the effectiveness of the approach by increasing the ability to find unexpected interactions that occur between seemingly unrelated features that are found in different categories. The second and larger contribution is that the problem of prioritizing a large feature set can be abstracted and reduced to the problem of prioritizing the identified feature categories, and the problems of prioritizing the smaller sets of intra-category features. The categories are constructed such that the abstraction preserves the functionality and goals of those features within the category, and hence preserves the presence of feature interactions between categories. When features are composed into categories, the reduction in the cost to prioritize features is significantly high for large sets of categorizable features.

Principles of Proper System Behaviour: The principles for proper system behaviour are a set of principles that describe the correctness conditions that should hold within the system. These principles provide the justification steps that are used when ordering the features and their categories using the categorization design rules. These principles are identified by the system designer and/or end users and are used to indicate the conditions under which the ordered feature set meets the specifications for the domain. Our contribution is demonstrating that such a set of conditions, in our case for the telephony domain, can adequately constrain the system and identify necessary restrictions on the feature categories. We demonstrate this result, in the telephony domain, by providing a set of principles and using these principles to order the telephony categories. Defining a set of correctness criteria for a system is a balancing act between defining too many criteria, which overly constrains the system, and defining too few criteria, which results in poor system behaviour as unacceptable behaviour may not be prevented. It is especially challenging that the correctness criteria must apply irrespective of which features are present. Good decomposition of features into categories eases the task of creating the correctness criteria, since we can use the goals of the categories to define principles that hold for both the base system and its features. These principles are used to evaluate the behaviour of potential feature-category orderings.

Automatically Generated Orderings : Another contribution of this thesis is that we designed and developed a Prolog model that automatically generates and formally verifies optimal orderings for a set of feature categories, according to a set of input principles of proper system behaviour. Prolog exhaustively searches the possible call execution paths created using all possible feature-category orderings. The Prolog model evaluates the category orderings' adherence to the principles. Because manual analysis of feature interactions is tedious and error-prone, and because all possible feature orderings are systematically considered and evaluated, the model's output results are more accurate than orderings generated during manual analysis. Moreover, the model's output provides designers with a reduced solution space of optimal orderings, so that designers can concentrate their efforts on determining the best-optimal ordering from the set of viable candidates output by the Prolog model.

Search Optimizations: One of the main concerns with the prioritization problem is the large search space that needs to be explored. In an effort to reduce this search space further, we introduce several techniques that identify invalid suborderings. For example, one technique explores all category pairs for principle violations and removes from consideration all larger feature sets that contain a pair known to cause a principle violation. The existence of one such pair reduces by half the number of full orderings that must be searched. Adding

these techniques to our categorization approach allows for a substantial cost reduction when prioritizing large feature sets.

Our evaluation of the optimized Prolog model, using the categories and principles identified in Chapter 4 for the telephony domain, shows a significant reduction in the cost of determining priority orderings for a set of telephony features.

1.5 Thesis Organization

The rest of this thesis is organized as follows. We begin in Chapter 2 with an overview of the telephony domain and the terminology we use throughout the remainder of this work. Chapter 3 gives a detailed study of the feature-interaction problem. Chapter 4 describes the categorization approach and its application to the telephony domain. Chapter 5 describes a Prolog model that simulates the Distributed Feature Composition telephony architecture and that automatically generates optimal orderings of input feature categories according to the principles of proper system behaviour. In Chapter 6, we discuss the results of the Prolog simulation, analyze the reduction in cost of using categorization to prioritize features, and discuss the limitations of our approach. We conclude and discuss future work in Chapter 7.

Throughout this thesis, we use a combination of different fonts to designate special words and phrases, which are identified below:

Definition: identifies a new term that is introduced and **defined**, either for the first time or as an extension to a previous definition.

Highlight: identifies the first occurrence of a **key word** or **phase** in a new chapter or section.

Category Name: designates a feature category name, such as **BlockS** or **RedialT**

Principle Name: designates a principle name, such as *Logging* or *Accessibility*

Prolog Data Structure: designates a data structure in our Prolog model, such as *call state* information and *database* variables

Prolog Equation: designates a Prolog equation, such as a feature transition rule or principle assertion

Signal Name: designates a signal issued during a call-attempt, such as *avail* or *teardown*

Chapter 2

Background and Terminology

The majority of research into the feature-interaction problem focuses on the problem as manifested in the telephony domain [6, 16, 17, 30, 9, 43]. In this chapter, we review the structure of a telephone call including how features are implemented on top of the basic telephone system and how these features affect the call. We have chosen to work with a model based on AT&T’s Distributed Feature Composition (DFC) [27], since its architecture is well-suited to the development of modular features. Consequently, we use DFC-based terminology throughout this work.

In this Chapter, we explore the various components of the telephony domain, starting with the basic structure of a call in Section 2.1. Section 2.2 describes the different address zones associated with a call, while Section 2.3 explains how multiple-user calls are established. Section 2.4 explains the execution of a telephony feature. Section 2.5 identifies common types of feature interactions in telephony. In Section 2.6, our DFC-based model is compared against the original DFC architecture.

2.1 Basic Call Structure

We define a **service** as the core functionality of the system, and a **feature** as any add-on functionality that extends the basic service. For example, a feature in the telephony domain is Voice Mail, which offers to record a message for the subscriber when the subscriber is unavailable to answer the phone. In telephony, each feature subscription is associated with an address (i.e., telephone number); the addresses involved in a call determine which features are instantiated for that call. We use the term **module** to refer to a component that is part of a feature’s implementation. We use the term **feature** to refer to both a feature and its component modules. In addition to feature modules, there are **device-specific interface modules**, designated IM in Figure 2.1, that translate between the protocol signals (i.e., $sg1, sg2, sg3$) understood by features and the device signals (i.e., $sg1', sg4', sg5'$) issued and understood by a device (e.g., telephone, computer, network gateway). Thus, via an interface module, a feature can be used in combination with

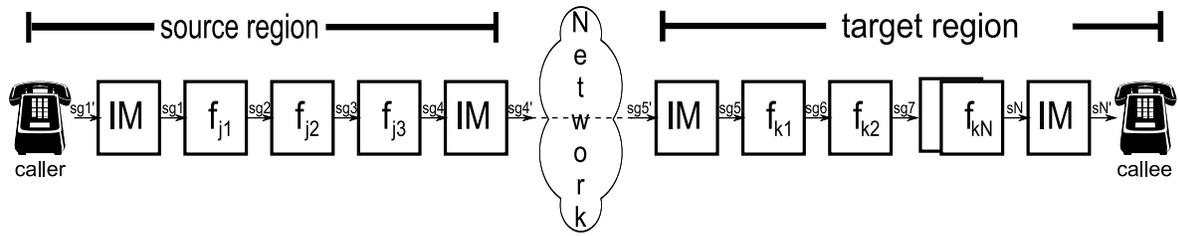


Figure 2.1. A basic call is initialized by the $sg1'$ signal from the caller's end device. The signal is interpreted by the device's interface module and then passes to the caller's features, $f_{j1} - f_{j3}$, possibly changing values along the way. The $sg4$ signal uses the Network's interface module to route the call setup signal through the network, where the $sg5$ signal enters the callee's feature, f_{k1} , and continues through the remaining features, $f_{k2} - f_{kM}$, and into the corresponding end device to establish a voice connection.

different devices.

Definition 2.1.1. Address:

*Telephony services are associated with unique telephone numbers that represent the **address** at which an end user can be reached. Each address is associated with a specific set of features that have been subscribed to by the owner of this address. Any call that originates or terminates at a specific address will include in its call path all of the features associated with that address.*

Definition 2.1.2. Module:

*Each **module** is a communicating finite state machine (CFSM) and represents either a feature module, an interface module, or a device module. Each CFSM has at least two ports on which they can send or receive signals. When a signal is received on one of the ports, the signal will trigger the execution of a transition within the model. This transition may generate a new call state, effect changes to the database system, and/or issue an output signal that is sent out along one of the named communication ports.*

Definition 2.1.3. Feature Module:

*A **feature module** is a module that represents the functionality of any add-on feature to the basic service. A single feature may be represented by multiple feature modules; these feature modules will each represent a core component or functionality of the feature.*

Definition 2.1.4. Device-specific Interface Module:

*A **device-specific interface module** is a module that represents both the functionality of an end device and the interface used by the device to process incoming and outgoing signals. Device-specific interface modules are usually found at the end of a call path where the end device is located. These modules are able to accept or reject an incoming or outgoing call on behalf of the device, as well as representing user input by initiating signals that can be sent along the call path. For example, a device-specific interface module can propagate a signal that indicates that the end device is free and able to accept calls (avail) and is also able to propagate a signal that indicates that the end device is busy and unable to accept new incoming calls (unavail).*

A **call** is a sequence of feature and interface modules, as shown in Figure 2.1. A call becomes **established** when a voice connection is completed between the end-users. The end points of an established call are the users, but if the call is in the process of being set up or torn down, then the end points could be modules. The goal of each call attempt is to form an established call. For any given call, we distinguish between the user (**caller**) who initiates, or whose features initiate, the call and the user (**callee**) who receives, or whose features receive, the call. Furthermore, we identify a feature's **subscriber** as the user who signs up for and pays for the feature, which may or may not be the user who invokes or is affected by the feature.

Definition 2.1.5. Call:

*A **call** is the abstract entity representing a sequence of modules that make up a telephone connection (ideally or actually) between two end users.*

*A **call** refers to either an established call or a call attempt.*

Definition 2.1.6. Call Attempt:

*A **call attempt** is a call (i.e., an abstraction of a sequence of modules) that is in the process of trying to establish a connection between two end users. In a call attempt, at least one of the two end points of the call is not stable, as the routing of the call is not complete and more feature modules will be added to the call.*

Definition 2.1.7. Established Call:

*An **established call** is a call (i.e., an abstraction of a sequence of modules) that has established a voice connection between two end users. Once the voice connection is established, all the features associated with this call are represented by one or more of the feature modules in the sequence forming the call. The end points of*

the sequence are usually device-specific interface modules representing the device (e.g., telephone) used by the caller or callee. However, an end point can also be a module that represents a feature designed to interact with the user (e.g., the voice mail server, automatic message relay features).

Each call attempt progresses towards establishing a call by adding new feature modules one at a time to the **call path**. When all of the feature modules belonging to the caller have been successfully added to the call path, then the appropriate network interface module is added to the call path. Next, the interface module transforms and passes the *setup* signal (the request to establish a call) to the Network module, which routes the *setup* signal to another network interface module at the callee's address. The **Network** module represents the routing of the call through the switching system of the basic telephony system. For simplicity, we represent the Network module and its corresponding network interface modules as a single module in all future figures. When the *setup* signal is passed to a new address's interface module, it transforms and continues the *setup* signal into the callee's features, which are added one-by-one into the call path. Finally, the *setup* signal reaches the callee's phone interface module, which interprets the signal for the end device. The end device rings, notifying the callee of the incoming call. The call becomes established once the callee answers the call.

Definition 2.1.8. Call Path:

*A **call path** is the concrete sequence of communicating modules that are instantiated as part of an established call or call attempt. A pair of modules are connected by a first-in, first-out communication channel that maps from a port on one module to a port in its neighbouring module. There is no limit on the number of signals that can be enqueued by the channel. Each port can be connected to at most one communication channel and hence only modules with more than two ports can connect with more than two neighbours; the number of neighbours a module has is always less than or equal to the number of communication ports owned by the module.*

Definition 2.1.9. Network Module:

*A **network module** is a module that models abstractly the routing of a call from one address to another (i.e., functionality of the trunk line in land lines or network routers in IP telephony). Once the network module has been added to the call path of a call, the module routes signals from one address to another.*

Definition 2.1.10. Call Attempt Protocol:

*A call attempt is **initialized** when one of the modules, usually a device-specific interface module, sends a *setup* signal from one of its ports. When the *setup* signal is*

output by the module at the unstable end of the call path, the next module is added to the call path by adding a communication channel between the sending port of the sending module and an unconnected port on the new module. Modules are added to the call path in the following order:

- 1. A device-specific interface module or a feature module with the ability to automatically generate a setup signal.*
- 2. The feature modules (in order) for the initial address.*
- 3. The network module (or a device-specific interface module) that continues the call attempt into the next address.*
- 4. The feature modules (in order) for the next address.*
- 5. If another address is to be added to the call path, then go to step 3. Otherwise, a device-specific interface module representing the final end point is added and the call attempt protocol is complete.*

Using DFC terminology [27, 52], each call is partitioned into a **source region**, which comprises the caller's part of the call (e.g., the caller's device and features) and a **target region**, which comprises the callee's part of the call (see Figure 2.1). We use the informal terms **outgoing call** and **incoming call** to refer to a call setup signal in the context of source-region or target-region features, respectively. Usually, these terms are associated with end-users, in that a caller places outgoing calls, while a callee receives incoming calls. Note that an end-user can act as both a caller and a callee when the end-user is involved in multiple calls simultaneously. For example, if Tim receives a call from Sam and then uses his Three-Way Calling feature to initiate a call to John, thereby establishing a three-way voice connection, then Tim is the callee with respect to the call involving Sam and is the caller with respect to the call involving John. Because a user may be both a caller and a callee, and because these designations are dynamic, as shown in the above example, all of a user's features must be included in every call, rather than just the caller's source features and the callee's target features.

Definition 2.1.11. Source Region:

*The **source region** is a portion of the call path representing the modules associated with the person who initiates the call (i.e., caller). The source region includes the device-specific modules used by the caller and the feature modules subscribed to by the caller and any modules found in addresses through which the caller forwards the outgoing portion of the call attempt. In a successful call attempt, the source region is followed by a target region.*

Definition 2.1.12. Target Region:

*The **target region** is a portion of the call path that represents the modules associated with the person who receives the call (i.e., callee). The target region includes the device-specific modules used by the callee and the feature modules subscribed to by the callee and any modules found in addresses through which the callee redirects the incoming portion of the call attempt.*

2.2 Address Zones

The existence of features that redirect and forward calls to new addresses add to the complexity of a basic call structure. Calls may be routed through many different locations (e.g., telephones, user addresses, trunk switches, network routers, PBX devices) before a connection is established. Each of these locations, which we call **address zones**, can add features appropriate to the location as the call attempt progresses towards an established connection. Examples of features that may be members of different address zones include speed dialling programmed inside a user's telephone, call waiting and other service-provider features, routing features executed by the router to direct the call, and PBX features that monitor billing. As well, the same feature may be present in multiple address zones simultaneously. For example, a call may pass through multiple address zones in either region (e.g., a call is forwarded from a home-based service-provider address zone and into a work-based service-provider address zone) and any feature, such as call waiting and caller identification, can be present in any/all of these address zones.

Definition 2.2.1. Address Zones:

*The source and target regions of a call can each be broken down into multiple **address zones**. Each address zone represents the feature and interface modules associated with an address. In addition, the call path may include device-specific address zones that add feature modules associated with any device (e.g., telephone functionality, routing features) found along the call path.*

To represent all these different sets of features, we divide the call attempt into a set of source and target address zones, where each address zone has its own feature set. For example, if the initial call request to one target address is forwarded to another target address, then there will be multiple address zones in the target region. In such a case, the features j_1, \dots, j_m associated with the initial target address are included in the call, but only up to the feature that, like j_2 in Figure 2.2, redirects the call to a new target address; after that, the feature set k_1, \dots, k_n associated with the new target address is incorporated into the call. Similarly, there will be multiple address zones in the source region if the caller routes her call through different originating addresses - such as routing a call from home through a work address. Such a call incorporates the features f_1, \dots, f_p

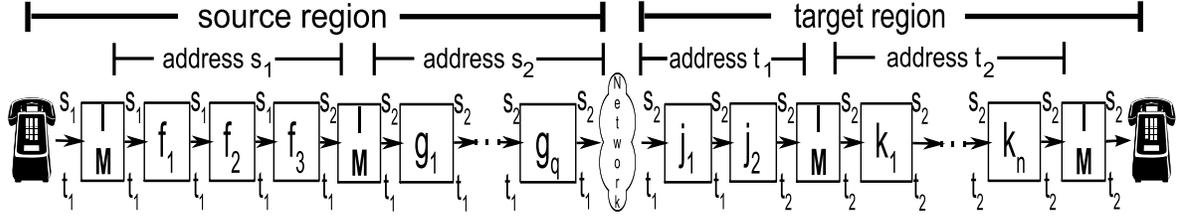


Figure 2.2. An established call, starting with an initial source address s_1 and target address t_1 . The arrows between modules are annotated with the current source (above) and target (below) address. Feature f_3 redirects the call through a second source address s_2 ; while feature j_2 redirects the call to a second target address t_2 .

associated with the home number only up to the feature that, like f_3 in Figure 2.2, redirects the call; after that, the features g_1, \dots, g_q subscribed to by the work number are incorporated. If feature sets from two different address zones use different protocols (e.g., the feature sets are from different providers and use different signals), then an interface module translates the signals as they cross the address zone boundaries. In our telephony model, we abstract away the device-specific address zones and the network/routing device address zones, and represent these zones as singleton interface and network modules. This decision simplifies the call simulation process without effecting the behaviour of the features.

Incorporating multiple address zones into a call path introduces an additional layer of complexity with respect to feature interactions and the ordering of features in the call path. Zave discusses this specific problem in [52] and introduces the notation of **Ideal Address Translation** (IAT) as a methodology to determine how features in different address zones should work together. The new terminology introduced throughout this section is taken from the concept of IAT.

The address zones, each of which is a collection of ordered features, are ordered according to IAT. In Figure 2.2, we see a call that was composed by incorporating features from two source address zones, s_1 followed by s_2 , and two target address zones, t_1 followed by t_2 . When a call transitions from one address zone to another, an Interface Module (IM) is used to connect the address zones; the IM can represent a simple interface module between different address zones within the same network or the routing from one network to a device on another network. Each of these address zones represents features provided by an end device, or subscribed to by a person or a role (i.e., features assigned to a certain type of employee like an Executive Assistant). Using IAT terminology, an address zone that is closer to the subscriber is more **concrete** than an address zone nearer the network, which is considered more **abstract**. For example, in Figure 2.2, s_1 could be the address zone of the end-device, while s_2 is a more abstract personal-address zone (i.e., a home may have multiple telephone devices but only one subscription to telephone

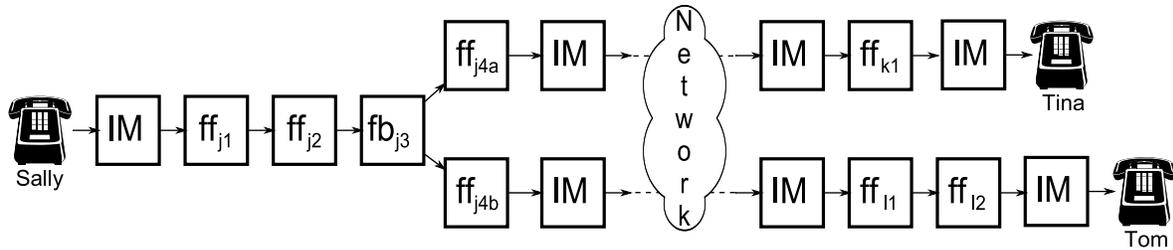


Figure 2.3. This figure shows Sally in two simultaneous calls. The same instance of free features ff_{j1} and ff_{j2} are used for every connection involving Sally, since these occur before the bound feature fb_{j3} , while new instances of the free feature ff_{j4} , ff_{j4a} and ff_{j4b} , are created for each call (Tom and Tina) connecting to Sally.

service and only one set of subscribed features). As the call is routed to each address zone, the features subscribed to in that address zone are added to the call path. There are no restrictions on what features are applied in each address zone; consequently, the same type of feature box may be found in multiple address zones.

Address translation and the IAT principles are explored in more detail in Section 4.6, where we discuss how address zones and our categorization approach are integrated. We also discuss the results of combining these two approaches in Section 4.4.

2.3 Multiple-User Connections

Normally, a new feature instance is instantiated whenever the feature is invoked in a call. For example, a new Call Screening feature module is instantiated for each incoming call attempt to the subscriber; each instance of this feature has access to the same call-screening list and is able to appropriately terminate all undesirable incoming calls. In addition, there are special multi-user features that coordinate calls between three or more users. These features need to be able to coordinate between all possible connections involving the subscriber, so these features cannot be instantiated anew for each call in the same manner as the Call Screening feature.

To address the needs of special multi-user features, we distinguish between free and bound features. A **free feature** is one for which a new instance is created for each call. When a user is involved in a call and features have been instantiated for that call, then a second call to that user will spawn new instantiations of the user's free features. In contrast, a **bound feature** is one for which there is a single instance for each user, and all calls involving the user are routed through this instance. Any feature, such as Call Waiting, that needs to know about all calls to the user is implemented as a bound feature.

Definition 2.3.1. Free Feature Module:

*A **free feature module** is a feature module that is transient and interchangeable. Whenever a free feature module is added to the call path, a new instance of the module is generated.*

Definition 2.3.2. Bound Feature Module:

*A **bound feature module** is a feature module that is persistent and dedicated to a particular address zone. Only one instance of this feature module will exist for each address zone at any given point in time, so that if a new call is initiated or received while a bound feature module is already in use by another call, then this new call will use the already instantiated version of this bound feature module.*

As shown in Figure 2.3, there is a single instance of free features (ff_{j1} and ff_{j2}) that are linked into the call between the subscriber and the bound feature (fb_{j3}) and these features operate on all calls/call attempts in which the subscriber is connected. In contrast, free features on the network side of the bound features (e.g., ff_{j4}) each have a separate instantiation (ff_{j4a} , ff_{j4b}) for each call/call attempt.

The existence of bound features changes the traditional appearance of a call, as portions of the call path may be shared by several calls. As a result of this change, we introduce two new terms **call segment** and **subcall**. Each **call segment** identifies a different portion of the call that is formed due to the presence of a bound feature¹ [28]. In Figure 2.3, we have two separate calls², one between Sally and Tina and the other between Sally and Tom. The call between Sally and Tina is composed of the call segment from Sally's end device to the bound feature box fb_{j3} plus a second call segment that joins to the bound feature fb_{j3} and incorporates the remaining features starting with the free feature ff_{j4a} and continuing towards Tina's end device. Similarly, Sally's call to Tom is composed of the same initial call segment from Sally's end device to the bound feature fb_{j3} and the call segment linking to this initial call segment and containing all remaining features, starting with the free feature ff_{j4b} and continuing towards Tom's end device.

Definition 2.3.3. Call Segment:

*A **call segment** is a distinct portion of the call path that indicates where new call attempts can be connected into the existing call path. Each completed call segment is delimited at either end by either an end device interface module or a bound feature module. A call segment is incomplete if routing is still applying new modules to the segment's call path: these new modules become part of this call segment. Whenever*

¹These features will be identified as the Multiplex feature category in Chapter 4.

²The voice connections for these calls may be joined in a three-way connection, however we still consider these to be two distinct calls.

a bound feature module is added to the call path, the module marks the end of the current call segment and the beginning of a new call segment.

The term **subcall** represents any subsequence of a call path. To distinguish between a call segment and a subcall, consider the following: each call segment forms a subcall, however a subcall may span multiple call segments.

Definition 2.3.4. Subcall:

*A **subcall** is a term used to express any continuous subsequence of communicating modules in a call path.*

2.4 Feature Modules

Each **feature** is modelled as a communicating finite state machine (CFSM). Each feature sends and receives signals through two or more communication **ports**. A **signal** refers to any inter-feature communication, which may be a message, an event, a method invocation, and so forth. It is through the passing and receiving of signals that the features coordinate the setup and teardown of a call. When a feature, f_{j1} , receives a signal, $sg1$, it changes state as it reacts to the signal by performing the appropriate actions (e.g., setting variable values, issuing signals). Each destination state is either a stable or unstable state: a **stable state** is any state that accepts incoming triggering events (i.e., signals) and whose transitions to other states are triggered by these events, and an **unstable state** is any temporary state whose state transitions are based on conditional results, unstable states are used to transition from one stable state to another stable state. Once in a stable state, the feature remains in this state until another input signal is received. The feature can pass the received signal unchanged, pass a new signal, or delay issuing a signal pending user input.

Definition 2.4.1. Signal:

*A **signal** is any communication event that can be sent along the communication channel that links the ports of neighbouring modules. The signals may be generated by an end user's device or by one of the modules, and are part of a routing protocol for setting up and tearing down calls.*

Figure 2.4 shows the CFSM for the Outgoing Call Screening feature, which terminates outgoing call attempts made to a number found on the call screening list. This feature has two ports, *Port 1* and *Port 2*, and transitions from the stable state *Idle* upon the receipt of a signal at either of these ports. Notice that *InSig*, an input signal received by *Port 2*, is always output unchanged through *Port 1* to the neighbouring feature via the transition labelled

Outgoing Call Screening

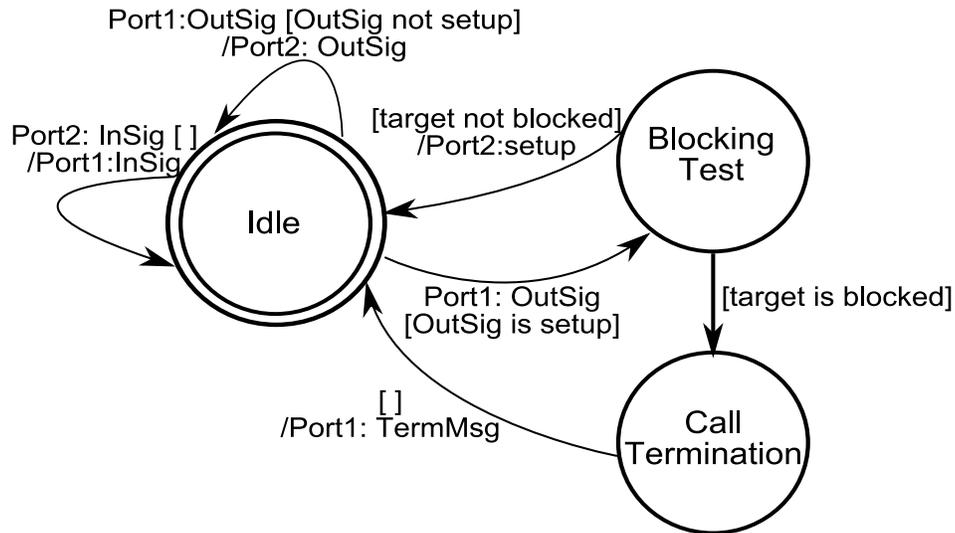


Figure 2.4. The goal of a Outgoing Call Screening (OCS) feature is to prevent any outgoing call attempts to a target address found on the call screening list. The transitions of this CFMS represent this goal, with undesirable call attempts being terminated via the path through the *Call Termination* state. The CFMS has one stable state, *Idle*, denoted by the double circle. Transitions from this state are triggered by the receipt of an input signal. *Port 1* is the port to the subcall leading to the caller, which receives the input signal *OutSig*, while *Port 2* is the port to the subcall leading to the callee. *Port 2* receives the input signal *InSig*. The transition label:

PortName : Input Signal [condition] /PortName : Output Signal

means that when the named port receives an input signal and the condition is met, then this state transition occurs and the output signal is issued out of the second named port. For example, the feature behaves transparently for all signals received by *Port 2*, propagating *InSig* along *Port 1* and transitioning back into the *Idle* state.

Port 2 : InSig [] /Port 1 : InSig. However, the input signal, *OutSig*, received by *Port 1*, can either result in sending a termination message, *termMsg*, along *Port 1* via the path through the CFMS that leads to the *Call Termination* state, or in continuing the call attempt with *OutSig* being output unchanged through *Port 2* to a neighbouring feature.

In this thesis, we assume that all features use the same set of DFC signals and protocols; however a different set of signals and protocols can be easily accommodated using interface modules to translate between protocols. The core set of DFC signals are: **setup** (initiate a call attempt), **teardown** (end a call), **avail** (callee's device is available to receive a setup request), **unavail** (callee's device is unavailable to receive a setup request), and **redirect** (modifies the address for the caller or callee of this call). For simplicity, in this work, we assume that signals are sent directly from one feature to another; but, depending on the architecture of the domain

being considered, signals could be routed through a dispatcher, a feature-interaction manager, and so forth.

Definition 2.4.2. Implemented Core DFC Signal Set:

The standard DFC signal set that is implemented within our DFC-based telephony domain.

setup: *this signal initializes the call attempt. When a module issues this signal, the next feature module to be added to the call path is initialized and this signal is passed into the new module for processing. Usually a setup signal continues to be propagated through the call path until either a feature module terminates the call attempt or an end device is reached.*

avail: *when a device or feature module determines that the callee is able to accept the incoming call, then the module issues this signal to alert the caller of the successful status of the call attempt. This module also presents the incoming call to the callee. Usually, the avail signal passes transparently through the modules found along the entire length of the call path, however some feature modules, such as **Redial** features, can react to this signal and change the progress of the call attempt.*

unavail: *when a device or feature determines that the call attempt is unsuccessful, usually due to the unavailability of the caller, then the module issues this signal. Usually this signal passes transparently through each module in the call path, unless a module is designed to treat failed call attempts, in which case the module will usually redirect the call attempt.*

redirect: *when a feature module determines that the call attempt is to be redirected to an alternate address, then the module issues this signal. Usually this signal causes the call attempt to immediately exit the current address zone. The next module added to the call path processes the input redirect signal and, usually, outputs a setup signal continuing the call attempt by adding the modules associated with the new address zone.*

In addition to the above core set of signals, the following signals are also used: *answer* (call is answered and accepted), *ringTO* (call attempt is not answered), and *feature-specific* signals (signals designed to work with specific feature modules). See Definition 2.4.3 for the complete signal set used in our domain. To explain the advantages of using *feature-specific* signals, consider the following example: historically, when implementing Three-Way Calling (3WC) and Call Waiting (CW), the *flashhook* signal was used to trigger both of these features. When the

subscriber of 3WC issued a *flashhook* signal while already involved in a call, the 3WC feature would trigger, placing the other party of the current call on hold and then allowing the subscriber to initiate a call to a third party. These two parties would then be joined into a single call after the subscriber generated another *flashhook* signal. In addition, when the subscriber of CW issued a *flashhook* signal, while already involved in a call and after receiving notification of a second incoming call, the CW feature would trigger, placing the other party of the current call on hold and connecting the subscriber in a voice connection with the party of the incoming call. The CW subscriber could switch back and forth between these two calls by using the *flashhook* signal. Ambiguity arose when a user subscribed to both CW and 3WC simultaneously and when the subscriber was involved in an established call and a second incoming call was received; if the subscriber issued the *flashhook* signal, both features would be triggered, however the actions of these features conflict and there is no way to determine the subscriber's true intention of how the signal should be used. This ambiguity can be resolved by the use of *feature-specific* signals, where a signal *CWSwitch* is used by CW to switch between two active calls and a different signal *3WCInit* is used by 3WC to initialize a call to a third party. Now when the subscriber issues a signal, the appropriate feature will respond. Thus, feature designers are encouraged to use feature-specific signals whenever a user generates a non-core signal to trigger feature execution or feature action.

Definition 2.4.3. Complete Signal Set:

The signal set used in this DFC-based telephony domain includes both the implemented DFC signals identified in Definition 2.4.2 and the extended signals set identified below:

answer: *this signal is used to indicate that the callee's end device was answered. This signal generally results in the formation of the voice connection.*

ringTO: *this signal indicates that although the call attempt was successful in reaching the callee, the call was not answered after a designated amount of time. Usually this signal is passed transparently through each module in the call path, unless a module is designed to treat failed call attempts, in which case the module will usually redirect the call attempt.*

feature-specific: *a feature designer may define new signals that trigger functionality only within a specific feature module. Feature-specific signals are implemented to remove signalling ambiguity, since only a specific feature module (or set of feature modules) can react to this signal: all other feature modules execute transparently when this signal is received.*

In addition to signalling information, the telephony system needs to maintain information relating to the call, so that it is able to correctly execute feature tasks and direct the call. For example, call-state information records the execution status of the call (e.g., whether a voice connection is established), feature data records subscriber-defined information used by the features (e.g., Call-Screening lists, subscription information), and system data records system-related information (e.g., billing, time, resource allocation).

We partition these data into two types: dynamic and static. **Dynamic data** pertains to the structure of the call and stores information such as the composition of the call (e.g., a feature has been added, a voice connection has been established), state information for each feature in the call, and the set of signals found in the message queues for each feature. **Static data** pertains to information that persists beyond the lifetime of a call, such as feature data (e.g., who subscribes to which set of features) and system data (e.g., billing records). Static data that is shared between features (e.g., alias information) or that is accessed by different instances of the same feature (e.g., Call-Screening lists) is stored in the telephony database, instead of being stored internally by the feature.

Definition 2.4.4. Dynamic data:

Dynamic data is data that frequently changes within the domain. In our DFC-based telephony domain, this data includes information related to the structure of the calls and call attempts, such as whether or not a module is instantiated and which call segment it is associated with, the status of the call attempt (a voice connection is established, the call is being torndown), and status information related to the active modules.

Definition 2.4.5. Static data:

Static data is data that does not frequently change within the domain. In our DFC-based telephony domain, this data includes information that persists beyond the lifetime of a call, such as feature data that determines who subscribes to which set of features and system data that identifies permanently recorded information such as billing details.

A feature box's execution is triggered by the receipt of a signal. When an input signal is received, the feature uses the input signal, together with the dynamic data pertaining to the call and static data associated with this feature box, to determine the appropriate response.

A feature module can behave transparently or execute one or more actions. A **transparent** execution propagates the incoming signal unchanged to the next feature in the call path without changing the state of the feature's CFSM and with no changes to the database data. This transparent execution gives the appearance that this feature does not exist and that a direct connection

exists between the feature module's neighbours. Alternatively, the feature can perform an action, such as modifying dynamic or static data, issuing new signals, or modifying the call state. For example, if the feature terminates a call attempt, then the call state is modified and a *teardown* signal is issued.

2.5 Feature Interactions in Telephony

Features interact when the combined execution of a set of features changes the execution of one or more features in the set. Given that each feature's execution can affect the telephony database, call states, and signals, these are the main elements that are monitored for the presence of feature interactions. For example, if two features both try to record data to a shared data variable in the telephony database, the execution order for this set of features may result in the call attempt reaching an undesirable call state (i.e., the data recorded by the first feature modifies the call attempt so that the second feature does not behave as designed), or the features may execute in an order such that undesirable data is recorded in the database (i.e., the data recorded by the second feature incorrectly overwrites the data recorded by the first feature).

Definition 2.5.1. Feature Interaction:

A feature interaction occurs when the behaviour of a feature in the presence of another feature differs from the behaviour of that feature run in isolation.

In Section 4.2, we discuss principles of proper system behaviour, which express desirable or undesirable call states, variable values, and actions. As a basic example, a call state connecting two users in a voice connection is undesirable if a feature asserts that this connection should be prevented (i.e., blocked). Similarly, it would be undesirable for a Return Call feature to record information about a call that should be blocked, because the user presumably would not want to return a call that he never wanted to receive. Section 3.1 gives a complete description of the different types of feature interactions.

2.6 Comparison with the DFC Architectural Model

In this section, the main difference between our DFC-based model and the original DFC architectural model are highlighted.

Different signal sets:

The core set of DFC signals also includes: *unknown* (target address is invalid), *upack* (signal used to acknowledge receipt of the *setup* signal), and *downack* (signal used to acknowledge receipt of the *teardown* signal).

unknown: this signal indicates that the routing address received by the network interface module is invalid. This signal indicates that the call attempt cannot be completed as dialled and that this call attempt should be terminated.

upack: this signal is used to confirm the receipt of a *setup* signal. Whenever a *setup* signal is input into a feature module, the *upack* signal is sent back to the module that generated the *setup* signal to confirm to the sending module that the communication between them has been established.

downack: this signal is used to confirm the receipt of a *teardown* signal. Whenever a *teardown* signal is input into a feature module, the *downack* signal is sent back to the module that generated the *teardown* signal to confirm to the sending module that the communication between them has been torndown, and that no more messages along that channel will be forthcoming. The module generating the *teardown* signal will not be removed from the call path until after the *downack* is received.

Our DFC-based telephony domain abstracts away the DFC protocols that are used to establish channels between neighbouring feature modules, since establishing these channels is orthogonal to feature behaviour in the domain. Thus, we do not implement the *upack* and *downack* signals, which are used solely for the purposes of establishing and tearing down these channels. The *unknown* signals are not modelled in our domain, since these signals represent an unrecoverable failed call attempt and thus will not trigger the functionality of any features.

It was also noted in Definition 2.4.2, that we chose to enhance the set of signals that were modelled within the DFC-based telephony domain. The enhanced signal set allows the domain to consider a wider range of feature behaviour and the *feature-specific* signals allows the domain to remove interactions caused by signal ambiguity.

Signal Implementation:

Unlike in DFC, the signal routing data is not stored as part of the signal, but rather this data is stored separately in a centralized call database. The routing data includes the region in which this module resides (source or target), the source and target addresses for the call, and the alias information associated with the address of the opposite region.

Signal Translation:

In the DFC architecture the interface modules are designed to represent the translation of the type of signals from the type issued by the device in the previous address zone to the type of signals used in the next address zone. Since our DFC-based telephony domain is not implemented at run-time we simplify our device-specific interface modules by assuming all modules issue and react to signals from the same set, so that no translations are needed when connecting the modules from different address zones.

Reversible Modules:

Several features need the ability to generate a call attempt on behalf of the subscriber, or to handle multiple connections simultaneously. When features that generate a call attempt have the ultimate goal of including the subscriber in the call, then the feature must be able to issue the *setup* signal in both directions to reach the parties that will be included in the call. Hence, these features behave as a member of the source region in one portion of the call and as a member of the target region in another portion of the call. In the DFC architecture, this functionality cannot be represented in a traditional feature box, thus reversible feature boxes were added to the architecture to support this functionality. The reversible feature boxes must be able to perform their tasks under a variety of circumstances, and hence these feature boxes must be implemented in both the source and target region. In DFC, all bound feature boxes and all feature boxes that are able to initiate or generate a *setup* signal in both directions are reversible feature boxes and are always added to the call path of a call that includes the subscriber, regardless of whether the subscriber is the caller or the callee for this call. Within our DFC-based telephony model, we assume that *all* feature modules are added to every address zone regardless of whether they are reversible and regardless of whether the subscriber is the caller or the callee on that call. Hence, all our feature boxes are found in both the source and target region, and we do not distinguish between reversible and non-reversible feature boxes. When a module is located in a region where its functionality is not executed, the module simply behaves as a transparent feature box.

Chapter 3

The Feature-Interaction Problem

This chapter provides an in-depth study of the feature-interaction problem, starting with a description of the different types of feature interactions (Section 3.1) and the various techniques used to help eliminate or reduce the problem. Specifically, Section 3.2 describes techniques for filtering the set of features to be analyzed for interactions, Section 3.3 discusses several methods for detecting feature-interactions, Section 3.4 explores interaction-prevention techniques, and Section 3.5 explains several resolution techniques. Section 3.6 focuses on research that determines priority orderings that are used in resolution techniques, and is hence closely related to the approach presented in this thesis. We conclude our survey of the feature-interaction problem in Section 3.7.

3.1 Feature Interactions

To reiterate a few relevant definitions, a **service** is defined as the core functionality of the system, and a **feature** is defined as any add-on functionality that extends the basic service. A **feature interaction** occurs when the presence of one feature in the system affects positively or negatively the execution of another feature in the system.

In the Feature Interaction Workshops [6, 16, 17, 30, 9], many of the authors use different definitions for various types of feature interactions. The following classification of feature interactions incorporates the majority of interactions defined by these researchers: **shared-variable conflicts**, **constraint violations**, **global-invariant violations**, **data modifications**, **user/feature interactions**, **resource contention**, **reachability**, and **race conditions**.

Shared-Variable Interaction

A **shared-variable interaction** occurs when two or more features simultaneously attempt to assign conflicting values to the same variable.

Example 3.1.1. Call Waiting vs. Voice Mail.

Both Call Waiting (CW) and Voice Mail (VM) want to change the call-state of an incoming call when the called party is already on the phone: VM wants to redirect the call to an automated answering service, while CW wants to establish the connection and allow the called party to switch between his current call and this new call. Because both features want to update the new call's call-state and their proposed call-state values conflict, a shared-variable interaction exists.

Constraint-Violation Interaction

A **constraint-violation interaction** occurs when the action of one feature violates a constraint asserted by another feature. A constraint represents a restriction placed by a feature on the system to ensure that the feature's requirements are met. In the example below, the Terminating Call-Screening feature asserts a constraint that states that Mike should not receive calls from Sally.

Example 3.1.2. Terminating Call Screening vs. Operator.

Mike blocks calls from Sally using the Terminating Call Screening (TCS) feature. However, Sally asks the Operator to setup a call that connects her with Mike. The Operator does not have access to Mike's call-screening list and will likely place the call, bypassing the TCS feature.

Global-Invariant Interaction

Global-invariant interactions are similar to constraint violations, the difference being that global invariants are properties that are asserted in a system's initial state and are expected to persist throughout the duration of an application's execution. As an example, a global invariant may prohibit users from participating in more than one active call at a time. (Note that we model conference calls as a single active call, and the Call Waiting feature allows a user to switch between two active calls.)

Data-Modification Interaction

A **data-modification interaction** occurs when a feature reads and reacts to a shared variable value that has been modified by another feature. For example, the execution of one feature modifies the system environment, such that the second feature cannot be activated.

Example 3.1.3. Call Waiting versus Automatic Call Back.

Automatic Call Back (ACB) repeatedly queries a busy line and notifies the subscriber via a special ringback when the line becomes free. Call Waiting (CW) alerts the subscriber to a second incoming call when the subscriber is already on the phone. Suppose Alice subscribes to CW and is speaking with Charlie, when Bob, who subscribes to ACB, calls. Because Alice's CW feature alerts Alice to the new incoming call, Alice's line is not deemed busy. If Alice chooses not to accept the new call, Bob does not know to activate ACB. Even if Bob did activate ACB, ACB would treat the line as being free and would do a call-back immediately.

User/Feature Interaction

A **user/feature interaction** occurs when the behaviour of a feature conflicts with the behaviour that the user expects (or would likely prefer). For example, user/feature interactions can occur when two or more subscribers use the same physical device, but subscribe to different features. In a user/feature interaction, the features behave as specified: in some situations, users may be happy with the features' behaviour, and in other situations, users may not be happy. As an example, a Hotel Room Calling feature bills callers for a call after a call attempt has been initiated for 10 seconds, whether or not the call has been answered.

Example 3.1.4. Call Waiting vs. Personal Communication Services.

Two users (Alice and Bob) share the same physical phone, but have different phone numbers. Alice subscribes to Call Waiting (CW), but Bob does not. When Bob is on the phone and a new call arrives for Alice, does Alice's CW feature interrupt Bob or does Bob being on the phone override Alice's features? Alice and Bob may have conflicting opinions about what is the correct behaviour.

Resource-Contention Interaction

A **resource-contention interaction** occurs when the total number of requested resources exceeds the number of resources available, and prevents two features from executing at the same time.

Example 3.1.5. Call Waiting vs. Three-Way Calling.

A resource contention occurs between the Call Waiting (CW) feature and the Three-Way Calling (3WC) feature in land-line telephones because both features need access to a hardware device known as a bridge to establish a connection with the third user. Each land line has only one bridge, so both features cannot be active simultaneously.

Reachability Interaction

A **reachability interaction** occurs when the call reaches a state (or execution loop) that it cannot exit, preventing the call from progressing.

Example 3.1.6. Call Forward vs. Call Forward.

Consider the situation where both Bob and Sally subscribe to Call Forward (CF). When Bob leaves the office, he forwards all incoming calls to Sally. In the meantime, Sally sets her CF to forward calls to Bob while she is in a meeting. When John calls Sally, his call is constantly forwarded back and forth between Sally's number and Bob's number by the CF features; therefore John will not be able reach a state where he can successfully complete his call, creating a reachability interaction.

Race-Condition Interaction

Race-condition interactions occur when (1) two features access the same variable, (2) the order of the accesses depends on user actions or processing speeds, and (3) different orders produce different results. Race conditions are similar to data-modification interactions, in that one feature reads a system state that was changed by another feature. The difference is that there are several possible behaviours in a race condition, because different features can “win” the race in different situations.

Example 3.1.7. Automatic CallBack vs. Automatic Recall.

When activated, Automatic Recall (ARC) will dial the number of the last incoming call. If the line is busy, ARC will continuously test the line. When the line becomes free, a ringback will be issued to the subscriber. When the subscriber answers the ringback, ARC redials the number in an attempt to establish a connection. Consider the situation where Bob subscribes to Automatic CallBack (ACB) and Sally subscribes to Automatic ReCall (ARC). When Bob calls Sally and she is on the phone, he activates ACB. When Sally finishes her call, she chooses to initiate ARC. If Bob’s ACB and Sally’s ARC activate at the same time, then ACB can keep initiating new calls to Sally and ARC can keep returning calls to Bob. This repeated race condition could prevent either feature from receiving a free line status, leading to a reachability interaction. (Note: this is unlikely to occur simultaneously.)

3.2 Filtering

Filtering is a technique used to reduce the number of subsets of features that need to be tested or analyzed for interactions. The goal of filtering is to identify subsets of features where interactions are known or suspected to occur. Without filtering, the amount of work required to detect feature interactions in a system is often exponential in the number of features: each feature must be tested in combination with all possible subsets of the existing features in the system to ensure no interactions exist. Thus, filtering is used to identify subsets of features that do not have interactions, thereby reducing the number of test cases and the amount of analysis that needs to be performed and improving time to delivery.

The main idea behind filtering is to represent features at an abstract level. Subsets of abstract features are then tested to determine if (1) an interaction is definitely possible, (2) no interaction is possible, or (3) the possibility of an interaction is unknown (interaction-prone). From this test, only the sets of concrete features that are interaction-prone need to be further tested by a more thorough feature-interaction detection technique, making the detection process more efficient.

Despite the advantages of filtering, errors can occur if the features are represented incorrectly or if the abstraction level of the features is too high. In such a case, filtering returns false-positive or false-negative results. False positives occur when two or more features are marked as having an interaction or being interaction-prone when no interaction is possible. When too

many false positives are returned, the filtering is not efficient: this may cause an explosion in the number of feature sets that are analyzed in the detection stage. A false negative occurs when the filter determines that an interaction is not possible when in fact an interaction can occur. When false negatives are found, the filtering tool prevents interaction-prone features from ever being analyzed by the detection tool.

Filtering is most effective when applied at design-time in conjunction with a design-time interaction detection technique. The designers can test for interactions and modify the design or code to correct any interactions found.

3.2.1 Use Case Maps

To aid in the feature-interaction problem, Use Case Maps (UCMs) have been used as a filtering tool. Nakamura et al. [37] chose UCMs as their filtering tool for three (3) reasons: (1) UCMs visualize global call scenarios at the requirements level, and exhibit no information about the detailed system behaviour or complex semantic models; (2) UCMs have adequate characteristics, such as concurrency, alternative and hierarchical design, to describe features at the requirements level; and (3) a tool called UCM Navigator has been developed that helps designers to draw syntactically correct UCMs [37].

A UCM, called a submap, is created for the basic telephony system and for each individual feature. For each set of feature pairs, the submaps of the features are combined. The compositions of the submaps are then analyzed to test for interactions. If a conflict is found in the same stub (a specific place within the UCM scenario) of the composed feature submaps, then a feature interaction is detected. If the combined feature submap is compared against the feature submaps before composition and no change is detected, then no feature interaction has occurred; otherwise, the features are assumed to be interaction-prone. The interaction-prone feature pairs need to be further analyzed to determine whether they can in fact interact.

3.3 Detection

Feature-interaction **detection** techniques identify feature sets that can interact when the features are combined in the system. If a feature interaction is not detected, then it cannot be resolved and the resulting system will encounter situations where unanticipated or incorrect actions may occur. Such a system does not meet all of its specifications. In this section, a selection of feature-interaction detection techniques is described. These techniques are generally used in off-line analysis. Results from a detection analysis can be used to demonstrate that the system works correctly or to identify potential conflicts that need to be resolved.

3.3.1 Requirement-Level Techniques

A **requirements model** describes users' expectations (e.g., user goals with respect to functionality and implementation) of the features and the underlying service in terms of observable events. An accurate requirements model combats the feature-interaction problem since the consumers' desires are clearly communicated to the software developers, who can then adjust the software to address these concerns. For example, a requirements model can be analyzed to determine how different features respond to the same external event; such analysis can help developers detect an unexpected interaction.

Gibson argued that many feature interactions occur because a poor requirements model was used [21]. Gibson lists ambiguous signals (i.e., the *flashhook* signal is used by more than one feature) and restrictive assumptions as some of the problems in a bad requirements model. Gibson then proposes using formal mixed-method semantics to create the requirements model. Careful analysis of feature requirements during construction of the requirements model will limit the application of restrictive assumptions (e.g., Caller Id reveals the *telephone number* of the caller, which may restrict the feature's ability to be updated to reveal the *name* of the caller) or force these assumptions to be made explicitly. Once the requirements model is complete, features can be compared. If an interaction is detected, an appropriate resolution can be devised and added to the requirements model.

CHISEL and its editing tool SCF3TM/Sculptor [2] form a method for defining requirements for communication services. The tool is designed to aid in the creation of better requirements and to translate requirements specification into some commonly used formal languages (e.g., Message Sequence Charts and process algebras) for verification and testing. Feature requirements written in CHISEL are tested for interactions by merging the features and determining whether the resulting combination is consistent with respect to how the features behave in isolation. If the combined features' behaviour is found to be consistent, then the feature set is interaction-free; otherwise an interaction is present.

3.3.2 Formal-Methods Techniques

Formal methods are mathematics-based techniques for modelling and verifying systems. In the area of feature interactions, formal methods are used to help prove the absence of unwanted feature interactions or to detect feature interactions.

Feature-interaction researchers have used many types of formal methods, such as finite state machines (FSMs), Petri-nets, LOTOS, and Promela/SPIN. Below, we will describe some of the formal method techniques in terms of feature-interaction detection.

3.3.2.1 Finite State Machines

Finite state machines (FSMs) can be used to represent the different features in a system [50]. Each feature is implemented independently, applying the concept of separation of concerns. The FSMs are then composed and analyzed to discover patterns, which represent the different types of interactions that can occur. These patterns form a pattern language for feature interactions.

Fritsche [20] uses Generic Finite State Machines (GFSM) to resolve conflicts among features. A GFSM models all aspects of call control and monitors the system for the presence of trigger events (changes in the connection), feature activation messages, or external events (changes in device status). When multiple features are activated by the same event, the GFSM is then able to detect and resolve feature interactions by evaluating the effects of the features' enabled state transitions.

3.3.2.2 LOTOS

Language of Temporal Ordering Specifications (LOTOS) is a formal specification language that has both an algebraic component and an abstract-data-type component. LOTOS is a language used by researchers to formally specify features and detect feature-interactions, usually at the specification level (see [4, 19, 22, 46]).

LOTOS has also been used to implement constraint-/knowledge-oriented specifications [19]. The constraint-oriented specification style comprises three levels of constraints: (1) local constraints confined to a single telephone, (2) end-to-end constraints that apply to a phone connection, and (3) global constraints that hold in the entire system. The knowledge-oriented portion of this specification incorporates for each feature a set of knowledge goals that are specified as LOTOS processes. The knowledge goals are determined by the designer and identify the goal(s) of the feature with respect to the local point of view of the feature. Once the knowledge goals are written, the specification is simulated to check that a path leading to the knowledge goal is reachable. Non-reachable goals identify the presence of an interaction or a design error.

LOTOS can also be used with UCMs to detect feature interactions [4]. The features are designed at a high level of abstraction as UCMs. The UCMs are then refined manually into LOTOS specifications, which are at a lower level of abstraction. The refinement to LOTOS often reveals missing or ambiguous information in the original UCM models. The formal LOTOS specifications can then be verified and feature interactions can be detected using reachability analysis, model checking, or event simulation.

3.4 Prevention

Prevention techniques avoid interactions by coordinating features such that interactions are prevented from occurring. For example, a system that is designed so that features are executed

in series avoids interactions due to features reacting to the same input event at the same time, because the order in which features receive and react to the event is serialized. The first feature in the series receives the input: if the first feature does not react to the input event, then the event is passed to the next feature in the series. However, if the first feature reacts to the input event, that feature has a choice as to whether or not to pass the input event to the rest of the features in the series. When the first feature chooses not to pass along the input event, the second feature never receives the input event, and hence resolves the possible feature interaction that would occur if both features responded to this event.

Other researchers have made different architectural or design decisions that limit or prevent specific types of feature interactions from occurring in their systems. Prevention techniques can be architectural techniques, however their implementation differs slightly from the design-time architectural resolution techniques discussed in Section 3.5. The subtle difference is in the manner in which both techniques resolve interactions. The prevention-based architectural techniques impose a resolution to possible interactions, while design-time architectural techniques are employed to find a resolution only after detecting an interaction. In the above example, the serialized architecture is a prevention approach that prevents the manifestation of feature interactions. The prevention techniques may require information gathered from resolution strategies, such as the correct priority orderings for the features, to define correct system behaviour.

3.4.1 Distributed Feature Composition

Distributed Feature Composition (DFC) [27] is a pipe-and-filter style architecture that supports modular feature development and composition. Each feature is defined independently and is implemented as a feature box. A call is represented as a series of feature boxes that communicate through signal channels. When a signal is issued, it passes through each feature box in the call path; a feature box may choose to react to a received signal or to ignore it and pass it along to its neighbouring feature box.

When a feature box reacts to a signal, the behaviour of the system is modified. The feature box may choose not to send the signal any further along the call path, which prevents other feature boxes from reacting to the signal. The feature box may choose to perform an action, or to send out a new signal to modify the system's behaviour. Since each feature is implemented independently and the feature boxes are composed sequentially, it is easy to modify the set of features that are involved in a call.

DFC call and switch call protocols [27] are used to force features into a specific configuration, which determines the resolution to interactions. DFC is capable of resolving most of the following types of interactions: shared-variable interactions, since only one feature can respond to a particular signal at a time and thus there is little chance of conflict when assigning values

to the database¹; constraint-violation interactions, where one feature violates a restriction placed by another feature; global-invariant interactions where, the global invariant represents a property that should hold throughout the system; data-modification interactions where the execution of one feature changes a shared variable value so that another feature cannot execute; resource-contention interactions, where multiple features cannot execute at the same time due to a limited number of system resources; reachability interactions, where feature execution reaches a loop or call state where no further progress can be made to establish a call connection; and a small number of other types of feature interactions (i.e., user/feature interactions and race conditions).

There are several advantages to the DFC architecture. Non-linear calls (e.g., multiple-party calls, feature-initiated calls) can be handled and represented easily in DFC. The composition of features keeps a clear separation between feature-defined behaviour and feature composition. Individual feature behaviours and interactions between features can often be analyzed using model-checking techniques, although race-condition, and user/feature interactions cannot be easily represented or detected during analysis. Jackson and Zave used SPIN to test for interactions between protocols for the caller/callee and the DFC switch protocols [27]. Hall used a modular, abstract formalism, which was inspired by Jackson and Zave's work [27], to detect email feature interactions [25].

3.4.2 AIN

The Advanced Intelligent Network (AIN) architecture was developed by Bell Communications Research as an architecture that separates service logic from the plain old telephone switching. AIN has become recognized as an industry standard in North America, and the International Telecommunications Union (ITU) has created an equivalent architecture in other countries. Removing the service logic from the switching software has allowed new features and services to be added to the system without costly redesigns to the switching network. This has also allowed for vendor independence, as service providers can now add new services easily and offer their customers more choices, thereby increasing competition.

Researchers [12, 5, 11] have studied various aspects of the feature-interaction problem in the context of the AIN architecture. Cameron et al. [12] created a formal model of the AIN architecture and used it to analyze and manage feature interactions in the AIN environment. Blumenthal et al. used AIN as the base architecture in their feature-interaction tool [5]. Cameron et al. have also developed tools for use in creating AIN services [11].

¹The feature's local data is not shared, so the risk only occurs when multiple features override the same data inside the shared database. However, much of this data is feature specific and thus, simultaneous updates should not occur.

3.4.3 SIP

Session Initiation Protocol (SIP) is the Internet Engineering Task Force (IETF) protocol for Internet telephony, which runs as an end-to-end, client-server signalling protocol. SIP is designed to be easily programmable so that features and arbitrary services can be added and combined. This ease of implementation also means that users have the ability to customize features to match their needs (for more details see [32] or [42]).

SIP was designed to work as the basic protocol for signalling in Internet telephony. SIP is a flexible signalling protocol that initiates, manages, and terminates voice and video connections. SIP can be extended to manage and resolve feature interactions caused by signalling ambiguity. For example, both Call Waiting and Three Way Calling traditionally respond to the same *flashhook* signal; however, using feature-specific signalling in SIP prevents this problem from occurring. This ability is a prevention-based approach to the feature-interaction problem.

There are other aspects to SIP that also need to be considered, such as new feature interactions that have emerged with the creation of Internet telephony. For example, there are trust issues in Internet Telephony because it is easy to change the “number” associated with a call. This creates an unexpected interaction with Call Screening, since this feature cannot work as effectively as it does on the land-line telephones. Design rules, which are part of a feature interaction resolution strategy, are being created so that SIP features can be designed and implemented with minimal feature interactions. For example, new SIP standards are being developed to define parameters that can be used to resolve feature interactions [34].

3.5 Resolution

Resolution techniques determine an acceptable integrated behaviour for a set of interacting features. After a feature interaction is detected, it should be resolved to ensure proper working of the system. There are many issues involved in determining the best resolution for an interaction. Should one feature prevent the execution of another? Should features try to work together? Is there one correct resolution or will different customers want different resolutions? Should the resolution be determined by the situation in which the interaction occurred? Researchers working on the feature-interaction problem have been trying to determine solutions for this complicated problem.

There is also the question of when resolution should occur: statically or at run-time? Static resolution is a pre-determined solution that is usually hard-coded into the software. This means that feature interactions are automatically resolved, because the software has been designed to implement the resolution of interactions in a specific manner. The major limitation of static resolution is that the customers are forced to accept a specific resolution determined by the vendor, thereby making it harder for the customer to personalize how features should work together. An-

other limitation is development time: each new feature that is added to the system needs to be tested against all of the existing features for interactions.

In contrast, detecting and resolving feature interactions at run-time means that the system must constantly test a call's progress to prevent interactions by resolving them before they can occur. This constant testing can affect performance. For example, there may be an increased delay between the time a connection request is sent by a caller and the time it is received by the callee. In addition, the system may not be able to detect and resolve all interactions at run-time.

Most of the resolution strategies below can be applied statically or at run-time. It is also possible to incorporate a combination of static and run-time methods to create yet another resolution strategy. Below we alphabetically explore a selection of different resolution strategies.

3.5.1 Agent-Based Architecture Detection and Resolution

In **agent-based architectures**, features are represented as agents and these agents are monitored so that feature interactions can be detected and resolved. Several agent-based architectural environments and languages have been proposed to combat the feature-interaction problem, such as Multi-Agent Architecture for Networking Applications (MANA) [51] and Cognitive Agents Specification Language (CASL) [45].

In general, an agent-based environment defines an architectural style in which each feature unit is represented as a separate agent. These agents are able to communicate, either directly or indirectly, with other agents. In the feature-interaction domain, each feature is represented by a different agent. It is possible for a feature to be represented by more than one agent. Each agent knows what the "goal" of the feature is: that is, what the feature wants to achieve when it is active. The agent also knows how to perform the actions required by the active feature or is able to command subordinate agents to perform them.

When a feature is activated, it informs other features (through various methods, such as the use of a blackboard [8]) of its intention to act by asserting the goals it wants to achieve. If a new goal violates an already asserted goal, then a feature interaction is detected. Agent-based systems are best able to detect constraint-violation interactions. Other types of interactions, such as shared-variable interactions and resource-contention interactions, may also be detected if a feature's goals include descriptions of variable modifications and a list of resources required for execution.

Once an interaction has been detected, the agents that posted the conflicting goals use a form of negotiation to resolve the interaction. The essential goals of the features are used as the basis for finding an acceptable resolution for all features involved. For example, MANA uses off-line analysis of resources to prepare for negotiation over resource-constraint interactions [51]. As another example, a mediator can work with conflicting agents to determine a resolution [23].

3.5.2 Fuzzy Logic

Fuzzy logic is a logic system that allows truth values of properties to be imprecise. Fuzzy logic is used in systems where the input data are continuous in nature, and need to be translated into discrete values for use in calculations. Each property is expressed as a degree of truthfulness or falsity, represented as a value between 0 (completely false) and 1 (absolutely true). Fuzzy logic has three defined operators: **and**, **or** and **not**. The definition of **not**(x) is accepted to be $(1 - \text{truth value}(x))$; however, the definitions of the other two operators are often modified to suit the needs of the researcher. Commonly, **and**(x,y) is defined as the minimum truth-value of x and y, and **or**(x,y) is defined as the maximum truth-value of x and y.

Fuzzy logic has been widely used in artificial intelligence to represent systems that use vague or imprecise knowledge, such as linguistic approximation and expert systems [7]. See [7] for more details, including a brief history of fuzzy systems, a basic tutorial, and a small analysis of the advantages and disadvantages of fuzzy systems.

Fuzzy logic has been applied to feature interactions as a means for determining the most appropriate solution to a given conflict. For example, weighted priorities in the form of fuzzy policies can be used to resolve feature interactions [3]. As an example, consider the situation where the caller has a Do Not Forward value of 0.5 and the callee has a Call Forward feature with a value of 0.9. When these two features interact, the call is forwarded since Call Forward has a higher truth-value.

3.5.3 Negotiation

Negotiation is a run-time resolution technique that resolves feature interactions by negotiating a resolution that will best satisfy all features involved. There are three types of negotiation that can be used: direct, indirect, and arbitration.

Direct negotiation occurs directly between the features involved. The features progress through a series of proposals and counter-proposals until an agreement is reached. **Indirect negotiation** occurs between the involved features and a mediator. The mediator passes proposals and counter-proposals back and forth between the features. By being in command of this transfer of information, the mediator is able to ensure that progress is being made in the negotiation. All of the features and the mediator are able to suggest a resolution to the conflict. In **arbitration negotiation**, the features inform the mediator of their intentions and the mediator then becomes responsible for determining a resolution to the conflict.

When indirect or arbitration negotiation is used, the mediator can be specialized so that it has experience with the type of conflict that is being resolved. The mediators can learn the best way to resolve specific interactions from past negotiations, and can subsequently reach an acceptable resolution quickly.

Griffeth and Velthuisen [23] use agents to represent the users of a telecommunications system; these same agents are also responsible for negotiating a resolution when interactions between the users' features occur. Agents identify the goals of the features, and the negotiation mechanism uses the goal hierarchy to determine which proposals are acceptable and which are unacceptable. For example, a basic goal is to form a call between the end-users, which is represented as $call(x, y)$. The goal hierarchy identifies the different ways in which this goal can be reached. For example, a $call(x, y)$ is formed if there is a connection between the end-devices relating to the users, which may or may not require a pass-key to connect the users, such that $call(x, y)$ is satisfied if $connect(x, s, y, t)$ or $keyConnect(x, s, y, t)$. By exploring the different options in the goal hierarchy, the negotiation mechanism is able to identify and propose different possible resolutions to feature interactions.

3.5.4 Priority Schemes

A **priority scheme** resolves feature interactions by using priority values assigned to each feature to resolve conflicts between competing features that each want to react to the call situation. In its most basic form, when a feature interaction is detected, the feature with the highest priority is chosen for execution. We begin this section by exploring general approaches to applying priority schemes as a resolution technique and then explore actual priority schemes implemented by other researchers. We discuss other related work, including approaches to determine priority orderings, in the final section of this chapter.

There are several possible implementations for a feature-interaction resolution priority scheme. For example, we can use the basic priority scheme described above, where priority values are determined by the creators of the features and are enforced statically by the system. This method uses the priority values of the features, which are shared by all subscribers to the features, and may result in feature-interaction problems when the priority values of features from one vendor do not work when combined with the priority values assigned to features designed by a competitor.

Other methods use a priority scheme to resolve interactions when the initial resolution strategy fails. For example, in [48], the initial resolution strategy counts the number of conflicts each feature has and chooses the feature with the smallest number of conflicts for execution. If there is a tie for the least number of conflicts, then feature priorities are used to break the tie.

A priority-based resolution strategy is both easy to understand and implement once the priorities have been determined. However, difficulties can arise when trying to determine the correct priority value to assign to a feature. If the value is incorrectly assigned, then the resolution determined by the algorithm will not be acceptable to customers.

Priority-based resolution schemes can be applied at multiple levels. Below, we discuss strategies that are the result of architectural design decisions and schemes used during the design and

implementation stages to resolve feature interactions.

Architectural approaches to the feature-interaction problem involve using the structure of the underlying service as a method to reduce the number of feature interactions. In some situations, these architectures are designed to work in combination with a priority-based resolution scheme. One such architectural approach is AT&T's Distributed Feature Composition (DFC) architecture for telephony systems [27]. DFC serializes features in a pipe-and-filter style architecture, where features are added one-by-one to a call attempt. Each pair of features is considered to determine whether or not a feature interaction exists. If both orderings generate the same behaviour, then no feature interaction exists between the pair of features and either ordering is acceptable. However, when the two orderings result in different behaviour, an interaction is identified and the feature designer must determine the correct ordering for this feature pair and assign priority values to the features accordingly. As another example, Zibman et al. [54] use precedence rules together with an agent-based architecture to resolve interactions using priorities. This architecture distinguishes between various roles within the telephony system. For example, a user is separate from his end device: multiple users (e.g., family, technical support groups) can access the same end device, and a single user may access several end devices (e.g., home phone, work phone, cell phone). This rule-based architecture is combined with a processing model that monitors for feature interactions that occur as violations of feature or service assumptions (e.g., new hardware services remove the need for existing assumptions) or due to role confusion. Feature interactions are detected and resolved through the use of precedence rules by determining which events or messages have priority.

In design-stage approaches to feature-interaction detection, researchers use various techniques to identify feature interactions and then use feature priorities to resolve these interactions. Nakamura and Tsuboi [38] use a frame model, which is a theory that uses data structures to represent and construct knowledge about features during the design stage, for the purposes of detecting and eliminating feature interactions. In this work, each feature is expressed as a frame that contains slots that hold (or will hold) values relating to attributes (e.g., source and target address) about that feature and call. The modelled features are then composed, analyzed, and evaluated with respect to three types of feature interactions: **exclusive interactions**, which are equivalent to shared-variable interactions; and **connective and recursive interactions**, which are both similar to data-modification interactions where a variable assignment causes the execution of another feature or modifies the expected output of an already executing feature, respectively. When an interaction is found, the feature designer is prompted to eliminate the interaction by choosing the most appropriate ordering and the system stores this solution as a feature priority to be enforced.

As another example, Harada et al. [26] define features in the service description language Standard Transition Rule (STR). In this work, a feature that does not depend on the target system behaviour is described from the user's viewpoint, so that the feature is modelled as both a

description rule and a set of implementations that describes the feature's behaviour. A feature interaction (aka service interaction) is detected between a pair of features if the applicable actions and conditions that form the implementation of the feature's behaviour results in transitions, one for each feature, that contradict each other (i.e., they are both able to execute but have conflicting results). When a feature interaction is detected, the designer is asked to verify the interaction, at which point the designer can prioritize the conflicting transitions, or use another technique to resolve the interaction.

Kawauchi and Ohta [29] also used STR when they created a mechanism for three-way interactions and proposed a detection system to identify feature interactions among sets of three features. The authors analyze cases of three-way interactions where two of the three features do interact, but this interaction is not apparent until the third feature is added (e.g., some functionality of the two interacting features is blocked unless a third feature is present). The features are modelled as a set of three rules: application rules, which define the event and pre-conditions necessary to trigger this feature's functionality; precedent rules, which are used to determine the feature that is executed when multiple application rules are satisfied; and state change rules, which simulate the execution of the feature. This approach uses prioritization in two ways: all feature pairs are assumed to be correctly ordered via a priority ordering, and the precedence rules give different features priority when determining execution.

Thistle et al. [47] also use prioritization in their work on supervisory control theory, where the supervisors are composed together to form the behaviour of the system by controlling the actions of features. In this work, the authors add a reporter map to the supervisory control theory, which acts as a filter to erase event streams that are observed by the supervisor: the erased or filtered event is not disabled within the supervisor. When the supervisors are joined, the reporter maps are placed between the source of events and the supervisor, which effectively weakens each supervisor's control of its feature, because the erased event is not passed through the reporter map to the supervisor. Consequently, some supervisors, which could have reacted to the event do not notify the controller, are not considered when the controller makes its decision as to which supervisor will respond to the event. Thus, the reporter map can impose a priority ordering between the supervisors with respect to events.

As a final example, Utas [49] proposes the use of patterns to identify and resolve feature interactions during the implementation stage. Each pattern represents a set of similar interactions. Once the interaction is detected and resolved for a pattern, the identified resolution is applied to all other feature sets in which a feature interaction is detected that matches this pattern. Some of the patterns use feature priorities to resolve interactions.

3.5.5 Rollback

Rollback is a run-time resolution strategy that allows features to run whenever they are triggered; when an interaction is detected, the rollback procedure reverses or “rolls back” the call to a state before the interaction first occurred and then uses resolution to determine a proper course of action for the system and the interacting features to take next. The concept of rollback is borrowed from transaction-processing theory, and is best applied in systems where the occurrence of a feature interaction is rare.

Marples and Magill use a Feature Interaction Manager (FIM) to monitor and control the rollback of features [36]. Each execution step in their system begins with an event being passed to a stable state; the features are then able to react to the event and the system moves into an unstable state. Features continue to react to the event or the actions triggered by the event until either there are no more features willing to react to the event and the system returns to a stable state, or until a loop is detected. In this work, a feature interaction occurs and is detected whenever multiple features react to a single stimulus. When an interaction is detected, the FIM rolls back the system to the stable state immediately before the interaction occurred. The FIM then generates a tree of possible outcomes in response to the event that led to the interaction. The possible outcomes are examined to determine the most appropriate resolution. If an appropriate resolution can be determined automatically, then this is applied and the system proceeds. Otherwise, the user is queried to determine which resolution should be applied.

3.6 Related Work

This thesis focuses on using prioritization as part of a resolution technique. This section describes methods for determining priorities of a set of features.

An intuitive way to determine feature-priority values is to allow users to determine the priority of each feature, and the system dynamically resolves interactions according to the provided priority scheme. User-defined priorities allow the user to decide how she would prefer feature interactions to be resolved. There are different methods for specifying user-defined priorities. One method is to assign a default set of feature priorities and allow each user to modify this set as desired. Another method is to query the user the first time a new interaction is detected. The user’s preference is then stored and used to resolve future occurrences of the same interaction. However, this is not truly an option in feature-rich domains where there are a large number of feature-interactions and where incorrect prioritization of the features could disable system functionality. Hence, more systematic approaches are required.

The technique introduced by Tsang and Magill in [48] is most closely related to the approach studied in this thesis. Tsang and Magill’s technique determines feature priorities dynamically by employing a run-time feature-interaction manager. Whenever a trigger event that activates the

feature’s main functionality is received, the feature-interaction manager analyzes the different feature orderings to predict/detect whether an interaction is about to occur. If there exists a feature ordering that does not result in any interactions, then this ordering is chosen and the features are prioritized accordingly. However, if all feature orderings result in an interaction, then the ordering with the fewest number of constraint conflicts is chosen, with a static priority scheme used to break ties. The notable differences between Tsang and Magill’s technique and our approach are:

- Tsang and Magill’s technique is designed to promote to **feature privacy** (aka service privacy) [48]. The issue of feature privacy is a concept that arises in Internet telephony, in which users can subscribe to features from multiple service providers and expect these features to work together. Service providers want to keep the “inner” workings of their features private, making it difficult to analyze feature interactions between features created by different companies. To accommodate feature privacy, Tsang and Magill designed their technique to work solely on observable feature behaviours. The feature interaction manager detects all feature interactions by considering the properties that define the feature’s observable behaviour (e.g., generated output signals and resource states, such as connection status and ring state). If the predicted behaviour deviates from an individual feature’s expected behaviour, then a feature interaction is found. While this decision allows integration of features from multiple vendors, it also limits the types of interactions that Tsang and Magill can identify during runtime analysis. For example, Tsang and Magill’s detection algorithm cannot detect when a feature inappropriately records or stores data, as this is a private event known only to the feature. Our approach allows a limited degree of feature privacy in that all vendors must categorize their features using the same set of categories. However, we do not require specification or implementation details to determine prioritization of the categories, since our categories are an abstract representation of the features found within the category. Our approach fails to allow for feature privacy, in our assumption that features will be modular and designed with a specific purpose, such that more complex features will be separated into multiple feature modules that are ordered independently, but work together through feature-specific signalling².
- Tsang and Magill’s technique was only tested on feature sets of at most size four, while our categorization approach is capable of ordering much larger sets.

Cattrall et al. [13] present a call-processing model that is designed to avoid feature interactions. A call is broken down into different roles and subroles inside which features are applied. A role

²When ordering features internal to a category, the designer can choose whether or not to accommodate for feature privacy, a decision at this level not to respect feature privacy will negate the ability to allow for feature privacy at the category level.

represents a user, a group of users, or, even more specifically, the life-role of a user (e.g., work role, family role, team captain role); and a subrole represents the distribution of a call (e.g., the end device used, a specific user within a group). The relationship between the roles and subroles automatically define a priority ordering that allows call attempts to better satisfy user intentions by distinguishing between network users and their equipment (i.e., redirect this call to me, *roleA*, at my new location, *subroleB*). By redesigning the call architecture, to separate users from end devices, some interactions can be avoided as roles and subroles assert their priority to control the call and the features executed during the call attempt. This concept of roles and subroles is closely related to the Ideal Address Translation principles [52] that we use in this thesis to combine different address zones as the call attempt progresses through different user feature sets, either through normal call progress or via a call redirection.

Elfe et al. [18] determine feature priorities as part of their detection and resolution algorithm, which tests for constraint violations. Pairs of features are tested, and when a constraint violation is found, the feature ordering is reversed and retested. If the alternate ordering does not result in a violation, this ordering is selected and the feature pair is prioritized accordingly. When both feature orderings cause constraint violations, the decision is referred to the feature designer who is asked to choose the correct ordering. We use a similar approach when implementing our pairwise optimization techniques. We begin our prioritization of feature categories by first testing pairs of feature categories to detect principle violations. If a principle violation is found with respect to both orderings of the pair, we choose the ordering with the fewest number of violations as the correct ordering. The results of the pairwise comparison are used to reduce the number of full-sized feature category orderings that need to be explored.

Another example of a prioritization technique is Chen et al.'s [15] work on supervisory control theory, which uses functions to dynamically determine a partial ordering among supervisors to prevent blocking. The authors consider modular feature development and represent feature requirements as finite state machines, where the supervisors for each feature are composed to form the system. A priority function is associated with each individual supervisor and is used to control the actions of the supervisor in case of interactions. This scheme is called modular control with priorities. After introducing this scheme, the authors describe four algorithms that can be used to assign priority values to the supervisors' priority functions. The first two algorithms are for mixed priority functions that are designed to prevent blocking in live-lock free systems (i.e., the execution can always reach a stable state) between supervisors. The first algorithm is iterative and uses a cost function, based on the sum of the cost values along a trace to a stable state, to determine priorities between blocking entities by choosing the trace with the minimum cost. The second algorithm is non-iterative, and works by identifying deadlock states and assigns priority values to events that will resolve the deadlock. The last two algorithms focus on designing the priority function to reflect the dominant supervisor. These algorithms differ in their assumptions

on whether or not the state space of the supervisors is the same. In both cases, the priorities are assigned so that the dominant supervisor has priority over the other supervisor whenever a conflict occurs.

The techniques described here for prioritizing features can be used in conjunction with our approach to determine the priorities for intra-category features. While these techniques face several limitations, such as testing only pairs of features [18, 15] and resolving only a limited class of feature interactions (e.g., those relating to blocking and redirection) [13], the biggest limitation is the high cost of calculating priorities for a large set of features.

3.7 Summary of the Feature-Interaction Problem

The feature-interaction problem is a difficult issue faced in feature-rich domains, such as telephony. Researchers have spent decades working on different approaches (e.g., filtering, detection, prevention, resolution) to minimize this problem. There are three places where interactions can be dealt with: at the features' design stage, in the infrastructure that deploys the features, or during feature execution. These three places correspond respectively to detecting and resolving feature interactions in the design stage, in the coding stage, or at run-time.

During any stage of development, prevention (avoidance) is the best strategy for resolving conflicts. Prevention reduces the number of conflicts that can occur, and hence decreases the number of interactions that need to be found and corrected. However, some interactions cannot be avoided and must be both detected and resolved for the system to work correctly. Specific management methods, such as those described in Sections 3.3 and 3.5, can be used to detect and/or resolve feature interactions that cannot be avoided.

The feature interaction problem is a large problem that is being studied by many researchers [6, 16, 17, 30, 9, 43].

When new telecommunication [features] are introduced, though the additional service design itself is not particularly difficult, designing the [feature] interactions between existing [features] and the additional [feature] is quite difficult. [26]

Researchers agree that the feature interaction problem is complex and may never be fully resolved, however there is a definite need to find better methods to manage this problem.

As we discussed in Section 3.5.4, using a set of prioritized features to resolve detected feature interactions is a common technique [15] and the importance of determining a priority ordering is reflected by the variety of feature interaction approaches that use prioritization of features as part of their approach. These approaches use prioritization to prevent interactions, to automatically resolve detected interactions, or as a back-up plan for alternate resolution techniques. The priority ordering determined through the approach presented in this thesis can be used as either a precedence or priority ordering. When the generated ordering is used as a precedence ordering,

it determines the order in which the features should be applied. However, when the ordering is used as a priority ordering, it determines which of the conflicting features should be chosen for execution at this instance but not how the features behave with respect to each other in other circumstances. Throughout the remainder of this thesis, we will refer to the generated ordering as a priority ordering, although it works equally well as a precedence ordering.

The determination of the priorities is itself a difficult and costly task. As shown in Section 3.6, most existing prioritization strategies are limited in their ability to fully order a set of features with respect to the full range of interactions that can occur in the system. The approaches discussed [15, 13, 48] are all limited in the types of interactions that are considered when prioritizing features. In [48], no interactions can be detected relating to unobservable actions, hence no interactions related to data recorded by the features are considered, while in [15] only interactions that can lead to a blocking or deadlock states are considered. Cattrall et al.'s work [13] only prioritizes roles and subroles (aka address zones) and does not prioritize features within a role or subrole.

Furthermore, the existing approaches are limited by restricting the size of the feature sets being tested or by the high cost involved in calculating the results for large feature sets. For example, the approach in [48] is limited to comparing features sets of at most size four, while the approach in [18] is limited to comparing pairs of features. The iterative algorithms in [15] are able to handle larger feature sets, however the amount of work required increases with respect to both the number of features and the number of blocking entities that exist when the features are combined. Although the cost of this algorithm may not be as high as the traditional cost of prioritizing features, it can become very highly in situations that have large numbers of both features and blocking entities. Thus, a more complete approach is needed to prioritize features with respect to different types of feature interactions and this approach should significantly reduce the cost of prioritizing the features. The approach presented in this thesis is designed to significantly reduce the cost to determine a priority ordering among features with respect to a wide variety of types of feature interactions.

Chapter 4

Prioritization Using Categories and Principles

Our work stems from the hypothesis that classifying features into categories and prioritizing these categories can decompose the feature prioritization problem. The prioritized categories provide worked-out resolutions to interactions between features in the different categories. This approach decomposes the prioritization task into the following steps: defining the categories, categorizing the features, prioritizing the categories, and determining each feature's intra-category priority with respect to other features in the same category. In order to determine the correct priority ordering between categories, it is necessary to find a set of principles of proper system behaviour that can be used to resolve interactions.

The creation of the categories and the categorization of each feature is a manual and time-consuming process that requires a linear amount of manual work with respect to the number of features, as each feature is either added to an existing category or spawns the creation of a new category. Guidelines for the creation of the categories are given in Section 4.1; these guidelines help reduce the effort required by the category designer. Ideally, the number of categories is far fewer than the number of features, and there is a logarithmic number of categories with respect to the number of features. The creation of principles for proper system behaviour, which are used to determine acceptable category orderings, is also nontrivial because the principles must not overly constrain the system and must hold regardless of which features are in the system. The creation of the principles also requires a linear amount of manual work with respect to the number of categories, since each of the categories is examined to determine if a principle(s) should be constructed to reflect any properties that must hold true for the category. The work required to prioritize the categories is factorial with respect to the number of categories; however, if the number of categories is small enough, it is possible to automatically generate the priorities using the set of identified categories and principles. Prioritizing the intra-category features remains a task that must be performed by the system designer using either another interaction detection

resolution technique or by manual analysis. The cost to compute intra-category priorities within each category is factorial with respect to the number of features in the category. However, the size of the problem is reduced from that of prioritizing all features.¹ The combined result is an adequate ordering of features that adheres to principles of proper system behaviour and that serves as input to priority-based resolution strategies, such as those described in Section 3.6.

The category-based approach to prioritizing features reduces the problem of prioritizing features as long as 1) the number of categories is significantly smaller than the number of features and 2) the features are roughly distributed among the categories. Preliminary analysis indicates that this relationship holds among telephony features: we were able to group over 350 features into 14 distinct categories. As previously mentioned, this work has been evaluated only on features in the telephony domain and, as a result, all categories and principles identified in this work relate to telephony services that are typically provided to residential clients.

The remainder of this chapter is organized as follows. Section 4.1 explores the different feature categories in the telephony domain. Section 4.2 identifies the different principles of proper system behaviour found within the telephony domain. Section 4.3 shows how the principles identify and resolve interactions that exist between features in different categories. In Section 4.4, we prove that our categorization approach holds with respect to features found within the categories. Section 4.5 shows the results of our manual analysis to generate the prioritized ordering of the categories with respect to the principles of proper system behaviour; this analysis is used to verify the automatically generated category orderings output by our Prolog model, which is described in the next chapter. Finally, in Section 4.6, we show how our categorization approach can be merged with Address Translation properties and prove that the results satisfy the combined set of principles.

4.1 Classification of Features

This section presents the results of categorizing features within the telephony domain. Features are categorized by their goals and essential functionality. A feature's **goals** are its user- or service-provider-defined objectives. A feature's **functionality** is its behaviour, which is performed while attempting to realize its goals. We do not claim that our proposed feature categories are complete or that they generalize to other domains. This work merely evaluates, within the telephony domain, whether telephone features can be effectively categorized.

We employ a simple set of guidelines when creating feature categories:

1. Each category should represent a single core functionality. Thus, a new category is created whenever a feature has a core functionality that does not correspond to any of the existing

¹In both cases, the cost changes from factorial to exponential (for a large collection of features), if all possible subsets of categories and features are tested rather than pairs of categories and features.

categories.

2. Once all features have been categorized using step one, the goal of the features within each category is examined. If multiple goals are found and if these goals may result in conflicting advice about how to order the categories, then the category should be further decomposed into multiple categories. This step may be considered after an initial analysis of the original categories determined in the first step.
3. The resulting category set is checked by the designer to ensure high cohesion among the features within a category and low cohesion between features in different categories. Usually, the application of the previous two steps will generate a category set that adheres to these conditions. However, this is in fact the most important step in the process, since these conditions will effect the performance of this approach.

This final step is important because, when there is low cohesion between the features within a category, then the creation of a representation feature for this category will be difficult. Another side effect is that a large number of feature interactions may be found between the non-cohesive category and other categories, if the interactions involve only a subset of concrete features in this category. In such a case, our approach will find interactions between categories that might not be realized by the majority of concrete features from those categories.

High cohesion between the features in different categories indicates that there are features in different categories that perform similar functions, therefore, it is probable that these features were designed knowing that they would interact with one another. As these features are probably designed to interact, our approach will likely identify a large number of interactions between the categories, with each interaction placing more restrictions on how the categories should be ordered. The final interaction set may determine that one or more of the categories cannot be “correctly” ordered with respect to another category. This type of interaction, between similar features, usually needs to be analyzed and resolved by a human expert. In contrast, the goal of our approach is to identify the unexpected interactions that occur between seemingly unrelated categories (i.e., to identify interactions that could be missed by a human expert), so that the expert is free to focus on ordering features that were designed to interact.

Below, we list the categories identified for the set of traditional subscription-based features in the telephony domain. We considered 352 features from 6 sources: the feature interaction benchmark [10], the second feature interaction contest [31], and from industry sources, such as Nortel and 3Com [1, 39]. Each category name is followed by a header, which identifies the region in which the features are active. At the end of each category description, we list some of the more commonly identified telephony features found in that category. These features and others can also be found in Appendix A, which contains the descriptions of a variety of features sorted by category.

Alias: (source and target)

An **Alias** feature allows the user to employ an alias, such that dialled codes are mapped to network addresses and vice versa. For example, a codename or a Speed Dial code can be dialled by the user and mapped to its corresponding network address, which is used to complete the call. **Alias** features are found in both the source and target regions of the call. In the source region, the subscriber dials a code and the feature retrieves the corresponding network address, possibly based on call-scenario details, such as time of day. In the target region, the inverse occurs: the feature performs a reverse mapping, translating the network address of an incoming call to its corresponding alias, the alias is then recorded and can be accessed by other features. Features in the **Alias** category include Area Number Calling, Parallel Dialling, Personal Directory, Sequential Dialling and Speed Calling.

Authentication: (source and target)

An **Authentication** feature verifies whether the user is authorized to initiate or receive a call passing through this address zone. For example, when a caller routes an outgoing call attempt through a remote address zone, a Remote Access Authentication feature can prompt the caller for a password before allowing the call to enter the remote address zone. A feature found in the **Source Authentication** category is Remote Access Authentication.

Billing: (source and target)

A **Billing** feature records the billing information for the subcall from one address zone to the next address zone – towards the callee. Alternately, **Billing** features, like Collect Call, may change which address is billed for each connection. Thus, a source address is billed for redirecting the call to another source address (e.g., from home to a work address), and a target address is billed for forwarding the call to another target address. The owner of the last address in a call's source region pays for the connection from the source region to the target region.

Some **Billing** features may change who is billed for a subcall. For example, a Collect Call (CC) feature reverses the charges incurred by a **Billing** feature. When CC is invoked, a message is played on the voice channel requesting authorization to reverse the charges for the cost of the subcall between the address zone containing CC and the next address zone (typically, this is the transition from a source address to a target address). The call will not be completed unless authorization is received from the callee. A Feature in the **Billing** category is Collect Call.

Features in other categories may also charge for their use. Such pay-per-use features are responsible for sending billing information to the billing database directly. Because billing is not their goal or functionality, these features are not considered **Billing** features.

Blocking: (source or target)

A **Blocking** feature prevents the completion of an outgoing (incoming) call whose target (source) address is found on the feature's blocking list. For example, Originating Call Screening aborts call attempts that originate from one of the subscriber's source addresses, and that are destined for a target address in the feature's blocking list. Another example is Incoming Toll Restrictions, which prevents the subscriber from receiving calls that result in extra charges, such as long-distance toll charges.

Features found in the **Blocking** category include Incoming Toll Restrictions, Terminating Call Screening, Originating Call Screening, Outgoing Toll Restrictions, and Teen Line. The first two features are active in the target, while the last three are active in the source region.

Delegate: (target only)

A **Delegate** feature redirects an incoming call to an agent acting on the subscriber's behalf (e.g., voice mail services, secretary). Some features, like Call Forward Universal, delegate immediately on receiving a *setup* signal; others, like Voice Mail, delegate on receiving a failure signal. The goal of a **Delegate** feature is to redirect an incoming call to a delegate, thus these features are active only in the target region. Features in the **Delegate** category include Call Forward, Call Transfer, Send Mail, and Voice Mail.

Disabling: (virtual category - source or target)

A **Disabling** feature stops another feature in this address zone from fulfilling its purpose. For example, Cancel Call Waiting (CCW) allows the subscriber to temporarily disable Call Waiting (CW) for the duration of a call. A **Disabling** feature is subscribed to as a separate feature from the feature it disables, and while such a feature offers a distinct service, its functionality is to "turn off" another feature. This functionality is most easily modelled by extending the disabled feature's module to respond to a *feature-specific-disabling* signal. Continuing our example, a user would be unlikely to subscribe to CCW without also subscribing to CW. By choosing to subscribe to both of these features, the user is assigned the extended CW feature model, which reacts to an incoming *cwDisable* signal by disabling its functionality for the duration of the call.

This extension of the feature module does not affect the modularity between the categories, because the ability to disable a feature has no effect on other categories. However, implementing the **Disabling** feature separately would not increase modularity, since the two features are so tightly coupled. We acknowledge the existence of **Disabling** features and recommend that these features be implemented as extensions of the feature module that they disable. Hence, **Disabling** features form a virtual category that is not prioritized.

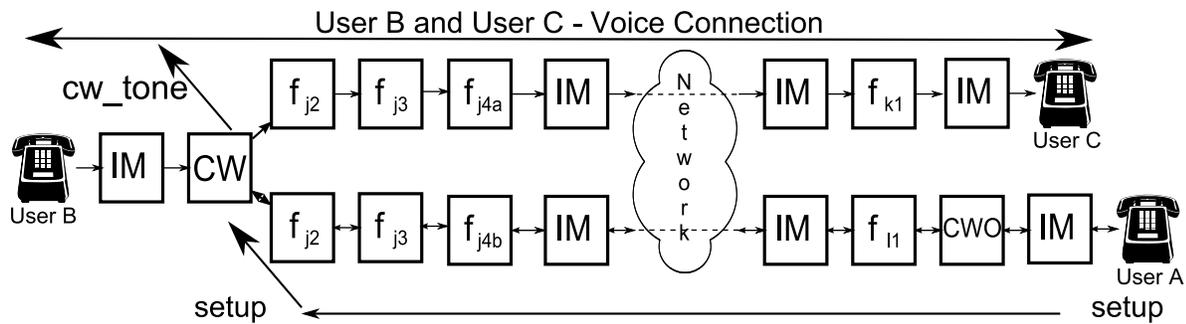


Figure 4.1. This figure shows the how a Multiplex feature behaves when a second incoming call is received.

Features in the **Disabling** category include Cancel Call Waiting, Call Waiting Exempt, and Originating Call Screening Override.

Filter: (target only)

A **Filter** feature selectively blocks or redirects calls not meant for the callee.² For example, a **Filter** feature may react to all incoming calls by offering the caller a menu of options on how to direct the call and then continuing, redirecting, or ending the call according to the caller's response. **Filter** features are strictly target-region features, since they react only to incoming call attempts. Features in the **Filter** category include Automated Role Identification.

Multiplex: (both source and target)

A **Multiplex** feature allows the subscriber to be involved in multiple call connections simultaneously. For example, Conference Calling allows multiple users to be involved in a single voice connection. **Multiplex** features are bound features, meaning that each subscriber has one static instance of each **Multiplex** feature, and all of the subscriber's calls pass through this feature instance. This means that although a **Multiplex** feature is found in both the source and target region, it is implemented as a single feature, and hence **Multiplex** is treated as a single category. This way, a **Multiplex** feature is aware of all of the subscriber's calls and call attempts and can coordinate among the calls as per their respective goals. Features in the **Multiplex** category include Call Waiting, Conference Calling, and Three-way Calling.

When a **Multiplex** feature is active in a call path (i.e., a connection exists between the Subscriber and the feature) and it receives the *setup* signal of another call, it will not propagate

²The **Blocking** category dispenses with undesired calls, whereas the **Filter** category handles misdirected or to-be-directed calls. As well, **Filter** features may reveal information to help the caller complete her call and we do not want this private information to be revealed to a callee who's call attempt will be blocked.

the signal; instead, it passes a *feature-specific* signal through the existing connection to the subscriber as shown in Figure 4.1. For example, Call Waiting sends a special notification signal along the voice channel, rather than propagating the *setup* signal along the traditional signal path. Consequently, features that lie on the call path between the subscriber and the Multiplex features do not get re-instantiated for the new call attempt and will not receive signals (e.g., DFC signals) issued during the initiations of a new call attempt.

Presentation: (source and target)

A **Presentation** feature presents information about the call to a user. For example, Call Display will display information about an incoming call, such as the caller's alias or phone number, on the end device. Features found in the **Presentation** category include Call Display, Distinctive Ringing, and Group Ringing.

Redial: (source or target)

A **Redial** feature is used to place a call to a previously recorded address. The most basic **Redial** feature places a call to the target (source) address of the last unsuccessful outgoing (incoming) call attempt. For example, Automatic Call Back (ACB) is used to reach an unavailable callee: ACB records the target address of the outgoing call. If the callee's number is busy, the caller can activate ACB and hang-up. Behind the scenes, ACB continuously checks the status of the callee's number. When the callee becomes available, ACB first establishes a connection to the original caller and on acknowledgement completes the call to the callee.

Redial features are also found in both the source and target region. We represent these features as separate categories, one for each region, because they record different information depending on whether the call attempt is incoming or outgoing. Features in the **Redial** category are Automatic Call Back (source) and Call Return (target).

Redirect: (source or target)

A **Redirect** feature is used to route a call attempt through another address zone in the same region, without changing the intended participant of the call. In the source region, a caller might do this to change the caller-identification information that the callee sees, to bill an outgoing call to a different source address, or to access the features associated with a different source address. For example, the Remote Access feature allows a caller at one location to dial a special access code to link to a remote address. The outgoing call will appear as though it originated from the remote address, the caller will have access to any features associated with the remote address, and the remote address will be charged for the call to the target address. In the target region, a callee might redirect the call attempt to an alternate device, where the callee is currently located. When this redirection occurs, no

other features, except billing which is linked to the Network, are added to the call path in the current address zone.

Note that many telephony features currently serve as both **Delegate** features and **Target Redirect** features. For example, Call Forward features can redirect a call either to the callee at an alternate address or to a delegate. These multiple-purpose features (see Section 4.1.1) need to be decomposed into single “goal”-based feature modules.

Remote-Control Invoking: (source or target)

A **Remote-Control Invoking** feature allows the subscriber to invoke an unsubscribed feature in the remote user’s address zone. For example, the caller can invoke the source feature Call Waiting Originator (CWO) to give the callee use of the feature Call Waiting (CW) for the duration of the call. Each Remote-Control Invoking feature is implemented as two feature modules: a **Remote-Control Invoking** module, which is used to invoke the feature, and a **Remote-Control Action** module, which provides the service associated with the Remote-Control Invoking feature. Features in the Remote-Control Invoking category include Busy Override, Call Waiting Originator, Call PickUp, and Malicious Call Hold.

As shown in Figure 4.2, the Remote-Control Invoking module triggers the activation of the remote user’s corresponding Remote-Control Action module by issuing a signal, either a unique feature-specific triggering signal or a modified *setup* signal (e.g., *setup(trigger)*) that triggers the feature’s functionality. When a modified *setup* signal is used and the call attempt reaches the appropriate position, the Remote-Control Action module is added to the call path. The Remote-Control Action module is positioned according to normal placement in the feature ordering: that is, the Remote-Control Action feature is placed as if it were subscribed to by the remote user.

Remote-Control Override: (source or target)

A **Remote-Control Override** feature allows the caller to disable the functionality of a feature subscribed to by a remote user. These features are distinct from **Disabling** features, which override other features within their own address zone. For example, the source feature Make Set Busy Override (MSBO) can override the reported unavailability of the remote user, *if* the unavailability is due to the callee’s Make Set Busy (MSB) feature. The MSBO feature reacts to the incoming *unavail* signal by providing the caller with the option to retry the call attempt using a modified *setup* signal (e.g., *setup(MSBDisable)*) that would disable the callee’s MSB feature, so that the remote user’s MSB feature would have no effect on the incoming call attempt’s ability to reach the remote user. This category can be thought of as a combination of Remote-Control Invoking and Disabling

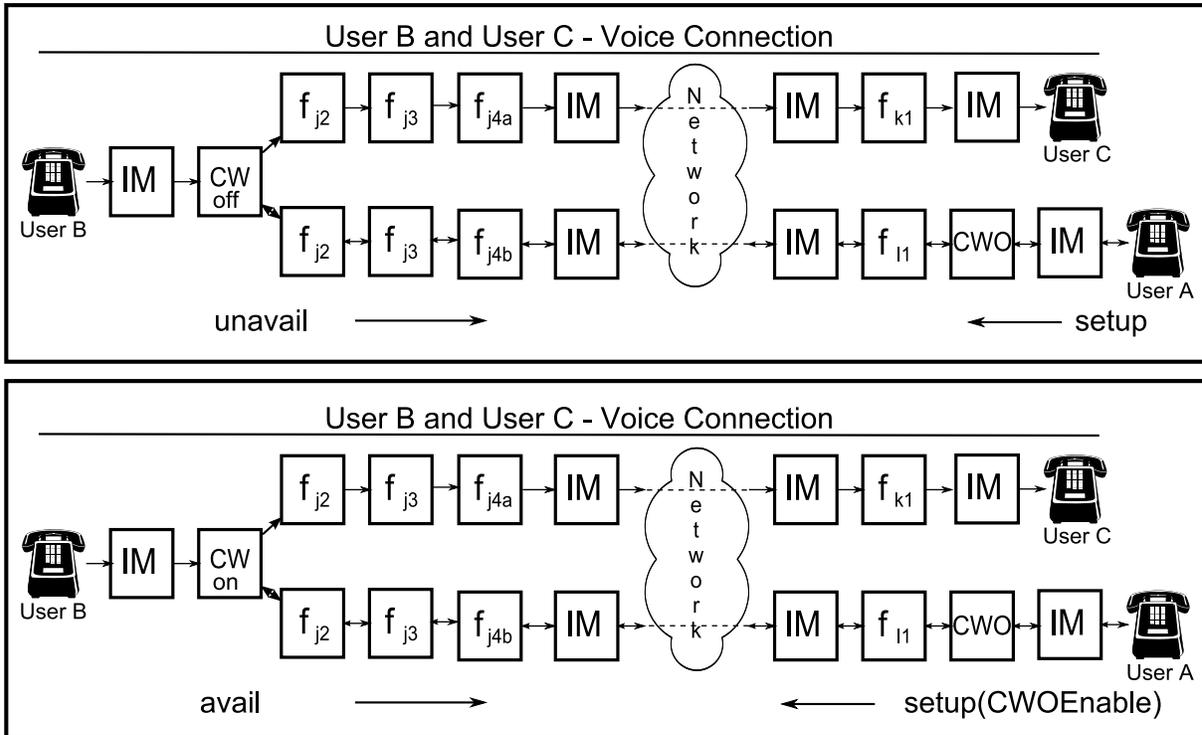


Figure 4.2. The top image shows an initial call attempt from user *A* to user *B*, since user *B* is already involved with a call to user *C*, user *A* receives an *unavail* signal. At this point user *A*'s Call Waiting Originator feature prompts user *A* and offers to retry the call attempt by giving Call Waiting (CW) to user *B*. User *A* selects this option and the new call attempt is shown in the bottom image. The new call attempt is initialized with *setup(CWEnable)*. Note that CW is found in user *B*'s address but is not subscribed to and is hence turned off. When this CW feature receives *setup(CWEnable)*, the feature becomes active and User *B* now has the option to respond to CW and answer the incoming call from user *A*.

features, where the remote command is a *disabling* signal and the action module is an extended feature module that responds to the *feature-specific-disabling* signal. Hence, each **Remote-Control Override** feature is composed of two modules: a **Remote-Control Override** module, which sends the *disable* signal and a **Remote Control Disable** module, which disables the subscriber's functionality. The **Remote-Control Disable** modules are special-case disabling features, where the trigger signal is sent by a remote user rather than by the subscriber. As with **Disabling** features, we recommend that the original feature be redesigned to respond to a *feature-specific-disabling* signal that turns off the feature's functionality. Features in the **Remote-Control Override** category include **Make Set Busy Override**, **Call Display Blocking**, and **Terminating Call Screening Override**.

Set Outcome: (target only)

A **Set Outcome** feature can assert the outcome of a call attempt by issuing a signal on behalf of the callee. For example, **Make Set Busy** intercepts all incoming *setup* signals and issues an *unavail* signal in response - regardless of the callee's actual status. **Set Outcome** features are active only in the target region, as they generate failure signals in response to incoming call attempts. Features found in the **Set Outcome** category include **Make Set Busy** and **Do Not Disturb**.

4.1.1 Multiple Purpose Features

The above categories describe the different types of goals for which a feature can be designed. However, it is possible that a feature may have several purposes and thereby fall into more than one of our feature categories. For example, the **Call Forward on Setup (CFS)** feature falls into both the **Delegate** and the **Target Redirect** categories. CFS is designed to redirect the incoming call attempt whenever a *setup* signal is received. However, the call attempt may be redirected either to the subscriber at a new location or to a delegate for this subscriber.

We recommend that **Multiple Purpose** features be decomposed into multiple feature modules, one for each of the feature's purposes. The modules' implementation may be the same, but the system's architecture or routing protocol would include them at different points in a call. These feature modules can be designed to work together and can communicate via feature-specific signals. In the above example, two CFS feature modules would be created, CFS-S for call attempts that are redirected to the subscriber, and CFS-D for call attempts that are redirected to a delegate.

Alternatively, a **multiple purpose** feature can be placed in the feature category of the more dominant purpose, as long as this placement does not violate any category-ordering restrictions imposed by the feature's other categories. For example, "pay-per-use" features have two purposes: to provide some service and to bill the subscriber for each use of this service. The feature module implements its billing functionality by sending bill information directly to the billing database, which does not violate any of the principles.

4.2 Principles of Ideal Feature and System Behaviour in Telephony

We have identified a set of **principles of proper system behaviour** that identify properties that should hold for the system and for the set of feature categories, regardless of which feature categories are present. When taken together, these principles help identify invalid feature orderings and ensure desirable interactions between feature categories.

We have identified two types of principles: **constraint principles** and **criterion principles**. Constraint principles are critical to proper system behaviour and must hold to ensure proper system functionality and often represent safety or liveness properties in the domain. Criterion principles reflect desirable system properties that should hold whenever possible and usually desirable domain requirements. These principles are designed to hold within a single address zone and to be feature independent (i.e., they hold regardless of which categories are subscribed to by that address zone).

Constraint Principles

Abortion: Calls made to (or from) numbers that appear on a subscriber-defined blocking list for outgoing (or incoming) calls should be aborted.

Authorization: When authentication is required, a user's identity must be verified before the user can interact with any of his features. A call's voice connection between the end-users is not fully formed until user authentication is complete.

Invoicing: For every address zone, A , connected through the Network to another address zone, B , the cost of the subcall from zone A to zone B is charged to some user. Normally, the owner of zone A is billed for the cost of the subcall to zone B , unless some feature on behalf of another address zone agrees to accept the charges.

Network: Whenever a call attempt reaches the Network's interface module, which routes the call attempt through the Network and to the next address zone in the call path, the network address of the next address zone is known. The Network does not know how to route a dialled code, such as "Mom", and hence a code must be translated into an address before the call attempt reaches the Network for routing.

Criterion principles

Accessibility: All of the features associated with any address zone in the call will be included in every established call. Each end user expects that her full set of features are accessible to her for the duration of the call.

Concretization: Any source feature that makes decisions about call progress should have access to both the call's target address and any available alias information. Consider a feature that blocks outgoing calls that incur long distance fees. If the feature receives a *setup* signal whose target address is a dialled code, then the feature cannot properly determine whether or not a long-distance fee will be incurred, since the network address is unknown.

Failure: Any feature that is triggered by the receipt of a *failure* signal (i.e., *unavail*, *ringTO*) is positioned such that all *failure* signals in this address zone of the call path will pass through this feature before exiting the address zone. For example, the user's Voice Mail feature should be triggered by every *unavail* signal that passes through its address zone, even if the *unavail* signal was generated by another feature, such as Make Set Busy.

Logging: Information about all successful and non-successful calls should be recorded, with the exception of incoming call attempts that are blocked or instantly redirected to a delegate. Blocked calls are treated as if they never occurred, and instantly delegated calls are treated as if they do not involve the subscriber.

Personalization: Aliases should be used, when they exist, to present user-related information to subscribers in the target region. For example, a subscriber would rather be notified of an incoming call from "Mom" than be presented with the caller's network address. Features that record network addresses and that can subsequently initiate calls for the subscriber should also record and use aliases, to take advantage of potential changes to alias assignments (e.g., redial "Mom", so that redialling selects the correct target address for "Mom", which may be dependent on the time of day).

Presentation: Only information about incoming calls that reach the subscriber's end device should be presented to the subscriber, and all calls that reach the subscriber's end device should be presented, when a presentation feature is subscribed to. The subscriber should not receive information about calls that have been blocked or instantly redirected to a delegate.

In total, four constraint principles and six criterion principles were identified. We feel confident that these principles identify the majority of the system requirements due to the fact that these principles correctly identify the majority of known feature interactions, as discussed in research papers presented in the Feature Interaction Workshop proceedings [6, 16, 17, 30, 9, 43], that occur within a single address zone. As mentioned above, these principles apply to features associated with a particular address. Thus, a subscriber whose calls pass through more than one source address might have multiple blocking lists, one for each source address; each of the addresses is billed for the subcall to the next address; and for each of the end users, all of the features associated with all of their addresses are included in the call.

4.3 Example Interactions

To illustrate how the given principles identify feature interactions between categories and indicate ideal feature orderings, we examine two well known interactions.

4.3.1 Three-Way Calling versus Personal Directory

The Multiplex feature Three-Way Calling (TWC) is used while in an existing call to initiate a new call to a third user and to include that third user in the voice connection of the original call. The Aliasing feature Personal Directory (PD) supports aliasing of dialled numbers. Due to the *Personalization* principle, Aliasing features should lie between Multiplex features and the Network. This placement allows a TWC subscriber to use a PD alias when dialling a third party: the PD feature will receive the dialled alias and translate it into a network address before the *setup* signal reaches the Network.

4.3.2 Call Return versus Make Set Busy

The Redial feature Call Return is used to return the last incoming unanswered call. The Set Outcome feature Make Set Busy issues an *unavail* signal to all incoming calls. Due to the *Personalization* principle, Redial features are on the Network side of Set Outcome features, so that Call Return is applied first and records the source address of the incoming call before Make Set Busy intercepts the *setup* signal and rejects the call. Hence, the callee is able to return the call.

4.4 Correctness of Prioritization Using Categories

This section considers the correctness of our categorization approach: Are the concrete features within the categories correctly ordered, with respect to the principles of proper system behaviour, when compared against concrete features in other categories. As previously mentioned, we do not consider feature interactions that occur between source and target features in the same call, since there are few principles that can indicate how these regions should behave with respect to one another. To prove this correctness criterion, we start by defining terminology and symbolic notation. Please note that some symbols are not defined in their traditional usage.

Definition 4.4.1. Generic Feature:

Each feature category, C_i , is represented by one or more generic features, f_{c_i} , that represent the functionality provided by that category. (Most categories will have a single generic feature.) This generic feature can be combined with specific feature data, d_i , to cover all possible execution cases that could be performed by a feature in the category.

Definition 4.4.2. Category Ordering:

A sequence, C^* , of feature categories, $C^* = [C_1, \dots, C_r]$, is applied such that each feature is added to the call path, in order, from left to right for outgoing (or from right to left for incoming) setup signals. The application of a feature category is represented by apply its generic feature, f_{c_i} .

Definition 4.4.3. Call Scenario:

A call scenario, cs , represents one of the many possible call paths that can represent a specific category ordering, C^* . For each feature category $C_i \in C^*$, the call scenario consists of a generic feature f_{c_i} that is associated with specific feature data, d_i (e.g., the caller's call-screening list includes $userT$, or the callee's Personal Directory feature maps the code $userC$ to the number $userT$). The symbol “ \circ ” denotes the composition of features in a DFC architecture.

$$cs = f_{c_1} \circ f_{c_2} \circ \dots \circ f_{c_u}, \text{ where } f_{c_i}(d_i) \in C_i$$

We use the symbol “ α ” to indicate that a call scenario, cs_i , is one of the possible call scenarios for a category ordering C^* . Thus, the following identifies all possible call scenarios for a given category ordering:

$$\forall k \cdot cs_k \alpha C^*$$

Definition 4.4.4. Call Simulation:

A call scenario, cs , is simulated by sequentially generating a sequence of call stages, $CS_1; CS_2; \dots; CS_u$, that represent the intermediate stages of the call.³ The first call stage, CS_1 , represents the initial state of the system, and each subsequent call stage, CS_i , represents the passing of signals and the execution of another feature category. A call simulation generates the call stages by executing the composition that defines the call scenario, where the type of signal received affects whether the composition is performed left to right (outgoing signal) or right to left (incoming signal). The symbol “ \sim ” denotes the simulation of a call scenario, and the symbol “ $;$ ” denotes the concatenation of the call stages.

$$\sim cs = CS_1; CS_2; \dots; CS_u$$

³The number of call stages corresponds to the number of features added to the call path, as the addition of each new feature generates a new call stage.

We use the symbol \in to denote a test as to whether a call stage, cs_i , is a stage in a call scenario simulation, $\sim cs$.

Definition 4.4.5. Principle Violation:

A principle, P , is violated in the simulation of a call scenario, $\sim cs$, if during the simulation, the call attempt reaches a call stage, CS_i , where the properties of the principle do not hold.

$$\exists i \cdot CS_i \in \sim cs \wedge CS_i \not\models P$$

A principle P_j in the set of all principles \mathbb{P} is either a constraint or criterion principle, but not both.

$$\begin{aligned} \forall P_j \in \mathbb{P} \cdot (P_j \in ConPrin \vee P_j \in CritPrin) \wedge \\ ConPrin \cap CritPrin = \phi \wedge \\ ConPrin \cup CritPrin = \mathbb{P} \end{aligned}$$

Definition 4.4.6. Violation-Free Call Scenario:

A call scenario, cs , is violation-free if the simulation of the call scenario does not violate any of the principles.

$$\forall i \cdot CS_i \in \sim cs \wedge \forall P_j \in \mathbb{P} \Rightarrow CS_i \models P_j$$

Definition 4.4.7. Violation-Free Ordering:

A feature category ordering, C^* , is violation free if there does not exist a call scenario with this ordering that violates any of the principles.

$$\forall i \cdot cs_i \alpha C^* \Rightarrow (\forall j, k \cdot CS_j \in \sim cs_i \wedge P_k \in \mathbb{P} \Rightarrow CS_j \models P_k)$$

Definition 4.4.8. Optimal Ordering:

A feature category ordering, C^* , is optimal, if (1) the ordering is violation free or (2) the ordering violates no constraint principles and contains the smallest criterion principle violation count, with respect to other permutations of the same set of feature categories. The criterion principle violation count for a category ordering C^* is determined by adding together the total number of different types of criterion principle violations, which occur on every possible call scenario for this ordering.

Let $cs_i \alpha C^*$: where cs_i is a call scenario that adheres to ordering C^*
 $perm(C^*)$: is a permutation of C^*

Define $count(C^*)$, the criterion principle violation count, to be calculated as follows:

$$count(C^*) = \sum_{\substack{\forall cs \alpha C^* \\ \forall P \in CritPrin}} (\exists k \cdot CS_k \in \sim cs \wedge CS_k \not\equiv P) ? 1 : 0$$

where $X ? 1 : 0$ returns 1 if X is true and otherwise returns 0

C^* is an optimal ordering if

- 1) C^* is violation free, or
- 2) $\forall C^o \in perm(C^*) \cdot count(C^o) \geq count(C^*)$

Definition 4.4.9. Minimal Violation Principle Set:

If a feature category ordering, C^* , is an optimized ordering and is not violation free, then some of the call scenarios, $cs_i \alpha C^*$, will violate one or more criterion principles. The set of all principles that are violated for C^* is called $MinVio(C^*)$.

The above terminology and symbolic notation are used in the correctness property below and in the correctness property found in Section 4.6.

Correctness Property 4.4.1. Correctness of Category Prioritization

Given the optimal ordering, C^* , for a set of categories, any set of features taken from these categories is also optimally resolved with respect to the principles used to derive C^* .

Validation:

Assumption: This property uses the terminology defined above and is validated with respect to a DFC-based architecture in which features are serially composed, as defined in Chapter 2.

Contradiction is used to validate this property.

Let $C^* = [C_1, C_2, \dots, C_u]$ be an optimized ordering of the categories

Assume $\exists f^* = [f_1, f_2, \dots, f_u]$ a sequence of features such that:

- 1) each concrete feature, f_i , is found in category, C_i
 $\forall i \cdot f_i \in C_i$
- 2) the simulation of a call scenario, cs , with ordering f^* violates principle P
 $cs \alpha f^* \wedge \exists j \cdot CS_j \in \sim cs \wedge CS_j \not\equiv P$

From the definition of a call scenario, using features instead of feature categories, we know that:

cs is associated with the feature data, d_i for each feature, $f_i \in f^*$

Therefore, we get:

$$\forall i \cdot f_i(d_i) \in C_i \wedge C_i \in C^* \wedge \exists j \cdot CS_j \in \sim cs \wedge CS_j \not\equiv P$$

Next we construct a corresponding call scenario, cs' , using the generic features for the categories in the category ordering C^* . We select the generic feature f_{c_i} that corresponds to the category C_i and select the feature data, d'_i that represents the same behaviour in f_{c_i} , as d_i represented in f_i , with respect to observable input and output results. When we consider the simulation of this new call scenario

$$\sim cs' = CS'_1; CS'_2; \dots; CS'_t$$

we see that it should generate CS'_k which violates P .

Given that $[C_1, C_2, \dots, C_u]$ is an optimized ordering, we know that it is either a) violation free or b) violates the minimum number of principles.

- a) if C^* is violation free, then by definition there does not exist a call scenario that violates any principle. Hence, cs' cannot violate P and we get a contradiction.
- b) if C^* violates the smallest number of criterion principles for any ordering of these categories, then the principle P either is (case i) or is not (case ii) a member of the minimal violation principle set for C^* .
 - i) $P \in MinVio(C^*)$: The violation of principle P is deemed an acceptable solution for resolving interactions in an overly constrained system. That is, the violation of this principle is known and accepted as correct behaviour in this system and hence this ordering is optimal.
 - ii) $P \notin MinVio(C^*)$: This generates a contradiction, because C^* is an optimized ordering of the categories. Thus, there does not exist a call scenario that violates principle P in C^* . \square

4.5 Manual Analysis Results

In Figure 4.3, we show the results of our priority ordering of feature categories. In this section, we begin with a brief overview of observations that we made during our analysis and then briefly explain how each principle affects the placement of the categories in the partial order. The partial ordering generated during our manual analysis is used as one of the criteria for evaluating the output of our Prolog model, which is described in Chapter 5.

The partial ordering shown in Figure 4.3 applies to both source and target address zones. Categories that combine both source and target features are represented by rectangular boxes, while source-only categories and target-only categories are represented by circles and pentagons, respectively.

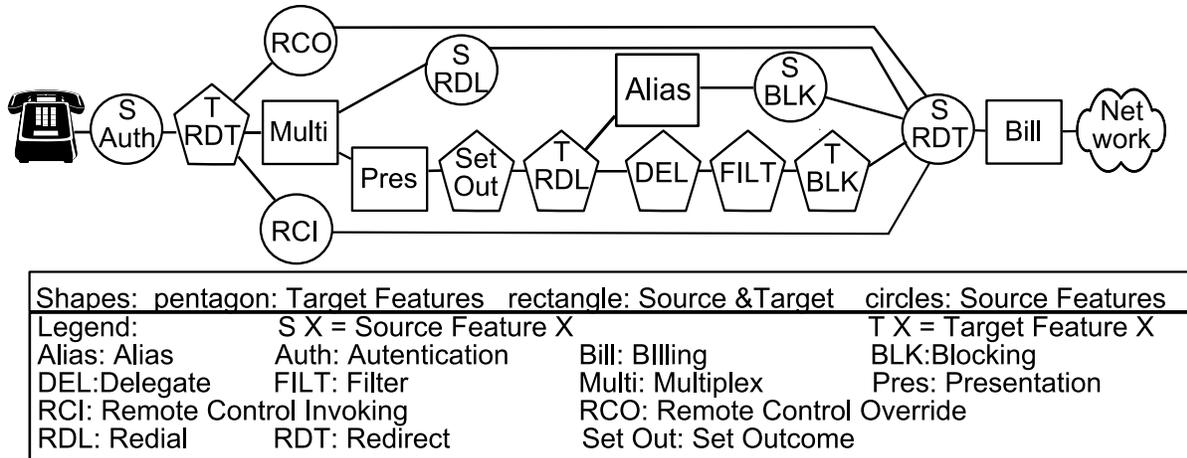


Figure 4.3. Partial Ordering of Feature Categories.

4.5.1 Observations

Initially, when designing the feature categories, we did not distinguish between source and target region features and we placed each feature in the category that matched the functionality or goal of the feature. However, during our manual analysis of the categories, we discovered that some categories behaved differently when simulated in the source or target region. As a result of these differences, no acceptable ordering could be found for the categories, since the placement of a category in the source region conflicted with the placement of the category in the target region. Thus, we decomposed some of the categories into two categories representing the source and target region features separately. We also observed, when this separation was necessary, that the goals of the features in either region had subtle differences. For example, the **Redirect** category had to be separated, because features in this category need to be among the last features to execute in their active respective regions, so that redirected calls include as many of the other features categories as possible. The subtle difference in their goals is that the source-region features are typically designed to add a layer of abstraction (e.g., route this call from home through my work address, so that the callee only sees work-related details) to the call attempt, while the target-region features are focused on locating the user at her current location, usually by removing layers of abstraction.

Other feature categories that were decomposed based on this observation were **Redial**, **Blocking** and **Authentication**. **Redial** was decomposed because we felt that the purpose of these features were subtly different in that source-region features allow the user to reattempt a failed call attempt made by the subscriber, while the target-region features allow the user to return an unsuccessful call attempt made by a third-party that failed to reach the user. The final results of our analysis show that this separation was not necessary, in that the placement of a single **Redial** category would have been possible.

Similarly, we represent **Blocking** features using two categories, one for each region, because their goals are slightly different: a source-**Blocking** feature prevents the callee from making certain calls (i.e., calls that incur a long distance charge), while a target-**Blocking** feature prevents certain callees from reaching the subscriber. Again, as with **Redial**, this separation was found not to be necessary in the final analysis.

Finally, the **Authentication** category is separated into two categories, because the placement of the **Authentication** feature should be such that user authentication is applied before other categories can be accessed by the user. Thus, the source and target-region categories will be on opposite ends of the category ordering. However, we subsequently deduced that the target-region **Authentication** features should be implemented as part of the feature that causes the call attempt to redirect and look for the subscriber at an alternate location, since no other features, except **Billing** are added to the call path after a redirection occurs. Therefore, the **Target Redirect** feature is the closest feature to the subscriber in its address zone and from this location, the **Target Redirect** feature is able to act as a gate to the remainder of the features in this address zone with respect to signals issued by the callee. If the redirected call requires that the callee be authenticated, then this is an acceptable position in which to validate the user's identity once the call is answered, since this location ensures authentication of the callee before the callee is allowed access to any of the address zone's other features. Thus, **Target Authentication** is implemented as part of the **Target Redirect** category. Features found in the **Redirect** category include **Remote Access** (source) and **Call Forwarding** (target).

Another observation we made is that due to the possible presence of a **Multiplex** feature in the call path, a user may be involved in multiple calls at the same time, and the user could be considered both the caller and the callee at the same time. Therefore, to ensure that all of a user's features are present in a call, even when the caller/callee status of the user changes after the initial call path has been established, we incorporate all of a user's features into every call⁴.

4.5.2 Principles and Category Ordering Results

The following list briefly explains how the principles in Section 4.2 effect the ordering of the categories from Section 4.1. The < symbol is used to indicate that the category on the left-hand side should be placed on the subscriber's side of the category found on the right-hand side. Note that when considering the **Multiplex** category, it is necessary to consider whether or not the other categories are affected if the **Multiplex** feature is already **instantiated** (i.e., in another call also involving the subscriber), since features on the subscriber side of **Multiplex** would not receive all generated signals issued during this call attempt.

⁴Technically, we need to incorporate only those features that are located between the user and the **Multiplex** category that is in the address zone closest to the network for each region.

Abortion principle indicates the following orderings.

- **Alias < Source Blocking**: This ordering ensures that any call attempt placed to a dialled code has been translated into the corresponding network address before the call attempt reaches the **Source Blocking** features. This ensures that the call can be properly continued or blocked based on either the dialled code or network address.
- **(Multiplex < Source Blocking), (Source Redial < Source Blocking)**: These orderings ensure that any feature that initiates a call attempt is positioned so that the call attempt is always tested by **Blocking** features before the call attempt reaches the Network.
- **Source Blocking < Source Redirect**: This ordering ensures that any outgoing call attempt is appropriately blocked before a redirection to another source address can occur.
- **(Delegate < Target Blocking), (Filter < Target Blocking), (Target Redirect < Target Blocking)**: These orderings ensure that any target category that redirects an incoming call attempt is able to do so only if the call attempt is not blocked.

Accessibility principle indicates the following orderings.

- ***All Other* Categories < Source Redirect**: These orderings ensure that if an outgoing call is redirected, all of the features associated with the current source address will be included in the call path. Note that ***All Other* Categories** does not include **Billing**, which is linked to the Network, but does include target-region features (which are included in all call paths since **Multiplex** features allow the user to be both a caller and a callee simultaneously).
- **Target Redirect < *All Other* Categories**: These orderings ensure that if an incoming call is redirected, all of the features associated with the current target address will be included in the call path. Note that ***All Other* Categories** does not include **Billing**, which is linked to the Network and hence is part of that address zone, but includes all other categories.

Authorization principle indicates the following orderings.

- **Source Authentication < *All Other Source***: This ordering ensures that none of the remaining source features are applied before the caller is authenticated.

Concretization principle indicates the following orderings.

- **Alias < Source Blocking**: This ordering ensures that all dialled codes for an outgoing call attempt are translated before reaching **Source Blocking**, so that the call can be aborted appropriately. This ordering also adheres to the *Abortion* principle.
- **Source Redial < Alias**: This ordering ensures that all dialled codes for an outgoing call attempt are recorded by the **Source Redial**, so that the call can be redialled using the same dialled code.

Failure principle indicates the following orderings.

- **Set Outcome < Delegate**: This ordering ensures that the **Delegate** features are able to receive and respond to failure signals issued by the **Set Outcome** features.

Invoicing principle indicates the following orderings.

- **Billing < IM and is linked to the Network address zone**: This ordering ensures that billing is always applied regardless of whether or not the call is redirected or proceeds normally. One way to represent this is to place the **Billing** category as a category inside of the Network's address zone.

Logging principle indicates the following orderings.

- **Target Redial < Target Blocking**: If an incoming call is blocked, then no information about the call attempt should be recorded.
- **Target Redial < Delegate**: If a call is instantly delegated, then no information about the call attempt should be recorded.
- **Target Redial < Filter**: If a call is blocked or instantly delegated by the filter feature, then no information about the call attempt should be recorded.
- **Set Outcome < Target Redial**: This ordering ensures that **Redial** can record information about the incoming call attempt, even if the call attempt is eventually unsuccessful.
- **(Multiplex < Target Redial), (Multiplex < Source Redial)**: These orderings ensure that **Redial** features are able to record call information regardless of whether or not a **Multiplex** feature is already instantiated. If a **Multiplex** feature is instantiated and the above orderings are reversed, then new signals, such as the *setup* signal, will not be propagated pasted the **Multiplex** feature. Thus, features on the subscriber side of **Multiplex** will not be able to respond to or be triggered by these signals.

Network principle indicates the following orderings.

- **Multiplex < Alias:** This ordering ensures that a dialled code can still be used to initiate a call attempt, regardless of whether or not a **Multiplex** feature is already instantiated. If a **Multiplex** feature is instantiated and the above orderings are reversed, then a new call initiated by a **Multiplex** feature with an aliases target (provided by the user) would (1) bypass the **Alias** feature as the signal enters the **Multiplex** feature via the voice channel or (2) pass transparently through **Alias**, as the signal would be a *feature-specific setup* signal and the call could not be completed as dialled, since the dialled code was not translated into a network address.

Personalization principle indicates the following orderings.

- **Presentation < Alias:** This ordering ensures that information presented to the subscriber includes any available aliasing information.
- **Target Redial < Alias:** This ordering ensures that if the last recorded number corresponds to an alias, then the redialled call can also make use of the alias.

Presentation principle indicates the following orderings.

- **Multiplex < Presentation:** This ordering ensures that an incoming call attempt will be presented to the subscriber regardless of whether or not a **Multiplex** feature is already instantiated. If a **Multiplex** feature is instantiated and the above orderings are reversed, then the incoming call attempt would not be propagated to the **Presentation** feature, and thus the call information would not be presented to the subscriber, even though the call attempt itself could be presented to the subscriber by the **Multiplex** feature.
- **Presentation < Set Outcome:** This ordering ensures that no information about a call terminated by a **Set Outcome** feature is presented to the subscriber.
- **Presentation < Target Block:** If an incoming call attempt is blocked, then no information about the call attempt is presented to the subscriber.
- **Presentation < Delegate:** If a call is instantly delegated, then no information about the call attempt is presented to the subscriber.
- **Presentation < Filter:** If a call is blocked or instantly delegated by the filter feature, then no information about the call attempt is presented to the subscriber.

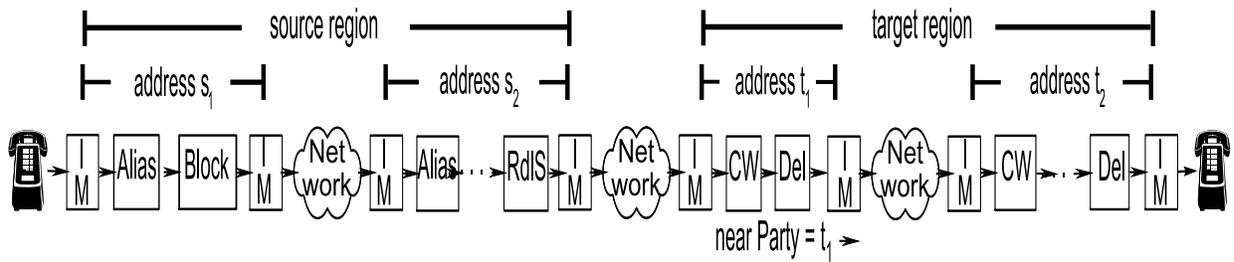


Figure 4.4. This figure shows a call established with two source address zones, s_1 and s_2 , and two target address zones, t_1 and t_2 . The call incorporates features from each address zone into the call path.

4.6 Combining Categorization with Address Translation

As discussed in Section 2.2, a call can be composed of several different address zones. These address zones are used to incorporate features associated with each of the addresses into the call as the call path is built. We use Ideal Address Translation (IAT) [52], which is the accepted approach for dealing with address translation in a DFC-base telephony domain, to order the address zones such that they adhere to the principles of IAT. Our categorization approach orders features within a single address zone with respect to the principles of proper system behaviour. To combine address zones with our feature categories, we nest feature categories inside each address zone, so that when a call attempt progresses to a new address zone, the features subscribed to by that address zone are added into the call path in the order determined by our feature category prioritization. This nested ordering is shown in Figure 4.4.

The nesting feature categories within several different address zones can potentially introduce undesirable interactions. This may occur when feature categories, which are prioritized with respect to a single address zone, are composed with other feature categories found in neighbouring address zones, which are prioritized according to Ideal Address Translation (IAT) principles. The nested ordering will almost certainly not adhere to the prioritized ordering computed for the feature categories. Hence, the question arises: Does the nested ordering result in undesired feature interactions?

In this section, we prove that the combined result is, in fact, an acceptable ordering that correctly resolves feature interactions based on the combined set of Ideal Address Translation (IAT) principles and our proper-system behaviour principles.

As shown in Figure 4.4, it is possible for multiple address zones in the same region to subscribe to the same feature (e.g., s_1 and s_2 both subscribe to *Alias*, and t_1 and t_1 both subscribe to the Multiplex-feature Call Waiting (CW) and to Delegate (Del)). In some cases, the feature copies

work well together, and in other cases, the features are likely to conflict with one another. As an example of a case where two or more copies of the same feature work well together, suppose each address zone ($t1$ and $t2$) subscribes to Call Waiting (CW). Each of these feature instances is present in the same call and acts as designed, so that the user can be involved in three separate call connections, instead of the typical two call connections. For example, user $s1$ places a call through address $s2$ as shown in Figure 4.4 and ends in a voice connection with user $t2$. At this point, another user $t3$ calls address $s2$ and the CW feature allows the user $s1$ to accept this call by putting user $t1$ on hold. Next, user $t4$ calls address $s1$, which also subscribes to CW, and once again, user $s1$ is presented with a new incoming call and is able to accept this call by putting $t3$ on hold. Thus, user $s1$ is involved in three distinct voice connections involving $t2$, $t3$, and $t4$ two of which are on hold. Problems may arise when determining which of the two CW features should react to a *cwSwitch* signal from the subscriber, but this could be solved by giving each CW feature a unique *cwSwitch* signal. Unique signals would also help the subscriber to determine from which address category an incoming call-waiting tone has been received (e.g., *cwHome* vs. *cwWork*).

As an example of a case where two copies of the same feature do not work well together, suppose that both address zones $s1$ and $s2$ subscribe to an *Alias* feature. Normally, the priorities of the address zones would automatically resolve interactions between features in different address zones, in that the feature in the abstract address zone has priority for incoming address zones and the feature in the more concrete address zone has priority for outgoing signals. However, in this example, when the subscriber dials a code, the *Alias* feature inside $s1$ is applied first and attempts to translate the code into a corresponding network address. If both address zones $s1$ and $s2$ have a mapping from the dialled code to a network address and these mappings are different, then when the signal continues into $s2$'s *Alias* feature, the second feature either overrides the mapping set by $s1$'s *Alias* feature or does nothing since the dialled code has been translated. (The call would not work if both *Alias* features translated the code into addresses because a call cannot have two different network addresses.) As mentioned, the priorities of the address zones automatically resolve this issue: because $s1$ is more concrete than $s2$, it is a reasonable assumption that the subscriber is dialling a code found in $s1$'s *Alias* feature and that this is the network address that should be used. Thus, when $s2$'s *Alias* feature sees that the network address is already supplied, it does not override this value.

However, there are cases where the default priorities imposed by IAT are undesirable. Suppose, for example, that address $t1$ is a role-based address for a group of sales representatives and address zone $t2$ is a personal address for a specific sales representative. If a call placed to the sales group is forwarded to the employee associated with address $t2$ and the employee is unavailable, then it would be preferable if address category $t1$ could re-delegate the call to a different employee, rather than defer to $t2$'s *Delegate* feature which takes a message. To address

this issue, IAT introduces the concept of near-party signalling. **Near-party signalling** is used to indicate to features in a more concrete address category that features in an abstract address category exert the right to respond to a failed call attempt. Consider Figure 4.4, in which a call attempt passes through $t1$, which generates a *near party = t1* signal. When the call attempt passes through address $t2$ and $t2$'s end device issues an *unavail* signal, $t2$'s **Delegate** feature knows not to respond to failed call attempts, and instead the **Delegate** feature continues the *unavail* signal unchanged. This protocol allows $t1$'s **Delegate** feature to receive and respond to an *unavail* signal and by redirecting the call to another employee. This example also holds true for any pair of conflicting features found in different target address zones that respond to failure signals (e.g., if $t2$ and $t1$ subscribe to any combination of **Delegate** or **Redirect** features). Using near-party signalling, an abstract address zone has the ability to exert priority over more concrete address zones.

Another complication resulting from the nested ordering of feature categories within address zones is that the overall feature orderings may be such that the features do not adhere to the ordering determined by our categorization approach. For example, in Figure 4.4, address $s1$ subscribes to a **Source Blocking (S BLK)** feature, while address $s2$ subscribes to a **Source Redial (S RDL)** feature; however our categorization approach tells us that this is a bad feature ordering because it violates the *Abortion* principle. Finally, this problem is implicitly addressed by the prioritization of the address zones. If $s1$ receives an outgoing signal that initiates a call attempt to a number blocked by the **Source Blocking** feature, then the call attempt will not reach $s2$'s address zone, since the **Source Blocking** feature will terminate the call before $s2$'s address can be added to the call path. Alternatively, if $s1$ receives an outgoing signal that is designed to initiate a call attempt by triggering the activation of $s2$'s **Source Redial** feature, then this signal will pass unchanged through address zone $s1$ and into address zone $s2$, where $s2$'s **S RDL** feature can generate a call attempt to a number found on $s1$'s call-blocking list. However, despite the fact that this violates our category ordering and the IAT principles - that is the concrete address $s1$ should have priority over a more abstract address $s2$ for outgoing signals - this is permissible. When a signal generated in $s1$ triggers the functionality of a feature in $s2$, the owner of $s1$ is acting in the role of $s2$ and $s2$ is allowed to place such a call. In the role of $s2$, it may be necessary for the owner of $s1$ to contact a person with whom the subscriber does not normally wish to speak (i.e., $s2$ is a work role and the owner of $s1$ is performing a work-related task from home).

These potential problems caused by nesting the categories inside the address zones are addressed in Correctness Property 4.6.1 below. This property states that the combined address zones and prioritized categories work together to correctly resolve feature interactions that violate the principles of proper system behaviour.

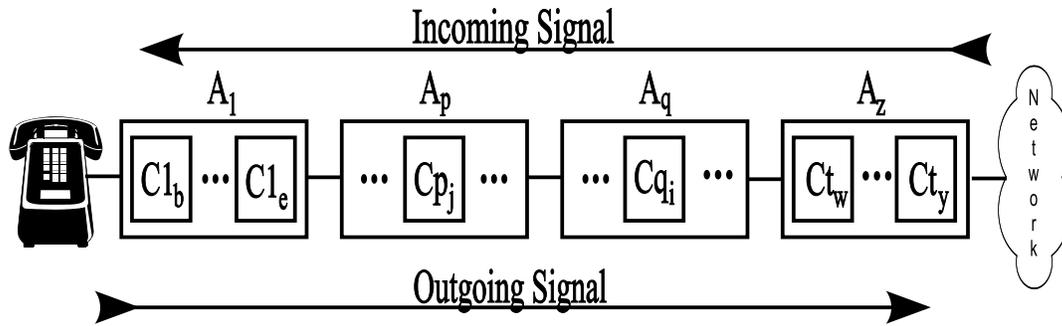


Figure 4.5. This figure shows the order in which the address zones and their categories are applied within a region, with respect to an incoming or outgoing signal.

Correctness Property 4.6.1. Correctness of Combined Address Zones and Category Prioritizations

Assumption: This property uses the terminology defined above and is validated with respect to a DFC-based architecture in which features are serially composed, as defined in Chapter 2.

Given a prioritized set of feature categories, C^* , and a set of address zones ordered according to Ideal Address Translation (IAT) principles, the resulting composition of all feature categories within all address zones in the call path correctly resolves interactions with respect to the categorization and IAT principles.

Problem Setup:

In Figure 4.5, we see how the address zones are composed and that the order of incoming and outgoing signals is such that an incoming signal passes through the address zones from right to left ($A_1 \leftarrow A_p \leftarrow A_q \leftarrow A_z$), while an outgoing signal passes through the address zones from left to right ($A_1 \rightarrow A_p \rightarrow A_q \rightarrow A_z$). Consequently, the order in which address categories are added to the call path is reversed in the target region. For example, when a user sends an outgoing *setup* signal, the signal prompts the addition of the address zones starting with A_1 and progressing towards A_z , which forms the source region for this call. When the *setup* signal reaches the target region for the call, the signal prompts the addition of the address zones in reverse starting at A_z and progressing towards A_1 . For simplify, we choose not to model the possible existence of network modules that can appear between the different address zones in the call, since the network model do not affect the signals that are sent or received along the call path and their presence would only clutter the figures.

For convenience, we restate some of the terminology related to IAT that was introduced in Sections 2.2 and at the beginning of this section. In Figure 4.5, we show the behaviour of signals in either region. If we consider the subcall shown, then address zone A_1 , which would be closest to the subscriber, is the most **concrete** address zone, while address zone A_z is the most **abstract**

address zone. Throughout the cases below, we use the fact that A_p is more concrete than A_q .

As mentioned above, IAT is used to order the address zones so that they adhere to IAT principles. However, there are situations where the default ordering of the address zones will not satisfy user requirements, thus the concept of near-party signalling was introduced to address this issue [52]. **Near-party signals** are issued by features in the abstract address zone and propagated to features in the more concrete address zone to indicate that the features in an abstract address zone exert priority in responding to failed call attempts. When a feature in the concrete address zone receives a *near-party* signal, the feature modifies its behaviour, so that it behaves transparently upon receipt of any *failure* signal (i.e., the failure signal is simply passed through the feature unchanged and the feature takes no action).

We use the following terminology in the validation below.

Let $C^* = [C_1, \dots, C_m, \dots, C_n, \dots, C_u]$
be the optimized ordering of *all* feature categories, although
not all address zones will subscribe to features in all categories.

$A^* = [A_1, \dots, A_p, \dots, A_q, \dots, A_z]$
be the set of address zones ordered according to IAT principles

$C_i = [C_{i_c}, C_{i_g}, \dots, C_{i_w}]$
be the ordered set of categories in address zone A_i ,
where the categories in C_i are

- 1) a subset of all the categories in C^*
- 2) adhere to the ordering in C^* , such that
 C_{i_j} is a member of C_i implies that $C_j \in C^*$

$A^* \oplus C^* = [C_1, \dots, C_p, \dots, C_q, \dots, C_t]$
 $= [C_{1_b}, \dots, C_{1_e}, \dots, C_{p_g}, \dots, C_{p_n}, \dots, C_{p_x}, \dots,$
 $C_{q_h}, \dots, C_{q_m}, \dots, C_{q_v}, \dots, C_{t_y}, \dots, C_{t_z}]$

is the nested composition of feature categories C^* within the address zones in A^*

Suppose a category $C_{p_n} \in A_p$ and a category $C_{q_m} \in A_q$ violate the ordering imposed by C^* .
That is:

- $A_p < A_q \in A^*$: satisfies IAT principles
- $C_m < C_n \in C^*$: satisfies categorization principles
- $C_{p_n} < C_{q_m} \in A^* \oplus C^*$: violates categorization principles

Then we need to prove that the behaviour of the combined addresses and their categories correctly resolves interactions based on the combination of principles of proper system behaviour and the IAT principles.

Validation:

This validation is performed using case analysis.

We decompose the validation into ten cases, one for each principle of proper system behaviour with respect to categorization. Three of these cases are given below; the remaining can be found in Appendix B. We further decompose each of the ten principle-based cases by identifying the pairs of feature categories that can potentially result in a principle violation, which we know from our manual analysis of the categories in Section 4.5, and we examine both orderings of each pair.

Case 1: Logging Principle

The *Logging* principle prohibits the recording of information about an incoming call attempt that is either blocked or instantly delegated, and ensures the recording of information for all other call attempts. The *Logging* principle is violated when 1) a **Log** feature records information about a call that is blocked or delegated in a target region or 2) when the **Log** feature fails to record information. The subcases to consider are:

1. Blocking - Log
2. Delegate - Log
3. Multiplex - Log
4. Set Outcome - Log

where **Log** is any feature category (i.e., **Source Redial** and **Target Redial**) that records information about call attempts.

Case 1.1: Blocking - Log

Assume that the features are ordered as shown in Figure 4.6. The *Logging* principle can be violated only during the initialization of a call attempt, so we consider call scenarios with an incoming or outgoing *setup* signal.

Incoming *setup* signal (target region): The **Blocking** feature in zone A_q is entered first, and the call attempt is either blocked or continued. If the call attempt is blocked, then the call attempt is terminated and the **LOG** feature in zone A_p never executes, so there is no violation of the *Logging* principle.

If the incoming call attempt is not blocked, then the *setup* signal continues towards the **LOG** feature in A_p , which records information about the call attempt; thus, there is no violation of the *Logging* principle.

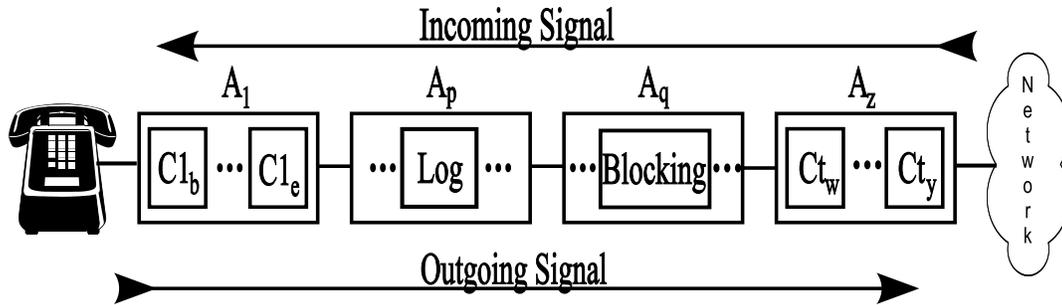


Figure 4.6. This figure shows that a feature category *Log*, which can be any feature category that performs a logging action, is in address zone A_p , while the *Blocking* category is in the address zone A_q .

Outgoing *setup* signal (source region): The *Log* feature in zone A_p is entered first and records information about the call attempt and the *setup* signal continues. Regardless of whether or not the call attempt is blocked by the *Blocking*, feature in A_q , no violation of the *Logging* principle occurs since the principle restricts the logging of information only in the target region.

Next, we consider the alternate feature ordering shown in Figure 4.7. Once again, we separate our analysis of the cases based on the direction (incoming or outgoing) of the *setup* signal and the region (target or source) of the interaction.

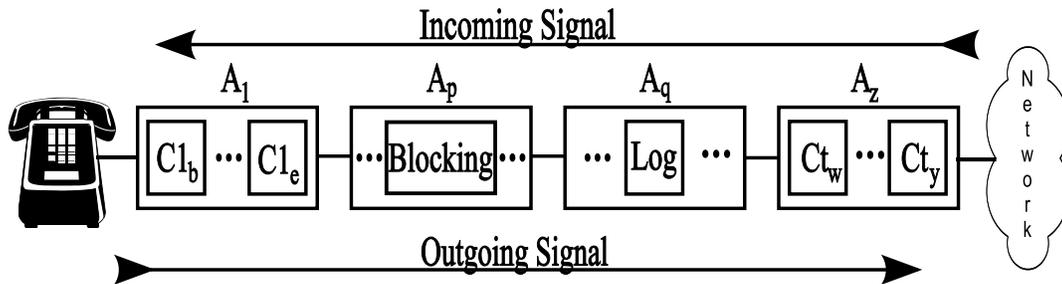


Figure 4.7. This figure shows that the *Blocking* category is in address zone A_p , while a feature category *Log*, which can be any feature category that performs a logging action, is in the address zone A_q .

Incoming *setup* signal (target region): The *Log* feature in zone A_q is entered first and records information about the call attempt and continues the *setup* signal. When the signal reaches the *Blocking* feature in zone A_p , the incoming call attempt will either be blocked or continued. If the call attempt is not blocked, then no violation of the *Logging* principle occurs.

However, if the call attempt is blocked, then a violation of the *Logging* principle has occurred, since the **Log** feature in A_q has recorded information about a call that is ultimately blocked in the target region. However, because zone A_q is more abstract than zone A_p in the target region, it has priority with respect to responding to incoming signals, according to IAT principles, and thus this is an acceptable violation. Consider a sales group, represented by address zone A_q , that receives an incoming *setup* signal from *Tom*. The call attempt is forwarded to a concrete sales-group representative in address zone A_p . Suppose that this particular sales representative blocks the incoming call attempt. In this case, we see that it is desirable for the sales group's features in address zone A_q to record information about the call attempt from *Tom*, since the group address zone assumes that the call is not blocked and is thus a call within their concern. It is the initial sales representative who does not want his features in A_p to record information about the call attempt that his feature blocked.

Outgoing *setup* signal (source region): The **Blocking** feature in zone A_p is entered first, and the call attempt is either blocked or continued. If the call attempt is continued, then the *setup* signal continues towards the **Log** feature in A_q , which records information about the call attempt, and no violation of the **Logging** principle occurs.

If instead the outgoing call attempt is blocked, then the call attempt does not enter the address zone A_q , so no *Logging* principle violation occurs.

Case 1.2: Delegate - Log

In this case, we explore the possibilities of an interaction between the **Delegate** and **Log** categories. The analysis of this case is similar to the above case involving the **Blocking** and **Log** categories. The above validation can be used to prove this case by simply replacing the **Blocking** feature with the **Delegate** feature and replacing “blocking a call attempt” with “instantly delegating a call attempt”.

Case 1.3: Multiplex - Log

In this case, we explore the possibilities of an interaction between the **Multiplex** and **Log** categories. The *Logging* principle can be violated only during the initialization of a call attempt, so we consider call scenarios with an incoming or outgoing *setup* signal. **Multiplex** features add another layer to our analysis as we must consider whether or not the subscriber's **Multiplex** feature is already active.

Non-active **Multiplex**

***setup* signal (either region):** When the **Multiplex** is not in use and receives a *setup* signal (in either region), it initializes the feature and continues the *setup* signal unchanged. Thus,

regardless of feature ordering, both the Multiplex feature and the Log feature will perform their duties, as the Log feature is not prevented from receiving the *setup* signal and recording the call information. Therefore, no violation of the Logging principle exists.

Active Multiplex

Assume that the features are ordered as in Figure 4.8 and consider the case in which Multiplex is already active.

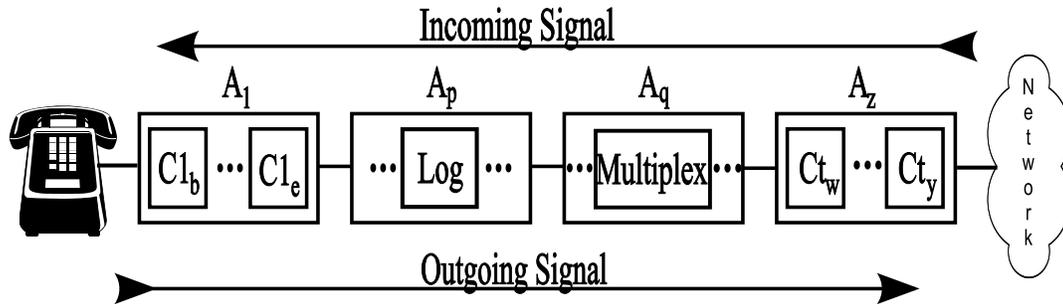


Figure 4.8. This figure shows that a feature category LOG, which can be any feature category that performs a logging action, is in address zone A_p , while the Multiplex category is in the address zone A_q .

Incoming *setup* signal (target region): Suppose that the Log feature is on the subscriber side of Multiplex in the target region, as shown in Figure 4.8, and the Multiplex feature is already in use, when the incoming *setup* is received. The Multiplex feature in zone A_q reacts to this call attempt by performing an action that allows the subscriber to interact with the new call attempt. However, because of the manner in which Multiplex features are designed, the *setup* signal is not propagated through the remaining features in the call path; instead the Multiplex feature notifies the subscriber of the new call using either the existing voice connection or a *feature-specific* signal along the call path. If the existing voice connection is used, instead of the typical signal routing through the features' ports, then the LOG feature does not receive any signal. Alternatively, if the LOG feature receives a *feature-specific* signal, it may not recognize the signal and may behave transparently. In either case, the LOG feature in zone A_p does not record any call information about this incoming call, even if the user chooses to answer the incoming call, and the *Logging* principle is violated. However, when the user accepts this incoming call, she is doing so in the role of the address zone A_q , interacting with the address A_q 's Multiplex feature. Hence, it is acceptable that the features in the more concrete address zone, A_p , do not record information about this call.

Next, we consider the alternate feature ordering shown in Figure 4.9. Once again, we separate our analysis of the cases based on the direction (incoming or outgoing) of the *setup* signal and the region (target or source) of the interaction.

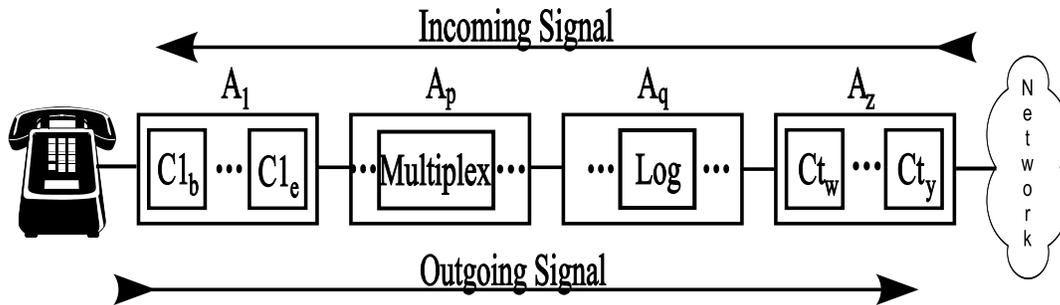


Figure 4.9. This figure shows that the Multiplex category is in address zone A_p , while a feature category Log, which can be any feature category that performs a logging action, is in the address zone A_q .

Incoming *setup* signal (target region): The LOG feature in zone A_q is entered first and records information about the call attempt and continues the *setup* signal. When the signal reaches the Multiplex feature in zone A_p , the incoming call attempt will be processed and presented to the user as designed. No violation of the *Logging* principle occurs, since the call information has been recorded.

Outgoing *setup* signal (source region): Suppose that a Multiplex feature that exists in an already established call is used to trigger another call attempt. The Multiplex feature in A_p generates a new *setup* signal, which is sent towards the LOG feature. When the LOG feature in zone A_q is entered, it records the call information and continues the *setup* signal, and no violation of the *Logging* principle occurs.

Case 1.4: Set Outcome - Log

In this case, we explore the possibilities of an interaction between the Set Outcome and Log categories. The Set Outcome feature is triggered only in the target region on receipt of an incoming *setup* signal, hence we need analyze only the cases shown in Figure 4.10 and Figure 4.11.

Incoming *setup* Signal (target region): In the target region, the *setup* signal first enters the Set Outcome feature, which either returns a *failure* signal or continues the call attempt. If the call attempt is continued, then the *setup* signal continues towards the LOG feature which records the call information and there is no violation of the *Logging* principle.

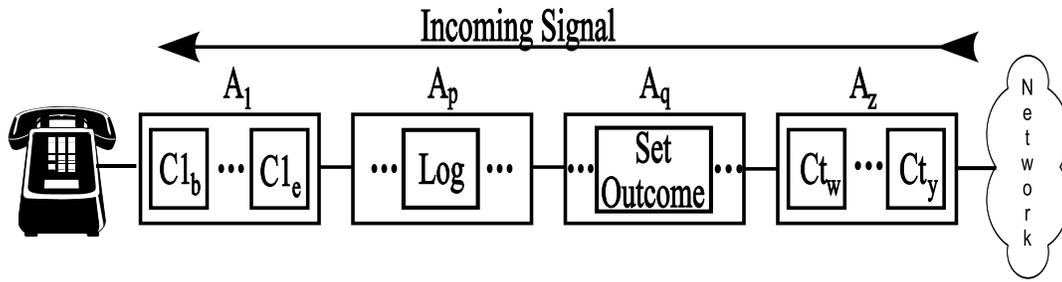


Figure 4.10. This figure shows that a feature category Log, which can be any feature category that performs a logging action, is in address zone A_p , while the Set Outcome category is in the address zone A_q .

However, if the Set Outcome outputs a *failure* signal in reverse, then the call attempt does not enter address zone A_p and the LOG feature in A_p does not record information about the call. However, because zone A_q is more abstract than zone A_p in the target region, it has priority with respect to responding to incoming signals, according to IAT principles, and thus this is an acceptable violation.

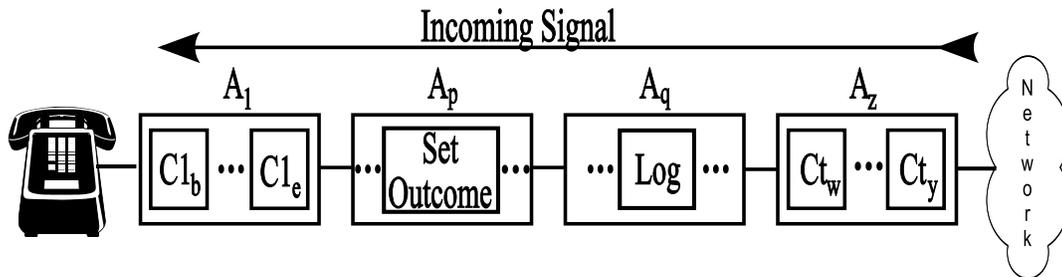


Figure 4.11. This figure shows that a feature category Log, which can be any feature category that performs a logging action, is in address zone A_q , while the Set Outcome category is in the address zone A_p .

Incoming setup Signal (target region): In the target region, the *setup* signal first enters the Log feature, which records the call information, before continuing the call attempt towards A_p . In A_p , the Set Outcome feature is entered and either returns a *failure* signal or continues the call attempt. In either case, there is no violation of the *Logging* principle, as the call information has been successfully recorded.

However, if the Set Outcome outputs a *failure* signal in reverse, then the call attempt does not enter address zone A_p . This violation of the *Logging* principle is acceptable, since the IAT principle gives A_q precedence over the more concrete A_p for incoming signals.

Case 2: The Abortion Principle

The *Abortion* principle states that no call to a blocked number will be established (i.e., no voice connection is formed). We know from our manual analysis of the categories in Section 4.5, that only the receipt of a *setup* signal can trigger a call attempt to be blocked. Consequently, we explore only category combinations that affect whether the *setup* signal reaches the **Blocking** feature or modifies the data associated with the *setup* signal (e.g., target address, alias codes). The subcases to consider are:

1. Alias - Blocking
2. GenCall - Blocking
3. RedCall - Blocking

where GenCall is any feature category that can generate a call attempt (i.e., Source Redial, Target Redial, and Multiplex) and RedCall is any feature category that can redirect a call attempt to an alternate location (i.e., Delegate, Filter, Target Redirect, Source Redirect and Target Redial, when triggered as its source counterpart).

Case 2.1: Alias - Blocking

Assume the features are ordered as shown in Figure 4.12.

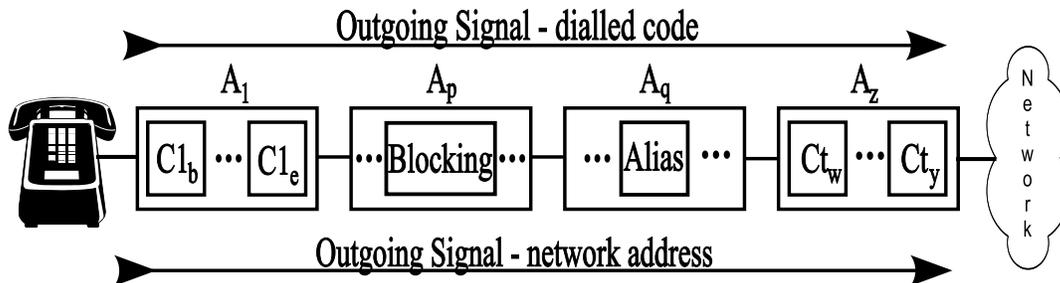


Figure 4.12. This figure shows that a feature category **Blocking** is in address zone A_p and the **Alias** category is in the address zone A_q .

Decoding an alias will affect only the **Source Blocking** features, since the **Target Blocking** features are designed to work solely on network addresses. Hence, we consider the receipt of an outgoing *setup* signal in the source region. Furthermore, since an **Alias** feature is present in the call path, the call attempt can be made to either 1) a dialled code or 2) a network address.

Outgoing *setup* signal to dialled code (source region): The **Blocking** feature in zone A_p is triggered by receipt of the *setup* signal and uses the dialled code to determine if the call should be blocked.

If the dialled code is found on the blocking list, then the **Blocking** feature terminates the call attempt and the *Abortion* principle is not violated. However, if the dialled code is not on the blocking list, then the **Blocking** feature continues the call attempt. When the call attempt is continued, the *setup* signal continues towards the **Alias** feature in zone A_q , which translates the dialled code into a network address.⁵ If the translated network address is not found on the blocking list of the **Blocking** feature in zone A_p , then the *Abortion* principle is not violated.

However, if the translated network address is found on the blocking list of the **Blocking** feature in zone A_p , then the *Abortion* principle is violated. This violation occurs because the call attempt is continued by the **Alias** feature and a voice connection is established with another user whose network address is found on a blocking list of a **Blocking** feature in the source region of the call path. According to the principles of IAT, this is the correct solution, since the caller chooses to act in the role of A_q when using A_q 's **Alias** feature to identify the network address for the callee. For example, a user may block personal calls to a specific person, but be required to speak with this same person as part of their job; hence it would be acceptable when calling from home through a work address to connect a call with this person.

Outgoing *setup* signal to network address (source region): The **Blocking** feature in zone A_p is triggered by receipt of the *setup* signal and uses the network (or any aliases identified by a previously executed **Alias** feature) to determine if the call should be blocked.

If the outgoing call attempt is blocked, then the *Abortion* principle is not violated. Alternatively, if the outgoing call attempt is not blocked, then the **Blocking** feature continues the call attempt. When the call attempt is continued, the *setup* signal continues towards the **Alias** features in zone A_q . Regardless of whether or not the **Alias** feature has a mapping from the network address to a code, this information will not affect whether or not the call attempt should have been blocked, since the codes for zone A_q will not necessarily correspond to the codes for zone A_p . For example, a feature in zone A_p might map the network address 1234 to “Pauline”, while a feature in zone A_q might map a different network address 2345 to “Pauline”, thus comparing the aliases determined in A_q with those on the blocking list for a feature in zone A_p is not desirable. Thus no violation of the *Abortion* principle occurs.

Next, we consider the alternate feature ordering shown in Figure 4.13. Once again, we separate our analysis of the cases based on the whether or not the **Blocking** feature can be effected by the

⁵If the dialled code is not translated, then a violation of the *Network* principle occurs and the call attempt cannot be completed.

presence of an Alias feature. Hence, we explore the subcases where an outgoing *setup* signal is used to initiate a call attempt to either 1) a dialled code or 2) a network address.

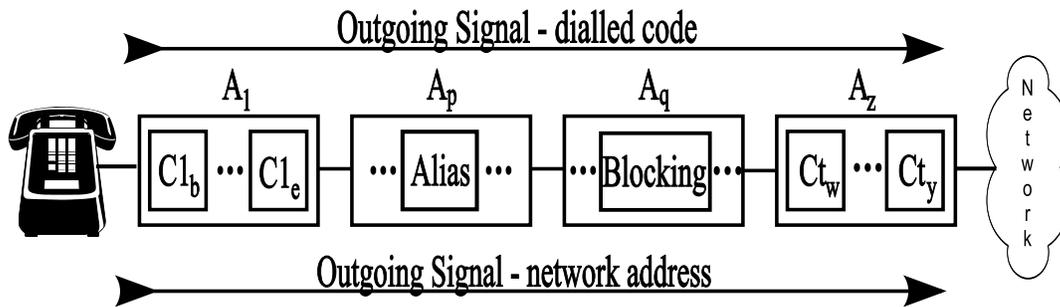


Figure 4.13. This figure shows that a feature category **Alias** is in address zone A_p , while the **Blocking** category is in the address zone A_q .

Outgoing *setup* signal to dialled code (source region): The **Alias** feature in zone A_p is triggered by receipt of the *setup* signal and translates the dialled code into its corresponding network address before continuing the *setup* signal towards the **Blocking** feature in zone A_q . When the **Blocking** feature receives this signal, it uses the network address for comparison against the blocking list to determine whether or not the call attempt should be blocked. The information recorded in the alias field is cleared between address zones,⁶ and hence the dialled code is not used to prevent the call attempt, which is correct, since the dialled code may have different interpretations in different address zones. Hence, the call will be appropriately blocked and no violation of the *Abortion* principle occurs.

Outgoing *setup* signal to network address (source region): Similarly, there is no violation of the *Abortion* principle in this call scenario. The **Alias** feature in zone A_p is triggered by receipt of the *setup* signal and translates the network address into its corresponding code, which is stored in the alias field if such a code exists. The *setup* signal is then continued towards the **Blocking** feature in zone A_q , which determines whether or not the call attempt should be blocked, using the network address for comparison against the blocking list. Again, the alias field is cleared between address zones and is not used when testing against the blocking list. Hence, the call will be appropriately blocked and no violation of the *Abortion* principle occurs.

⁶In the previous cases, the dialled code remains in the target address field, where it is stored until it is translated into its corresponding network address. Thus, the information is not cleared when switching address zones.

Case 2.2: GenCall - Blocking

Assume the features are ordered as shown in Figure 4.14. A new call attempt can be generated only in the source region⁷. Therefore, we consider 1) the passing of a *triggering* signal, usually a *feature-specific* signal, in the source region, which triggers a GenCall feature to initiate a call attempt or 2) the spontaneous generation of a call attempt by a GenCall feature.

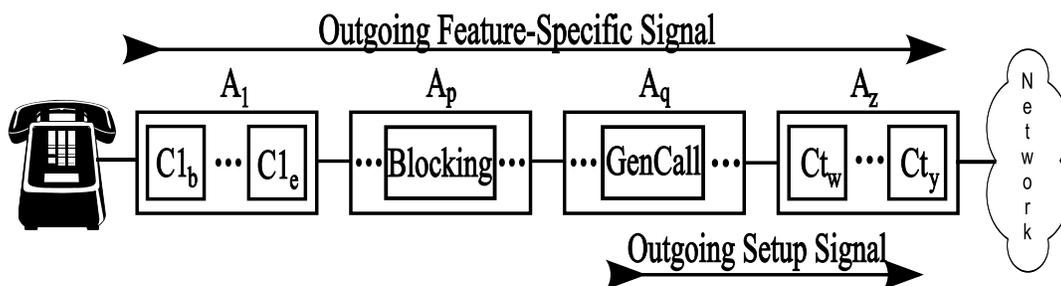


Figure 4.14. This figure shows that a feature category Blocking is in address zone A_p , while the GenCall category is in the address zone A_q .

Outgoing *triggering* signal (source region): The Blocking feature in zone A_p receives the *triggering* signal and behaves transparently, continuing the *triggering* signal. The *triggering* signal continues towards the GenCall feature in zone A_q and triggers this feature, so that a call attempt is placed to a target address determined by the feature or extracted from the *triggering* signal. If the target address is not found on the blocking list of the Blocking feature in zone A_p , then the *Abortion* principle is not violated.

However, if the determined target address is found on the blocking list of the Blocking feature in zone A_p , then the *Abortion* principle is violated. This violation occurs because the call attempt is continued by the GenCall feature and a voice connection is established with another user whose network address is found on a blocking list of a Blocking feature in the source region of the call path. According to the principles of IAT, this is the correct solution, since the caller chooses to act in the role of A_q when using A_q 's GenCall feature to identify the network address for the callee. For example, a user may block all long distance calls placed from their home address, but be required to return the call of a long-distance client for work. Hence, it would be acceptable when calling from home through a work address to connect a long-distance call initiated in the work address.

Spontaneous call generation (source region): Consider the case in which the GenCall feature in zone A_q has been activated, and it automatically generates a call attempt to a stored ad-

⁷A Target Redial feature records incoming call information in the target region, but when it initiates a call attempt, this occurs in the source region

dress on behalf of zone A_q (e.g., a Redial feature). Usually, the GenCall feature queries the availability of the callee so that the subscriber can be notified when the callee is available to receive a call, but this feature could simply be trying to connect with the callee in an attempt to play a message on behalf of the caller. In the first case, when the callee is deemed available, the GenCall feature generates another call attempt to the subscriber, with notification that a connection can possibly be established with the callee. In this more complex example, the *Abortion* principle is not violated during the initial call query, because zone A_p is not involved in the call attempt that queries the callee.

However, when the subscriber responds to the notification from the GenCall feature⁸, the subscriber sends an outgoing *feature-specific* signal to activate the call attempt using GenCall. This scenario is now equivalent to the outgoing triggering signal (source region) scenario (above), where violations of the *Abortion* principle were shown to be correctly resolved.

Next, we consider the alternate feature ordering shown in Figure 4.15. Once again, we separate our analysis based on the case of whether 1) a feature-specific signal triggers the GenCall feature to initiate a call attempt or 2) the call attempt is spontaneously generated by a GenCall.

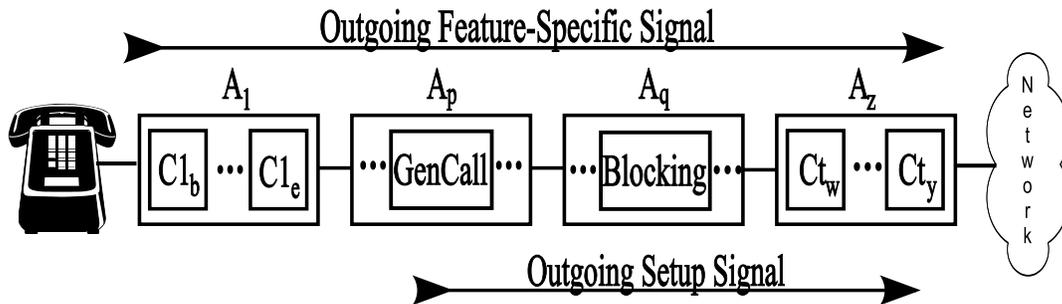


Figure 4.15. This figure shows that a feature category GenCall is in address zone A_p , while the Blocking category is in the address zone A_q .

Outgoing feature-specific signal (source region): The GenCall feature in zone A_p is triggered by the receipt of a *feature-specific* signal, which generates a call attempt to a target address by outputting a *setup* signal. The target address is determined by the feature or extracted from the *feature-specific* signal. The *setup* signal continues towards the Blocking feature in zone A_q , which determines whether or not the call attempt should be blocked. Hence, the call will be appropriately blocked and no violation of the *Abortion* principle occurs.

⁸We assume that the GenCall feature was triggered by the subscriber associated with zone A_q and that the notification sent to the subscriber follows the same call path that was established between the user and A_q when the user invoked GenCall.

Spontaneous call generation (source region): The GenCall feature in zone A_p automatically generates a call attempt to a network address on behalf of zone A_q to query the availability of the callee. The *setup* signal is output towards the Blocking feature in zone A_q . When the Blocking feature receives this signal, it determines whether or not the call attempt should be blocked. Consequently, the call is appropriately blocked or continued and no violation of the *Abortion* principle occurs.

Case 2.3: RedCall - Blocking

Assume the features are ordered as shown in Figure 4.16. A new call attempt can be redirected in the source or target region upon receipt of the *setup* signal or any *failure* signal. The *Abortion* principle can be violated only during the initialization of a call attempt, so we consider only call scenarios that involve the *setup* signal, separating the subcases into incoming and outgoing *setup* signal.

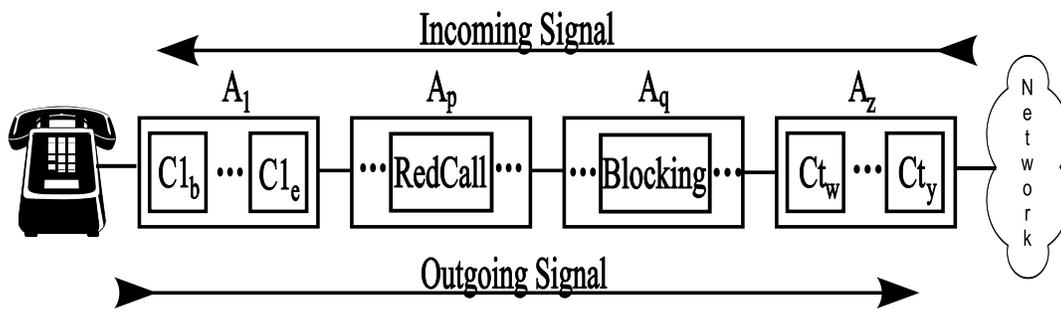


Figure 4.16. This figure shows that a feature category RedCall is in address zone A_p , while the Blocking category is in the address zone A_q .

Incoming *setup* signal (target region): The Blocking feature in zone A_q is entered first, and the call attempt is either blocked or continued. If the call attempt is blocked, then the call attempt is terminated and there is no violation of the *Abortion* principle.

If the incoming call attempt is not blocked, then the *setup* signal continues towards the RedCall feature in A_p , which redirects the call attempt to another target address zone. Address A_p 's RedCall feature causes the call to be redirected to another address zone. If this new address zone is not found on the blocking list for a feature in A_q , then no violation of the *Abortion* principle occurs. However, if the redirection is to an address zone found on the blocking list of a feature in A_q , then a violation of the *Abortion* principle occurs, since the call attempt reaches an address zone that should have been blocked. However, according to the principles of IAT, this is the correct solution, since the caller chooses to

act in the role of A_p when using A_p 's RedCall feature to identify the network address for the callee, and thus this is an acceptable violation.

Outgoing setup signal (source region): The RedCall feature in zone A_p is entered first and redirects the call attempt to another address zone A_q . The setup signal continues into the new address zone A_q where it enters the Blocking feature. The call attempt is then compared against the feature's blocking list and is blocked or continued accordingly, hence there is no violation of the *Abortion* principle as the call attempt is terminated or continued appropriately.

Next, we consider the alternate feature ordering shown in Figure 4.17. Once again, we separate our analysis of the cases based on the direction (incoming or outgoing) of the setup signal and the region (target or source) of the interaction.

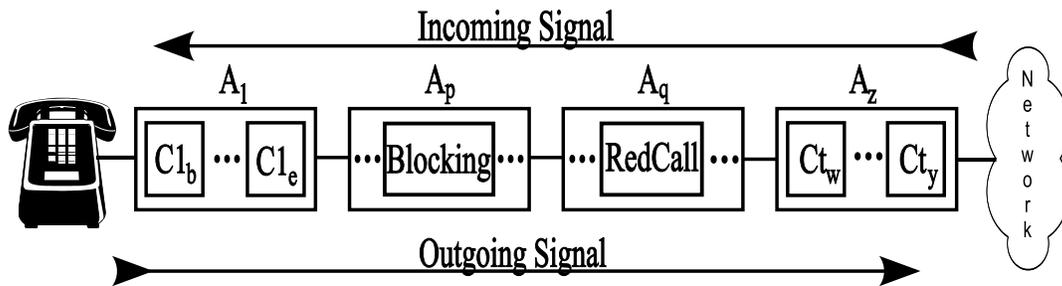


Figure 4.17. This figure shows that the Blocking category is in address zone A_p , while a feature category RedCall, which can be any feature category that can redirect a call, is in the address zone A_q .

Incoming setup signal (target region): The RedCall feature in zone A_q is entered first and redirects the call attempt to another address zone A_p . The setup signal is continued and enters the Blocking feature in zone A_p . The call attempt is then compared against the feature's blocking list and is blocked or continued accordingly, hence there is no violation of the *Abortion* principle.

Outgoing setup signal (source region): The Blocking feature in zone A_p is entered first, and the call attempt is either blocked or continued. If the call attempt is blocked there is no violation of the *Abortion* principle, since the call attempt is terminated.

If the incoming call attempt is not blocked, then the setup signal continues towards the RedCall feature in A_q . Address A_q 's RedCall feature causes the call to be redirected to another address zone. If this new address zone is not found on the blocking list for a feature in A_p , then no violation of the *Abortion* principle occurs. However, if the redirection is

to an address zone found on the blocking list of a feature in A_p , then a violation of the *Abortion* principle occurs, since the call attempt reaches an address zone that should have been blocked. However, according to the principles of IAT, this is the correct solution, since the caller chooses to act in the role of A_q when using A_q 's *RedCall* feature to identify the network address for the callee, and thus this is an acceptable violation.

Case 3: The Failure Principle

The *Failure* principle states that any feature that can respond to a *failure* signal should be in a position to receive any generated *failure* signals. The *Failure* principle can only be violated in the target region, where *failure* signals are generated. A *failure* signal can be generated by either a *Set Outcome* feature or the end-device. Therefore, the subcases to consider are:

1. Set Outcome - FailCall
2. FailCall - FailCall

where *FailCall* is any feature category (i.e., *Delegate*, and *Target Redirect*) that can respond to a failed call attempt.

Case 3.1: Set Outcome - FailCall

Assume the features are ordered as shown in Figure 4.18. The *Failure* principle can be violated only when a call attempt fails, so we consider call scenarios where either 1) the end-device or 2) some *Set Outcome* feature generates a *failure* signal.

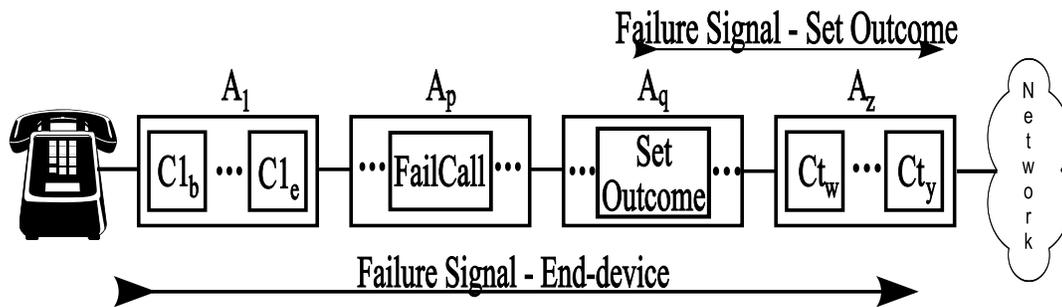


Figure 4.18. This figure shows that a feature category *FailCall* is in address zone A_p , while the *Set Outcome* category is in the address zone A_q .

End-device-generated *failure* signal (target region): The call attempt generates the call path shown in Figure 4.18. When the end-device receives the incoming *setup* signal, it responds by issuing a *failure* signal. This signal enters the *FailCall* feature in zone A_p first, and the failed call attempt is treated according to the functionality of the feature. There is no violation of the *Failure* principle.

Set-Outcome-generated *failure* signal: The call attempt generates the call path shown in Figure 4.18, but only up to the **Set Outcome** feature in zone A_q , which immediately outputs a *failure* signal back towards zone A_z . Thus, the **Set Outcome** feature prevents the call attempt from reaching the zone A_p , which results in a violation of the *Failure* principle, since the **FailCall** feature in A_p cannot respond to this *failure* signal. However, because zone A_q is more abstract than zone A_p in the target region, it has priority with respect to responding to incoming signals, according to IAT principles, and thus this is an acceptable violation. The result of this precedence is that A_q acts as a gate that restricts signals from accessing the more concrete address A_p .

Next, we consider the alternate feature ordering shown in Figure 4.19. Once again, we separate our analysis of the cases based on whether 1) the end-device or 2) some **Set Outcome** feature generates the *failure* signal

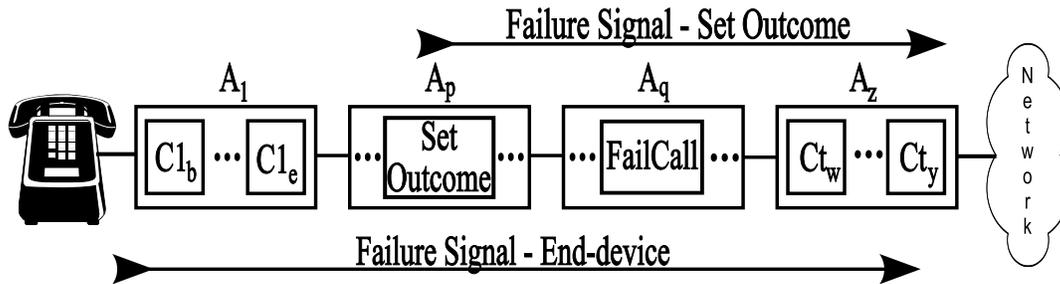


Figure 4.19. This figure shows that the **Set Outcome** category is in address zone A_p , while a feature category **FailCall**, which can be any feature category that reacts to a *failure* signal, is in the address zone A_q .

End-device-generated *failure* signal (target region): The call attempt generates the call path shown in Figure 4.19. When the end-device receives the incoming *setup* signal, it responds by issuing a *failure* signal. This signal enters the **Set Outcome** feature in zone A_p first, which simply continues the *failure* signal. When the signal reaches the **FailCall** feature in zone A_q , the feature responds by performing a failure treatment on the failed call attempt. Thus, there is no violation of the *Failure* principle.

Set-Outcome-generated *failure* signal: The call attempt generates the call path shown in Figure 4.19, but only up to the **Set Outcome** feature in zone A_q , which immediately outputs a *failure* signal back towards zone A_z . The *failure* signal is continued and reaches the **Fail-Call** feature in zone A_q , the feature responds performing a failure treatment on the failed call attempt. Thus, there is no violation of the *Failure* principle.

Case 3.2: FailCall - FailCall

Assume the features are ordered as shown in Figure 4.20. The presence of two FailCall features in different target-region address zones of a call attempt means that, if the call attempt fails, only one of the two features will perform its failure treatment. We analyze this call scenario by considering the receipt of an outgoing *failure* signal in the target region.

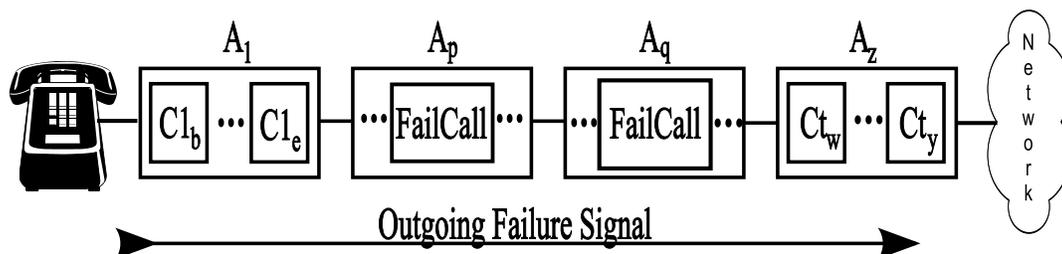


Figure 4.20. This figure shows that a feature category FailCall is in both address zone A_p and A_q .

Outgoing failure signal (target region): A failure signal is generated and sent along the call path shown in Figure 4.20. The signal enters the FailCall feature in zone A_p first, and the failed call attempt is treated according to the functionality of this feature. However, the failure treatment does not involve continuing the *failure* signal towards zone A_q , hence A_q 's FailCall feature does not get to respond to this signal and a violation of the *Failure* principle occurs. The default resolution according to IAT principles gives priority to A_p 's feature with respect to outgoing signals, since it is in the more concrete address zone. Thus, this is an acceptable violation.

However, as previously discussed, IAT designers noted that this might not be the correct resolution in all call scenarios and added *near-party* signalling as another IAT protocol that allows modification of the priorities for features in this call scenario (see Section 4.6). When the FailCall feature in zone A_q is initialized, it may issue a *near-party* signal. This *near-party* signal alerts the FailCall features in more concrete address zones, such as A_p , to behave transparently on receipt of any *failure* signal. Therefore, if a *near-party* signal is received by zone A_p 's FailCall feature, then the *failure* signal continues unchanged towards zone A_q , where its FailCall feature applies its failure treatment to the call attempt. Thus, users have the ability, via *near-party* signalling to change the default resolution for this principle violation.

This concludes the validation of the cases that cover three of the ten principles described in Section 4.2. The case analysis for the remainder of the principles can be found in Appendix B.

In the following chapter, we discuss how the categories and principles identified in this chapter can be modelled in Prolog and used to automatically generate optimal category orderings.

Chapter 5

Using Prolog to Automatically Generate the Partial Order

To support our category prioritization approach presented in Chapter 4, we developed a Prolog model to automate the generation of optimal category orderings in a more efficient and less error prone manner. Recall from Section 4.4 that an **optimal** ordering is any ordering that is violation-free or, if no such ordering exists, any ordering that has the smallest criterion violation count. Our Prolog model simulates the telephony environment described in Chapter 2. In this environment, a telephone call is formed by incrementally adding feature modules, interface modules, end devices, and network components in a serial ordering in an attempt to establish a user connection (i.e., a voice connection). Signals travel along the call path between the various feature modules and utilize system information to aid in call routing. The features are added to the call path in a specific ordering to reduce the occurrence of feature interactions. To calculate the priority ordering that determines the order in which features are added to the call path, we begin by abstracting features into a smaller set of categories and then prioritizing these categories.

The category prioritization is based on the categories and principles identified in Chapter 4, where features representing each category and assertions corresponding to the constraint and criterion principles are incorporated into our Prolog model. The principle assertions are used to identify unacceptable feature orderings that result in feature interactions, so that after searching all possible call scenarios, our Prolog model returns a list of optimal orderings between the feature categories.

The rest of this chapter is organized as follows. Section 5.1 gives a general overview of our Prolog model. Section 5.2 presents the data structures created to represent a call in our Prolog model. In Section 5.3, we summarize the execution model for how calls are generated. Section 5.4 describes how feature transitions are modelled, and Section 5.5 describes how during execution a feature transition updates the execution model. Section 5.6 presents the principle assertions and how assertions help determine optimal feature orderings. Section 5.7 describes

the output, such as the set of all feature category orderings that violate constraint principles, generated by our Prolog model. Finally, in Section 5.8, we discuss optimizations that truncate exploration of invalid call scenarios to increase effectiveness.

5.1 General Design Overview

Prolog, short for Programming in Logic, is a declarative programming language based on predicate calculus [44]. Prolog uses a logic paradigm, with backwards chaining (i.e., goal-driven reasoning) to automatically solve queries. The backwards chaining employs backtracking and pattern matching as it attempts to find a solution to its goal. When Prolog it performs an exhaustive search over all instantiations of the facts and rules contained in the Prolog database to return all possible solutions to its goal.

We chose to model our telephony environment in Prolog because there is a clear mapping between our telephony environment and the database of facts and rules that Prolog uses: our feature data can be represented as Prolog facts, our feature transition rules can be represented as Prolog inference rules, and our principles can be represented as Prolog assertion rules. A Prolog query in our model is designed to return the list of optimized priority orderings, as well as some other related outputs, for a given set of categories and principles. For clarity, we refer to the representation of our telephony environment in Prolog as the **Telephony Prolog Model** (TP model).

The TP model takes as input the feature categories to be prioritized and the principles that define acceptable system and feature behaviour. Each feature category is represented by a **Category Representative Feature** (CRF) that is encoded as a set of transition rules that mimic the feature’s finite-state-machine behaviour of reacting to input signals. Each principle is expressed as an assertion over variables representing the current state of the call/call attempt; if the assertion evaluates to false, then the principle is violated.

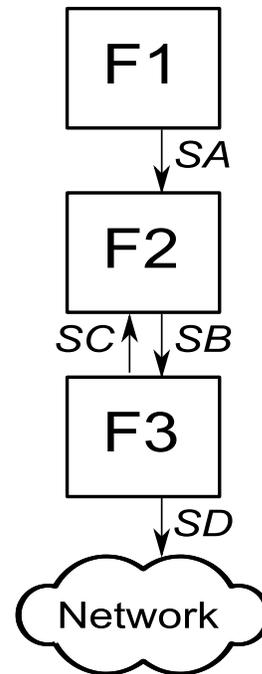


Figure 5.1. Feature Composition: Feature F1 outputs signal SA in a forward direction into feature F2. Feature F2 outputs signal SB in a forward direction input into feature F3. Finally feature F3 outputs 2 signals, SC in a reverse direction into feature F2, and SD in a forward direction into the network.

In our telephony environment (see Chapter 2), the feature and interface modules of a call are ordered serially. The TP model simulates this composition by mapping the output signals from one feature to the input(s) of its neighbouring features, as shown in Figure 5.1. Signals flow along the composition in both directions; signals sent in a forward direction towards the callee are input into the next (lower) neighbour in the call path, whereas signals sent in the reverse direction towards the caller are input into the previous (upper) neighbour in the call path. In this chapter, we change the orientation of a call path from horizontal to vertical, so that we have more space to show the branching that occurs when Prolog backtracks and explores an alternate evaluation along a call path for a particular call scenario.

The three main concepts used in simulating the execution of a call attempt in the TP model are **call state**, **feature transition rules**, and **principle assertions**. **Call states** represent the current execution state of each call attempt and the call attempt's effect on the model's environment (e.g., call information in the database). A **feature transition rule** is defined for each feature, which simulates the feature's behaviour upon receipt of a signal. **Principle assertions** represent feature interactions, or the absence of a particular feature interaction, in the TP model and are used to identify invalid feature orderings.

To identify all acceptable feature orderings, the TP model explores all possible feature category orderings and evaluates the acceptability of each. Each possible ordering is combined with a specific set of feature data (e.g., call forwarding information), which forms a distinct **call scenario**. For each call scenario, all potential call paths are generated, where each **call path** represents one possible execution path of a call. For example, the callee may either answer a call, not answer a call, or not be available (already on the phone) when she receives an incoming call. These different environmental situations result in the creation of three call paths, and the TP model explores all possibilities.

Recall from Section 2.3 that a call path is split into multiple **call segments** whenever a call path contains a **Multiplex** feature. Each call segment represents a portion of the call path that can be linked to other call segments to form a complete call path. Whenever a **Multiplex** feature is added to a call path, it causes the call path to generate a new call segment so that multiple call paths can access the same call segment, since only one instance of each **Multiplex** feature can be active at any given time for a specific address zone. Thus, in the TP model, each call path is represented as a sequence of one or more call segments that link together to form the complete call path.

After simulating all of the possible call scenarios and identifying unacceptable feature-category orderings, the TP model outputs a set of **call trees**, a set of **signal tables** relating to the call trees, a **Violation-Free List (FreeVioList)**, a **Criterion-Violation List (CritVioList)**, a **Constraint-Violation List (ConVioList)** and a **Allowable Criterion Violation List (AllowCritList)**. A **call tree** is created for each call segment that is generated during simulation and represents the

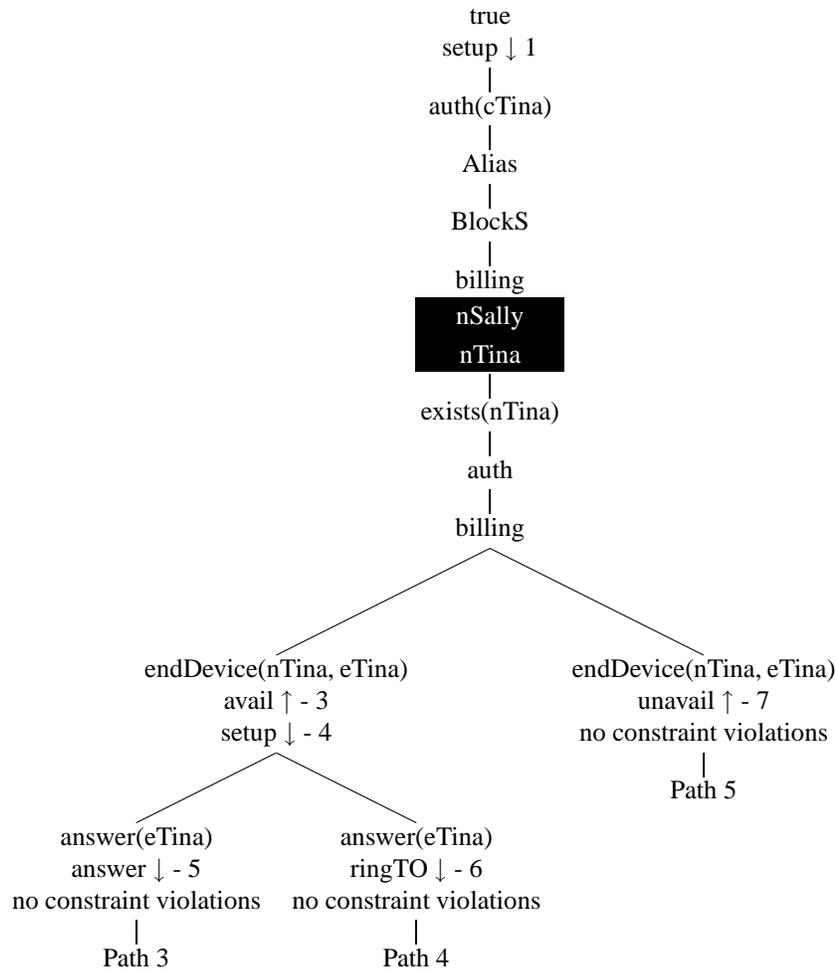


Figure 5.2. Call Tree call1aBbS2: This call tree, named call1aBbS2, shows one of the call segments that forms a specific call path. The first branch on the tree is generated based on whether or not the end device is available, while the second branch explores whether or not the callee answers the incoming call.

possible call paths that could execute when a *setup* signal is propagated through a call segment. When two call paths are based on the same category ordering and the same feature data, and differ only in the inputs and responses from the call's environment (e.g., whether the callee answers the phone or not), their corresponding call segments are merged into the same call tree. Figure 5.2 shows the call tree representing the different call paths generated for a specific feature ordering and a given set of feature data where all possible paths towards the callee's end device are explored. A **signal table** is recorded for each call path in a call tree and identifies, in order, all of the signals that were generated and received by the features during the simulation of the call path. Table 5.1 shows the signal table corresponding to the call path labelled 4 on the call tree in Figure 5.2. The table shows that the *setup* signal initialized the call attempt and is continued by

Table 5.1. Signal Table

Signal Table: **Path 4 for call1aBbS2**

Module	Signal 1	Signal 3	Signal 4	Signal 6
auth(nSally) - Dialed(cTina)	setup ↓	avail ↑		ringTO ↑
AliasS(nTina cTina)	setup ↓	avail ↑		ringTO ↑
BlockS('not blocked')	setup ↓	avail ↑		ringTO ↑
billing	setup ↓	avail ↑		ringTO ↑
exists(nTina)	setup ↓	avail ↑		ringTO ↑
auth(nTina)	setup ↓	avail ↑		ringTO ↑
billing	setup ↓	avail ↑		ringTO ↑
endDevice(eTina) - avail	setup ↓	avail ↑		ringTO ↑
answer(eTina) - ringTO			setup ↓	ringTO ↑

This signal table identifies the different signals sent along the call path with the label 4 found on the above call tree, call1aBbS2.

the feature modules in the call path, as the modules are added to the call path. When the *setup* signal enters the callee's end device, the device is found to be available and hence *avail* is sent back towards the caller through all the features in the existing call path. Nearly simultaneously, the end device continues the call attempt sending the *setup* forward towards the callee, which presents the incoming call to the callee (e.g., the device starts to ring). In this call path, the end device is not answered after a set amount of time and a *ringTO* signal is sent back towards the callee.

In addition to the graphical representation of the explored call paths, TP model outputs summary information about the feature interactions that were detected during call simulation. The **Violation-Free List (FreeVioList)** is a list of feature orderings that do not violate any principles; all of the orderings in this list are optimal orderings. The **Criterion-Violation List (CritVioList)** lists feature orderings that are known to result in a criterion violation, whereas the **Constraint-Violation List (ConVioList)** lists the feature orderings that can result in a constraint violation. The **Allowable Criterion Violation List (AllowCritList)** is a list of feature orderings that violate criterion principles, but that are accepted as optimal orderings if the system is overly constrained. The **AllowCritList** will only contain elements if **FreeVioList** is empty.

5.2 Modelling Abstractions

In this section, we identify and describe some of the data structures used to model our telephony environment in Prolog. After presenting a detailed description of the principle data structures, we end with a list of other data structures that maintain information about calls and search results. Finally, we conclude this section with an explanation of how we represent a category in the TP model using a **Category Representative Feature (CRF)**.

5.2.1 Call Stage Routing Information

The call's current point of execution is the feature module within the call path, where the signal is currently initializing or traversing the module. The routing-related information for the current point of execution is always determined during the initialization of the feature module. To track a call's current point of execution and future path, we record the following routing-related information for each call.

Zone: the address zone in which the current signal is currently being processed by a module, or in which the *setup* signal is building the call path.

Region: the region in the call path that the current signal is traversing: this can be either the **source** or **target** region

Source: address of the source region. When the current signal is in the source region, the values of the Source and Zone fields will be equal, unless a source redirection has just been initiated. When the current signal is in the target zone, the Source field will hold the value of the most recently traversed source address.

Target: address of the target region. When the current signal is in the target region, the values of the Target and Zone fields will be equal, unless a target redirection has just been initiated. When the current signal is in the source zone, the Target field will hold the target address or dialled code provided by the caller.

Alias: the code/alias value for an address in the opposite zone. In the source zone, feature **AliasS**, if in the call path, maps the dialled code to a target address and stores the originally dialled code in the Alias field. Analogously, in the target region, feature **AliasT**, if in the call path, maps the caller's address to the corresponding alias, and stores the alias in the Alias field.

In the signal routing information, we distinguish between addresses and aliases by prefacing addresses with n (numerical address) and prefacing aliases with c (code). For example, when an **AliasS** feature is added to the call path, the feature executes and 1) changes the target field in the routing information from the dialled code, cTommy, to its corresponding address, nTommy, and 2) moves the dialled code into the alias field. The dialled code is stored in the alias field to allow other features to access and use this information. For example, a **Source Redirect** feature can selectively chose to redirect an outgoing call attempt based on either the target's numerical address or its alias (i.e., a subscriber always redirects outgoing calls to the alias "MyBoss" through the subscriber's work address).

The alias information is always discarded when the call attempt is routed from the source region to the target region, since alias information is subscriber and address specific. For example, if the alias parameter holds the term “Mom”, this may refer to a different person when the region is changed. For the same reason, our model also discards alias information whenever the call attempt is routed into a different address zone in the same region (e.g., the alias “Tom” in the subscriber’s home address may refer to a friend, but the same alias “Tom” in the subscriber’s work address may refer to a co-worker).

5.2.2 Call Stage (CS)

To track the progress of a call in the TP model, we found that it is advantageous to separate each call into stages. Call Stages are not strictly necessary to implement the simulation of a call. However, using call stages simplifies the implementation of principle assertions, as some of the principle assertions need to be checked only when a *setup* signal reaches a particular point in the call-attempt process. The call-stage data structure provides an easy way to track these execution points in a call attempt. For example, the *abortion* principle is checked whenever the *setup* signal reaches the Network and begins routing the call into a new address, validating that this call attempt should not have been terminated. Using call stages also reduces the number of transitions to be considered for execution, because each transition is designed to be executed in exactly one call stage.

In Figure 5.3, Table *A* shows the call stage and routing information associated with a call attempt that is terminated by a feature in the caller’s address zone. In Table *B*, we show the call stages and routing information associated with a call attempt that leads successfully to a voice connection with the callee. In fact, Table *B* represents the progress of call path 4 in the call tree of Figure 5.2. The data shown in these tables correspond to the call stage and routing information that existed during the initialization of the call path.

Below we explain the different call stages:

cValid: the call validation stage, where a call is tested to determine if this address is able to receive/place calls and set up authentication requirements for any of the source (performed instantly) or the target (performed when the call is answered) addresses. A call normally starts in the call validation (*cValid*) stage (Figure 5.3, lines 1a, 1b) and reenters this stage whenever the call enters a new address (Figure 5.3, line 5b). For example, a device may restrict usage or require a passcode before allowing a call to be placed. After the call has been successfully initialized and authenticated, the call progresses to the addition (*cAdd*) stage (Figure 5.3, lines 2a, 2b).

cAdd: the addition stage, where features associated with the current address are added to the call path and their initialization behaviour is simulated. Normally, when a feature is added

cValid	(source, nSally, nSally, cTommy, empty)	1a
cAdd	(source, nSally, nSally, nTommy, empty)	2a
	(source, nSally, nSally, nTommy, empty)	3a
cEnd	(source, nSally, nSally, nTommy, empty)	4a

(A)

cValid	(source, nSally, nSally, cTommy, empty)	1b
cAdd	(source, nSally, nSally, cTommy, empty)	2b
	(source, nSally, nSally, nTommy, cTommy)	3b
netLink	(source, nSally, nSally, nTommy, cTommy)	4b
cValid	(target, nTommy, nSally, nTommy, empty)	5b
cAdd	(target, nTommy, nSally, nTommy, empty)	6b
netLink	(target, nTommy, nSally, nTommy, empty)	7b
cAnswer	(target, nTommy, nSally, nTommy, empty)	8b
vLink	(target, nTommy, nSally, nTommy, empty)	9b

(B)

In Figure A, the call ends when the call is terminated (e.g., by a **Blocking** feature) before the call can transition into the target zone. In Figure B, the call progresses successfully through the source and target zones and establishes a voice connection between Sally and Tommy.

Figure 5.3. Call Stage Progression

to the call path, it will have little or no effect on the call stage, although some features may modify a portion of the routing-related information (Figure 5.3, lines 3a, 3b). However, a few features have the ability to change the status of the call stage. For example, a **Blocking** feature, which prevents calls to specific addresses, can cause a call to abort by progressing to the call end (**cEnd**) stage (Figure 5.3, lines 4a). Once all features in the current region have been applied, or when a feature forces a change in the call's address (redirection), the call transitions to the network link (**netLink**) stage (Figure 5.3, lines 4b, 7b).

netLink: the network link stage, where the call is routed from one address zone to another address zone, possibly in the alternate region. For example, when changing from the source (caller) zone to the target (callee) zone of a call, the target address is calculated. The calculated address is added to the call path and used to continue the call. Alternatively, after adding all of the target (callee) features, the call is transferred to the interface module of the target's device.¹ When the call transitions from **netLink** into the target's device

¹We chose not to model interface modules in TP model, since we can choose to use a standard set of signals that do not require translation when entering a new device.

(Figure 5.3, line 7b), the call enters the call answer (*cAnswer*) stage (Figure 5.3, line 8b).

cAnswer: the call answer stage, where the user device receives the call attempt and responds accordingly (i.e., the telephone rings to alert the user to an incoming call). In this stage, the device reports its availability, sending out either an *avail* or *unavail* signal.² When the device is available, it also notifies the user of the incoming call. When a device is answered, the call progresses to the final voice link (*cVLink*) stage (Figure 5.3, line 9b), indicating that the call is successfully connected and communication is established between the source and target.

cVLink: the voice link stage, which is entered when a voice connection between parties is successfully established. In this stage, the call is established and the users can communicate with one another. If there are no further actions to be applied for this call (i.e., there are no user input signals to be processed), then the call enters the *cDone*.

cEnd: the call end stage, which is entered whenever a call is terminated, or whenever the call is aborted for any reason. As discussed above, the call attempt can be terminated by a feature or the call attempt may be terminated when one of the user's issues a *teardown* signal.

cDone: the call done stage, which is entered when a call has finished processing all of the call's transitions and has finished applying all of the user-initiated signals to the call. The call done (*cDone*) stage indicates that no further action will be applied by the TP model with respect to this call. This stage is a Prolog-simulation stage that indicates to Prolog that the call attempt has been fully simulated.

In keeping with the information presented in Figure 5.3, we create the data structure, **Call Stage (CS)**, which takes the following form:

```
CS = CallStage(Region, Zone, Source, Target, Alias)
```

where **CallStage** is a value representing the current stage of this call, and where the parameters (**Region, Zone, Source, Target, Alias**) record signal routing information about this call's state.

5.2.3 Segment Stage (SegStg)

A **Segment Stage (SegStg)** is similar to a **Call Stage (CS)** described in Section 5.2.2, however the **SegStg** specifically refers to the stage of execution of the call segment rather than to the stage

²Traditionally, a *setup* and *teardown* signal involves the use of the verification signals *upack* and *downack*. For simplicity, we chose not to model these two signals; since these signals are used to confirm a solid connection between neighbouring features in the call path before sending any subsequent signals. Therefore, removing these signals does not affect the results of our model.

of the call. **SegStg** is used to identify the active call segment, where the call's point of execution is located, and to identify completed call segments that are linked to other call segments and that form a portion of the call.

Each call has exactly one **active call segment**, whose segment stage (**SegStg**) is analogous to the call stage (**CS**) (e.g., if **CS** = *cAdd*, then **SegStg** = *sAdd*). The segment stage of each non-active call segment represents the status of the call segment. For example, the *sConnected* call segment status indicates that its corresponding call segment is linked to another call segment. No new features or interface components can be added to a non-active call segment. The term **executing call segment** refers to the call segment that contains the point of execution for this call in the TP model. Normally, the *executing call segment* is the *active call segment*, however signals may be sent along established call paths, in which case the executing call segment may refer to a completed call segment along which the signal is travelling.

The following list describes the different stages of a call segment:

sAdd: the segment addition stage, in which the call segment is the *active call segment*, and new features and interface components are still being added to the call segment (corresponds to call stage *cAdd*).

sAnswer: the segment answer stage, in which the call segment is the *active call segment*, and the target device is presenting the incoming call to its owner (corresponds to call stage *cAnswer*).

sVLink: the segment voice link stage, in which the call segment is the *active call segment*, and the call has been answered (i.e., established) and the users are communicating (corresponds to call stage *cVLink*).

sStartCon: the stage, in which the call segment is requesting the creation of a new call segment; the new call segment becomes the active call segment for this call. For example, when the **Multiplex** category is added to the call path in the source region, the next feature added to the call path forms the start of a new call segment.

sConnected: the start connection segment connected stage, in which the call segment is linked to another call segment. Signals can travel along a connected call segment (i.e., be the executing call segment) and may trigger feature reactions, such as the generation of a new signal, but a connected call segment can never again be an active call segment.

sComplete: the segment complete stage, in which the call is established and no user-input signals are left to be searched. When a call segment enters this stage, no further actions can be applied to this call segment, unless there is influence from an outside call (e.g., when

another call links to a multiplex feature belonging to this call segment) (corresponds to call stage *cDone*).

5.2.4 Call Representation in Prolog (CallID)

Each call is formed by connecting one or more call segments. In the TP model, the instantiation of a call segment is represented by a **Call Identifier (CallID)** data structure. **CallID** stores such information as whether the call segment has been linked to another call segment, whether another feature should be added to the call segment, and what criterion principles have been violated thus far during the call:

Call Name (CN): the name of the call segment.

Call Stage (CS): the data structure described in Section 5.2.2, which indicates the call's current stage (e.g., *cValid*, *cAdd*, *netLink*) and contains signal routing information.

Segment Stage (SegStg): the segment stage described in Section 5.2.3, which indicates the progress of the call segment and determines whether any transitions remain for this call segment to execute.

Ordered Feature List (OFL): an ordered list of all features or **CRFs** that will be applied to this call attempt. The order indicates the order in which each feature will be added to the call path. The feature list is appended by a number, **OFL:Pos**, whose value represents the position in this feature list that refers to the feature currently being added by the call path. The values in **OFL** will change when the call is redirected to a new address zone.

Applied Feature List (AFL): an ordered list of all features that are already part of the call path for this call segment. **Applied Feature List (AFL)** is always a subset of **OFL** that starts at some position X in **OFL** and ends at $\text{Pos} + 1$.

Criterion Violation History List (CritHis): a list containing each criterion principle along with a marker that records whether or not this principle has been violated thus far in this call attempt.

In the TP model, one or more calls can be modelled and be at different stages of their simulation simultaneously. This allows the TP model to simulate the effects of multiple call attempts (i.e., second incoming call) on the behaviour of the feature categories. A **Call List (CList)** data structure keeps track of the set of calls currently being simulated. **CList** is a list of pairs [**CallID**, **InSig**], where each **CallID** is a call in progress and **InSig** is that call's next input signal.

5.2.5 Other Interesting Data Structures and Terminology of Interest

Below we list other key data structures and some terminology that is used in the remaining sections of this chapter. The three database data structures below store **dynamic** or **static** data. Recall from Section 2.4 that *static data* contains information that persists after the call is torn-down, whereas *dynamic data* contains information that pertains to the structure of the call and is removed when the call is terminated.

- **Call Database (CallDB)** is a list that contains dynamic call-specific information logged by features in the current call path. When a call segment is torn down (i.e., removed from the telephony environment), any related call information is removed from **CallDB**. **CallDB** is a list, which contains sublists that are indexed by the call segment's unique name. The database entries can also have a secondary index based on the name of the feature that owns the data. For example, an element of **CallDB** could be [*call1*, **Present**, [*Mom*, 555 – 1234]], where *call1*'s **presentT** feature has recorded the alias, *Mom*, and the telephone number, 555 – 1234, to be presented to the subscriber during the current call.
- **System Database (SysDB)** is a list that holds static database information, recorded or required by the features and the main telephony system, that persist beyond the life of a call. Examples of system database information include billing data and the last number dialed. **SysDB** information is permanently retained until explicitly removed or updated by a feature. **SysDB** elements have the same structure as **CallDB** elements.
- **Feature Data Database (FeatData)** is a database that holds subscription information and related feature data for all users in the system. For example, **FeatData** contains the list of features subscribed to by each user, and the additional feature data (e.g., the blocking list for the feature **BlockS**) recorded by the user to guide feature execution.
- **Call state** is defined as the current execution state of a call and the call's effect on the model's environment. We can determine the call state of any call in the TP model by retrieving from the databases **CallDB** and **SysDB** all entries associated with the **CallID** of the call's active call segment.
- **FreeVioList** is a list of all feature orderings that have been tested and whose simulations were found to be free of any principle violations. These orderings are always optimal orderings.
- **AllowCritList** is a list of all feature orderings that are considered optimal orderings even though their simulations were found to contain some criterion principle violations. This list will only contain orderings if **FreeVioList** is empty and the system is overly constrained

with respect to the principles. If any elements are found in this list, then these elements are also optimal orderings.

- **CritVioList** is a list of all feature orderings that have resulted in a criterion violation together with the principle violated and the region in which the principle was violated.
- **ConVioList** is a list of all feature orderings that have resulted in a constraint violation together with the region in which the principle was violated.

5.2.6 Category Representation

A **Category Representative Feature (CRF)** is an abstract feature representing a feature category in the TP model. The **CRF** corresponds to the generic features defined in Section 4.4. For each category, the **CRF** is chosen so that it performs the essential behaviour of any feature in its associated category. Thus, any feature in the category is represented by the category's **CRF**. For example, a **CRF** representing the source blocking category, **BlockS**, would either block or continue an outgoing call. This **CRF** would not log call information about blocked calls, as this action is not part of the basic functionality for this feature category and is performed only by specialized **BlockS** features. As discussed in Section 4.1.1, these multiple-purpose features are decomposed into two or more features that represent the features functionality.

The **CRF** is sufficient for the purposes of our analysis of the call path, because every feature within the category represented by a **CRF** performs the same underlying functionality. The fact that many of these features will also perform extra functionality does not diminish or negate the fact that the basic behaviour of the feature can cause a violation to occur. Therefore, a **CRF** is used to identify feature interactions that occur between different feature categories.

5.3 Execution Model

During execution, the TP model chooses a [**CallID**, **InSig**] pair from **CList**, where **CallID** represents an active call segment. The next feature to be executed or added to the call path and other useful information are extracted from the **CallID**. This information is used by Prolog to identify any enabled feature-transition rules. The TP model then executes the feature-transition rule to generate a new call state. Any alternative feature-transition rules will also be explored as part of Prolog's exhaustive search. When the execution of a feature-transition rule adds a new feature or interface component to the call path, the resulting call state is tested to determine if the principle assertions still hold. When the principle assertion does not hold, a principle violation is identified.

A principle violation indicates that a feature interaction exists, and appropriate action is taken based on the type of violation found (i.e., constraint or criterion). When no principle violation

is found or if no principle testing is required, **CList** is updated to hold the new call state and the next input signal. The TP model then repeats the above steps until no further transitions can be applied to any call.

In the following sections, the three execution steps, feature application, principle testing, and call-state updating, are presented in greater detail. The feature-transition rules that simulate feature behaviour are presented first followed by an explanation of how the TP model updates the call state using the output from feature-transition rules. The principle assertions are presented last, since testing principle assertions is orthogonal to the progress of the call's execution.

5.4 Feature-Transition Rules

Each feature in the TP model is encoded as a set of feature-transition rules that model the behaviour of the feature with respect to the signals it receives. Each feature-transition rule (**transrule**) is based on the feature box's finite-state-machine model, described in Section 2.4. Possible **transrule** reactions include propagating the input signal to a neighbouring feature; performing some action, such as updating the **Call Database (CallDB)** or the **System Database (SysDB)**; and outputting a new or modified signal.

Most **transrules** have the form

```
transrule(CList:CallID, CList:InSig,  
          NextCS, OutSig, NewCallID)
```

Input parameters, **CallID** and **InSig**, provide information about the executing call segment and input signals, respectively. Information extracted from these input parameters is tested against the preconditions of each **transrule** to determine whether the **transrule** is triggered in the current call state. For example, the Ordered Feature List (**CallID:OFL**) identifies the feature or interface component being executed, while the Call Stage (**CallID:CS**) is used to further narrow the selection of which **transrule** can execute. The triggered **transrule** can extract further details from a database, such as feature data values from **Feature Data Database (FeatData)**, to determine the appropriate action to be applied by this rule.

The output of the triggered **transrule** parameters, **NextCS** and **Output Signal (OutSig)**, indicates the changes that should be applied to the new call state and the signal type (SigT) and direction (Dir) of any signal generated by this transition, respectively. When a **transrule** outputs multiple signals, the signals are applied one at a time, with the first signal being fully processed by the call before the next signal is processed.³ Occasionally, a triggered **transrule** will also output a **NewCallID**, which represents either a new call spawned by the feature or a continuation of the current call via a link to a new call segment.

³It is unusual for a feature to simultaneously send out more than one signal in the same direction along the call path; allowing one signal to fully execute before the next signal is sent will not effect the progress of the call.

```

transrule(CList:CallID, CList:InSig, NextCS, OutSig):-
  1  % transrule entry constraints and
  2  % useful variables extracted from the input
  3  CList:CallID:OFL:FeatN = BlockT,
  4  CList:InSig:SigT = setup,
  5  CList:CallID:CS = cAdd(source,Src,Src,Tgt,Al),
  6
  7  % this is a target feature in the source region
  8  % behave transparently
  9  NextCS = CallID:CS,
 10  OutSig = CList:InSig.

```

Figure 5.4. BlockT Transrule (Initialization Source Zone)

A **transrule** may also update other components of the TP model. For example, **CallDB** is updated each time a **transrule** records or updates temporary call-specific information, such as the identification of the incoming call that is presented to the subscriber. As another example, **SysDB** is updated each time a feature records persistent information, such as the last number dialled by the feature’s subscriber.

The psuedo-code shown throughout this chapter uses the following terminology. The “;” represents the conjunction (and) of commands and the “;” represents the disjunction (or) of commands. The more complex structure “ $(A) \rightarrow (B); (C)$ ” represents an if-then-else construct; if A holds true then execute command B , else execute command C .

5.4.1 Transitions Rules

In this section, we present two examples of feature-transition rules. The first example is a basic propagation **transrule**, which behaves transparently by propagating the incoming signal along the call path without performing any action. The second example is a termination **transrule**, which terminates the call attempt if the target address matches an address found on the feature’s blocking list.

Example 5.4.1. Transparent Feature-Transition Rule: BlockT transrule

A transrule is executed every time a new feature is added to the call path. This BlockT transrule executes only if 1) the executing feature is BlockT, 2) its input signal is of type setup, 3) the call is in the cAdd stage, 4) the setup signal is in the source region, and 5) the Source and Zone parameters are equal. These constraints are expressed on lines 3-5 of Figure 5.4. Recall that each call stage is formed by combining its name and signal routing data, so that CS = csName(Region, Zone, Source, Target, Alias).

When a target (source) feature is entered while the call attempt is in the source (target) zone,

```

transrule(CList:CallID, CList:InSig, NextCS, OutSig):-
 1  % transrule entry constraints and
 2  % useful variables extracted from the input
 3  CList:CallID:OFL:FeatN = BlockS,
 4  CList:InSig:SigT = setup,
 5  CList:CallID:CS = cAdd(source,Src,Src,Tgt,Al),
 6
 7  % is this call blocked?
 8  (checkSubInfo(outblock(Src,Tgt,block)) ->
 9    ( % block call attempt
10      OutSig:SigT = errorBlock,
11      OutSig:Dir = rev,
12      NextCS = cEnd(source,Src,Src,Tgt,Al)
13    );
14    ( % propagate call attempt
15      OutSig = CList:InSig,
16      NextCS = CallID:CS
17    )
18 ).

```

Figure 5.5. BlockS Transrule (Initialization Source Zone)

*then the feature behaves transparently. Hence, when the **BlockT** feature is added to the call path in the source region, the feature behaves transparently by propagating both the current call stage and input signal (lines 9-10) unchanged. The TP model then adds this feature to the call path and passes the output signal to the next feature in the call path.*

Example 5.4.2. Abort Call Feature-Transition Rule: BlockS transrule

*In this example, we consider the **BlockS** transrule that executes when the feature is added to the call's source region. This transrule executes only if 1) the executing feature is **BlockS**, 2) its input signal is of type setup, 3) the call is in the **cAdd** stage, 4) the setup signal is in the source region, and 5) the **Source** and **Zone** parameters are equal. These constraints are expressed on lines 3-5 of Figure 5.5. When this feature is first added to the call path, it checks to determine whether or not this call attempt should be terminated or continued.*

*On line 8, **BlockS** transrule checks the subscriber database, **FeatData**, to see if the subscriber of zone Src blocks outgoing calls to target zone Tgt. If the call should be blocked, then this call is aborted by returning an error message back along the established call path and by setting the call status to **cEnd** (lines 10-12). Otherwise, **BlockS** transrule continues the call by behaving transparently and propagating both the current call stage and input signal (lines 15-16). The routing algorithm then adds this feature to the call path and continues the output signal into the next feature in the call path.*

5.5 Updating the Call State

Once the feature transition rule has been applied, the transition's effects are used to modify the call state, which simulates the progress of the call. The call state is updated by filling the **NextCallID** data field using the potentially modified contents of the current call state, **CallID**, and the input signal, **InSig**, together with the transition rule outputs. Most of the data fields, such as the call name, do not change when the call state is updated, so the corresponding **CallID** data fields are copied into **NextCallID**. The **transrule** output parameter, **NextCS**, is used to set the **NextCallID:CS** data field, since this value reflects the changes imposed on the current call segment by the feature transition rule. Under certain call scenarios (e.g., call initialization, call redirection), new values of **NextCallID** data fields are specifically generated by the feature-transition rule.

During call initialization, when a **transrule** executes in response to a *setup* signal, many of the call-state data fields need to be set to reflect the addition of this new feature or interface component to the call path. The **OFL:Pos** value is incremented by 1 and the executed feature or interface component is appended to the end of **AFL**. The **SegStg** data field is set to be *sAdd*, to reflect the updated progress of this call segment, and is changed only when a new call segment is generated, the call attempt is terminated, or an end device is reached. Finally, **CritHis** is updated to reflect any criterion principle violations that occur when the call stage **NextCS** is added to the call environment.

When a call redirection occurs, the generation of the **OFL** data field is more complex. For example, consider the call scenario where Sally places a call to Tom, and where Sally's **OFL** is [*Init, Auth, RedirectS, Alias, Network*]. As the *setup* signal initializes this call attempt, the signal progresses through Sally's *auth* feature. Next Sally's **RedirectS** feature is executed and the call attempt is redirected, so that it passes through Sally's Work address before the call continues to Tom's address. The redirection prevents the execution of any remaining features in Sally's address, thus Sally's **OFL** must be updated for this call path, such that the **AliasS** feature is removed from the **OFL**. Next, the call attempt is routed by the network into Sally's work address, which triggers the billing feature; and the **OFL** is updated to include the **Delegate** and **Present** features associated with this address, such that **OFL** = [*Init, Auth, RedirectS, Billing, Auth, Delegate, Present, Network*]. This step is also repeated when routing the call to Tom's address zone, so that his features can be added to the call path.

When all of the **NextCallID** data fields are defined, the pair [**NextCallID**, **OutSig**] replaces the pair [**CallID**, **InSig**] in the list of call states, **CList**, readying the model for the next transition step for this call segment. The **NextCallID** data field updates the call identifier for this call segment and the **OutSig** records the input signal that is received as input to the next transition rule.

When a **transrule** outputs a **NewCallID**, the TP model uses a set of default values to set

any undefined data fields: the data fields **Call Name (CN)**, **CS**, and **OFL** are always explicitly defined. The data field **SegStg** defaults to *sAdd* indicating that this new call state is building the call path, **AFL** defaults to an empty list since no features have yet been applied to the new call state, and **CritHis** defaults so that each criterion violation contains a count of zero indicating that no criterion principle violations have yet occurred in this simulation of this call state. The completed **NewCallID** is appended to the modified **CList** (where **NextCallID** replaced **CallID**), finalizing the effect of this **transrule** on the call simulation.

After a **transrule** is executed and **CList** is updated, the TP model continues simulating transitions until all of the possible transitions have been applied to every call segment in **CList**. The call selected for execution in the next transition step is chosen based on the output generated by the last executed **transrule**. If the **OutSig** and **InSig** of the last transition rule are equal, then, to represent “instantaneous signalling”, the generated pair [**NextCallID**, **OutSig**] is selected; otherwise, the TP model selects a pair [**CallID**, **InSig**] from the updated **CList**, where **CallID** represents an executable call segment. In the situation where a **transrule** outputs two distinct signals, the first signal is simulated to completion before the next signal is executed.

Once all of the possible transition steps have been applied, the TP model has completed its evaluation of all calls in **CList**. The TP model then begins backtracking and executes other possible transitions (i.e., different feature data sets, different feature-transition rule executions) until an exhaustive search has been completed for this ordering of feature categories. At this stage, Prolog backtracks even further and generates a different permutation of the feature categories and evaluates the call scenarios for this ordering.

5.6 Principle Assertions

Principles represent necessary (constraint) and desirable (criterion) behaviours of the categories and the overall system. The principle assertions in the TP model test for the existence of a principle violation, which indicates that a feature interaction can occur. As described in Section 5.3, the execution of the TP model tests principle assertions whenever a new feature or interface component is added to the call path. In general, the majority of principle assertion tests will not detect a violation, since feature interactions are the exception rather than the rule.

A **constraint principle** represents required or prohibited behaviour. When a constraint principle assertion is violated, the call has reached an unacceptable state. No optimal feature ordering allows the possibility of a constraint violation, hence the call is considered to be invalid (failed) and the executed subset of this ordering is added to the **ConVioList** and removed from **FreeVioList**.

Criterion principles represent desirable feature and system properties. A criterion violation is not as severe as a constraint violation and is treated like a warning. As such, the existence of a criterion violation does not automatically result in the ordering being considered invalid. After

all possible call scenarios for this ordering have been simulated, the ordering is removed from **FreeVioList**. The process continues until all feature category orderings have been simulated, at which point **FreeVioList** is examined. If there exists at least one ordering in **FreeVioList**, then a violation-free ordering exists and the elements in **FreeVioList** form the partial ordering for this set of categories. However, if **FreeVioList** is empty, then the ordering in **CritVioList** with the smallest criterion count is selected and added to **AllowCritList**; this ordering is the optimal partial ordering for this overly-constrained set of categories.

Example 5.6.1. Abortion Constraint Principle Assertion

*The Abortion Principle Assertion, **abortion**, depicted in Figure 5.6, can only be violated when the call is redirected, via the network, to a blocked address. Therefore, we verify that the call is in the **netLink** stage (line 2). Line 3 extracts the name of the call, and lines 5-21 test whether or not this call has reached the netLink stage incorrectly. **Abortion** checks the subscriber database, **FeatData**, to see if the subscriber of this address zone, **Zone**, subscribes to a blocking feature (**BlockS**, or **BlockT**). If the blocking feature is subscribed to, then **abortion** queries the subscriber database, **FeatData**, to see if the appropriate routing address is on the blocking list. If the call should have been blocked, then the assertion will reach line 23 and **setConPrinValues** will record the information related to this constraint violation.*

Example 5.6.2. Personalization criterion principle Assertion

*The Personalization criterion principle, **personalization**, depicted in Figure 5.7, is tested whenever the call is in the **cAdd** stage and the last feature applied to the call path was an **Alias** feature (lines 3-6). Line 7 extracts the name of the call and lines 10-12 test whether or not the subscriber of this address zone, **Zone**, subscribes to any features that have requested access to the alias field. Line 12 determines if any of the subscriber's features, **FList**, are features that can access **Alias** information, which are identified in the set **AliasL**, and returns these features in **AliasAccess**. The **personalization** assertion then tests the call database, **CallDB**, to see if any of the identified features in **AliasAccess** have already recorded call information. If information has already been recorded, then **datatest** determines whether or not the alias, **Al**, has been properly recorded, which is unlikely unless no mapping exists for this call scenario. If the recorded **Alias** information does not match, then a **personalization** violation has occurred. In this case, **recordCritVio** records information relating to this criterion violation, as shown on line 18.*

5.7 Understanding the Output

This section describes the output generated by the TP model, starting with the lists that identify principle violations and the resulting set of optimal feature orderings. To enhance the usability of the TP model, we implemented an option to generate call trees and signal tables, which represent each call scenario simulated by Prolog. These call trees allow the user to verify that all expected

```

abortion(CList:CallID):-
1  % transrule entry constraints and variables extracted from input
2  CList:CallID:CS = netLink(Reg,Zone,Src,Tgt,Al),
3  CList:CallID:CN = CallName,
4
5  % test based on the region to determine if this call has been blocked
6  (Zone = source ->
7    ( % check the feature data of the current zone to see
8      % if the call to the target or alias should have been blocked
9      checkSubInfo(features(Zone,FList)),
10     member(BlockS,FList),
11     ( checkSubInfo(outblock(Zone,Tgt,block));
12       checkSubInfo(outblock(Zone,Al,block))
13     )
14   );
15  ( % check the feature data for the current zone to see
16    % if the call from the source should have been blocked
17    checkSubInfo(features(Zone,FList)),
18    member(BlockT,FList),
19    ( checkSubInfo(inblock(Zone,Src,block));
20      checkSubInfo(inblock(Zone,Al,block))
21  ) ) ),
22
23  % an abortion violation has been found record information
24  setConPrinValues(CList:CallID,'ABORTION').

```

Figure 5.6. Abortion Constraint Principle Assertion

```

personalization(CList:CallID,OldCName) :-
  1 % transrule entry constraints and
  2 % variables extracted from input
  3 CList:CallID:CS = cAdd(Reg,Zone,Src,Tgt,Al),
  4 % if most recently applied feature is an alias feature
  5 last( CList:CallID:AFL, Feature ),
  6 (Feature= AliasS;Feature= AliasT),
  7 CList:CallID:CN = CallName,
  8
  9 % get the list of all features that can record the alias information
10 checkSubInfo(features(Zone,FList)),
11 subscriberInfo(AliasL,PresentL,LoggingL,EndAccL,AccessL),
12 specialIntersection(FList,AliasL,AliasAccess),
13
14 % has any feature in AliasAccess recorded non-aliased information
15 b_getval(callDB,Data),
16 dataTest(CName,AliasAccess,Data,Al),
17 % a personalization violation has been found record information
18 recordCritVio(CList:CallID,per).

```

Figure 5.7. Personalization criterion principle Assertion

calls have been simulated and to visually confirm the call path along which a principle violation has been found. These call trees can be manually studied, to locate and explain why a violation has occurred or to verify the presence of an expected violation on a given call path.

The main purpose of the TP model is to output the set of acceptable and optimal feature category orderings. To this end, the TP model outputs the following lists:

ConVioList: a list of all feature orderings known to cause a constraint violation.

CritVioList: a list of all feature orderings known to cause a criterion violation.

AllowCritList: a list of all feature orderings known to violate one or more criterion principles that have been acknowledged as optimal orderings, which are called **allowable orderings**. The **AllowCritList** will only contain orderings when the system is unsatisfiable (overly constrained) with respect to all of the principles (i.e., when the **FreeVioList** is empty).

FreeVioList: a list identifying all optimal feature orderings, which do **not** contain any known principle violations.

The evaluation of TP model and its results are detailed in Chapter 6.

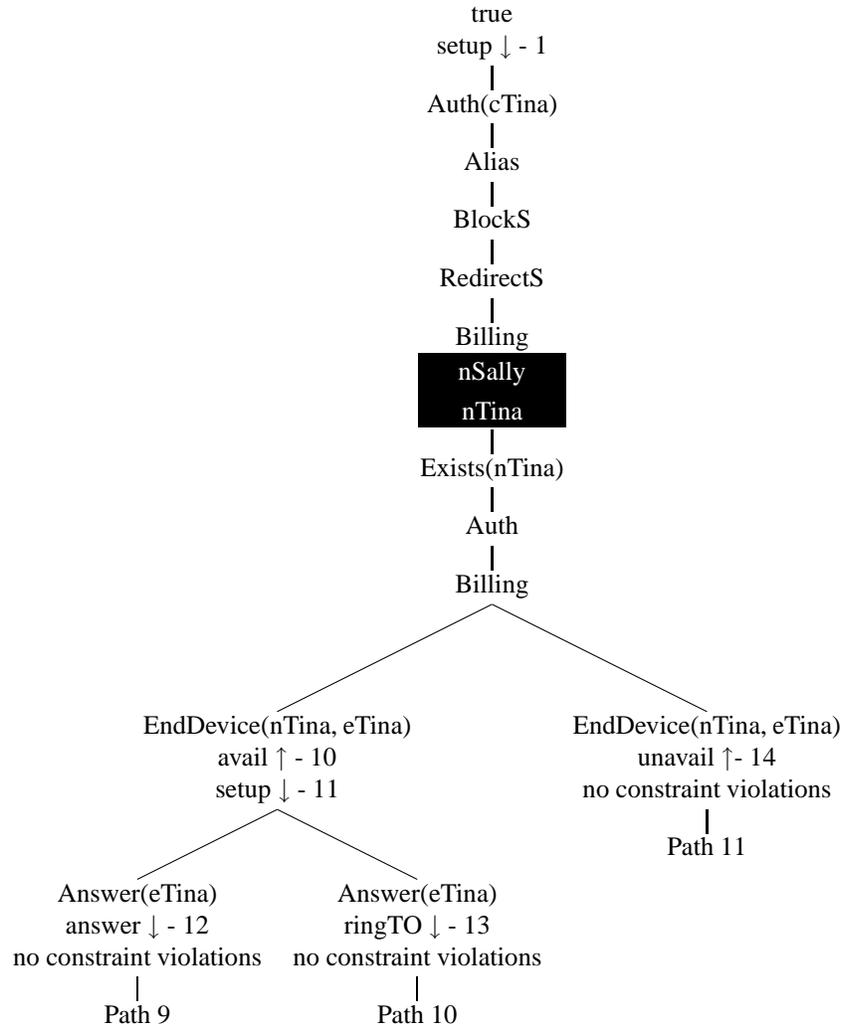
Next, we expand on an optional output that has only briefly been mentioned: **call trees**. The call trees are optional output that the user can analyze to help understand how the call simulations

are executed. A **call tree** is the visual representation of a set of paths, through a call segment, that are generated using the same call scenario. The call-tree nodes are the features and interface components found in the call segment. Each call tree displays multiple types of information, such as the location where new signals are generated, generation new of call segments, how the call segment is linked to other call segments, and the existence of constraint or criterion violations. The call tree is oriented so that signals sent from caller to callee (i.e., in a forward direction) are passed down the call tree, passing between features and interface component, following one of the branches. When signals are sent in reverse, from caller to callee, they are passed upward along the branches. To ease the task of following generated signals through a call tree, the TP model outputs a signal table (see Section 5.1) that records the flow of signals through the features and interface components in each call path. Figure 5.2 is a call tree for the call segment, call1aBbS2, and the three different branches indicate that this call segment explored three unique call paths.

When generating call trees, the TP model uses a call naming scheme that concatenates abbreviations to generate call segment names that correspond to the call paths they represent. Consider the example call1aBbSrS4 from Figure 5.8. The call name is generated by concatenating the name of the call instance (call1) entered into TP model by the user; with abbreviations (aBbSrS) that represents the feature ordering [aliasB, blockS, redirectS]; and with abbreviation (4), which represents the fourth of eight possible sets of feature data. In this example, S at the end of a feature name signifies that the **CRF** represents a source-region category, whereas a B at the end of a feature name signifies that the **CRF** is active in both the source and target regions. Similarly, a T at the end of a feature name signifies that the **CRF** represents a target region category.

When call segments are linked together, the flow of the signals between the call segments must be monitored and controlled. Three different methods for linking together call trees are used to represent the linking of call segments into calls. An example is given below (see Example 5.7.1). The following symbolic notations denote these methods:

⊖: **Standard connection** Consider a standard call scenario where *Sally* calls *Tom*. When the call passes through *Sally*'s **Multiplex** feature, a new call segment is formed, so that we have call segment *A* from *Sally*'s end device to the **Multiplex** feature and call segment *B* containing the rest of the call path. Whenever *Sally* sends a signal in the forward direction, it travels along both segment *A* and *B* in the forward direction as the signal propagates towards *Tom*. The signals also retain their direction when a signal is sent in reverse from *Tom* to *Sally*. Hence, a standard connection is used when signals flow in the same direction between the linked call segments. $\overrightarrow{A} \ominus \overrightarrow{B}$: a forward signal sent along call segment *A* will continue in a forward direction along call segment *B*. Similarly, reverse signals along *B* are continued in reverse along *A*.



Signal Table for **Path 10 of call1aBbSrS4:**

Module	Signal 1	Signal 10	Signal 11	Signal 13
Auth(nSally) - Dialed(cTina)	setup ↓	avail↑		ringTO↑
AliasS(nTina cTina)	setup ↓	avail↑		ringTO↑
BlockS('not blocked')	setup ↓	avail↑		ringTO↑
RedirectS('no redirection')	setup ↓	avail↑		ringTO↑
Billing	setup ↓	avail↑		ringTO↑
Exists(nTina)	setup ↓	avail↑		ringTO↑
Auth(nTina)	setup ↓	avail↑		ringTO↑
Billing	setup ↓	avail↑		ringTO↑
EndDevice(eTina) - avail	setup ↓	avail↑		ringTO↑
Answer(eTina) - ringTO			setup ↓	ringTO↑

Figure 5.8. The call scenario, **callaBbSrS4**, explores the category ordering [Alias, BlockS, RedirectS] together with the following set of feature data: [Alias(nTina,cTina), block(nTina,noblock), redirect(noRedirect)]. The signal table shows the passing of signals among features in the middle call path (Path 10). The *setup* signal is sent through the features, starting from the top of the call tree, until the end device is reached and an *avail* signal is sent in reverse. The *setup* signal is continued and awaits a response from the user. When the user does not answer, a *ringTO* is generated and sent in reverse.

⊙: **Incoming connection** Consider that while *Sally* and *Tom* are talking (as given in the example above), *Charlie* initiates a new call to *Sally*. The setup signal from *Charlie* travels forward along call segment *C* until it reaches *Sally*'s **Multiplex** feature (the same multiplex feature is used in every call involving *Sally*), which is at the end of call segment *A*. In order for the signal from *Charlie* to travel towards *Sally*'s end device, the **Multiplex** feature must reverse the signal as it is sent along segment *A*. Similarly, if the **Multiplex** feature forwards an outgoing signal (i.e., away from *Sally*'s end device), then the signal's direction must be reversed as it enters segment *C*. Hence, an incoming connection is used when an inbound call segment links with an existing call segment, which was also an inbound call. $\overrightarrow{A} \odot \overleftarrow{C}$: a forward signal sent along call segment *A* will be sent in reverse along call segment *C*, and a reverse signal sent along call segment *A* will transition towards *A*'s end device and not reach call segment *C*.

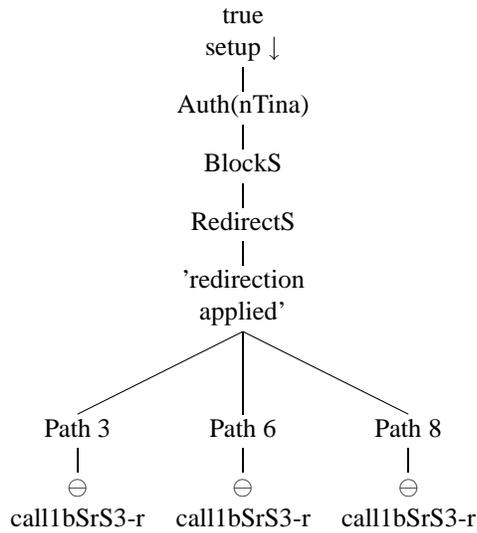
⊕: **Outgoing connection** Consider a call between *Sally* and *Tom*, where *Tom* subscribes to the **Multiplex** feature. This call uses a standard connection to compose call segment *A*, which contains the call path from *Sally*'s end device upto but not including *Tom*'s **Multiplex** feature, and call segment *B*, which contains the remainder of the call path. Suppose that *Tom* uses his **Multiplex** feature to call *Charlie*. *Tom* sends a signal in reverse along segment *B* to trigger the **Multiplex** feature, which initializes the new call attempt by sending a setup signal along a new call segment towards *Charlie*'s end device. The signal leaves the **Multiplex** feature in the reverse direction, but transitions along segment *C* in a forward direction as the new call segment is generated. Hence, an outgoing connection is used when two outbound call segments link together to form a connection. $\overleftarrow{A} \oplus \overrightarrow{C}$: a forward signal sent along call segment *A* will not reach call segment *C* and a reverse signal sent along *A* will transition towards *A*'s end device and not reach call segment *C*.

Example 5.7.1. Trees linked via the Standard Connection

When an existing call segment generates a new call segment, the appropriate link operator is added at the top of the new call segment and at the top (i.e., incoming connection) or bottom (i.e., standard or outgoing connections) of the existing call segment.

In Figure 5.9, the symbol \ominus indicates that the call segment *call1bSrS3* is linked to the call segment *call1bSrS3-r*, and that the two call segments have a standard link connection.

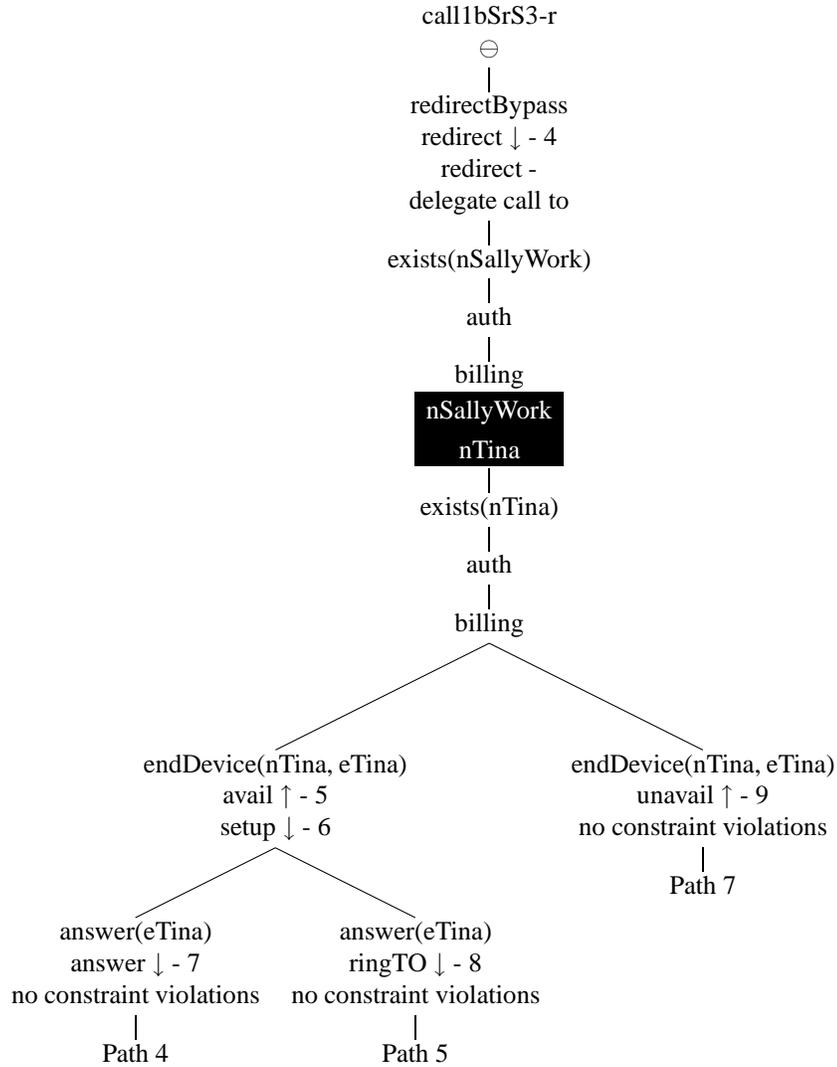
The signal table in Figure 5.9 corresponds to the first call path (Path 3) found in the corresponding call tree. The table shows that each node in the call path first receives a setup signal (starting with the node found at the top of the call tree). When the setup signal reaches **RedirectS**, the feature responds by issuing a redirect signal that forwards the call to a new address. This redirection spawns the creation of a new call segment, and its corresponding call tree, which



Signal Table: **Path 3 (3 + 4) for call1bSrS3**

Module	Signal 1	Signal 5	Signal 8
Auth(nSally) - Dialed(nTina)	setup ↓	avail ↑	answer ↑
BlockS('not blocked')	setup ↓	avail ↑	answer ↑
RedirectS(nSallyWork)	Redirection call1bSrS3-r ↓	avail ↑	answer ↑

Figure 5.9. A call scenario, *call1bSrS3*, where the source address subscribes to BlockS and RedirectS. In this scenario, BlockS does not block the call. However, RedirectS responds to the *setup* signal and redirects the call through another source address, nSallyWork, which generates call path *call1bSrS3 - r* and is shown in Figure 5.10. These two call trees are linked via a standard connection \ominus to form a complete call path. The signal tables are labelled with the corresponding path number from the connected tree to show how the signals travel along the different call trees. The information following the Path 3 label, (3 + 4), indicates that Path 3 along this call tree is joined with Path 4 found on the linked call segment.



Signal Table: **Path 4 (3 + 4) for call1bSrS3-r**

Module	Signal 1	Signal 4	Signal 5	Signal 6	Signal 7
not applied noMoreFeatures and remaining Features, redirect	setup ↓	setup ↓	avail ↑		answer ↑
exists(nSallyWork), authentication - setup		setup ↓	avail ↑		answer ↑
auth(automatic)		setup ↓	avail ↑		answer ↑
billing		setup ↓	avail ↑		answer ↑
exists(nTina)		setup ↓	avail ↑		answer ↑
auth(nTina)		setup ↓	avail ↑		answer ↑
billing		setup ↓	avail ↑		answer ↑
endDevice(eTina) - avail		setup ↓	avail ↑		answer ↑
answer(eTina) - answer				setup ↓	answer ↑

Figure 5.10. The second call tree, *call1bSrS3-r*, resulting from the call scenario, where the source address subscribes to BlockS and RedirectS. See Figure 5.9

is identified in the signal table for easy reference.

The call trees seen thus far have not resulted in any principle violations. In the next two examples, we present how the call trees output by the TP model show the presence of a constraint or criterion principle violation.

Example 5.7.2. A Call Tree with a Constraint Violation

The call tree in Figure 5.11 is a result of a single call with the *CRFs* *Alias* and *BlockS* subscribed to in the source address.

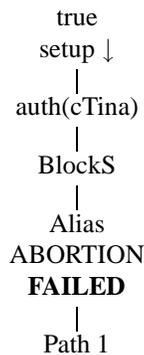


Figure 5.11. A call scenario, where the source address subscribes to *BlockS* features and *Alias* features.

An *abortion* principle violation is found in the call tree shown above. This violation occurs because when the *setup* signal is received by *BlockS*, the dialled code *cTina* has not yet been translated into its corresponding network address, and the code is not found on the blocking list. Next, *Alias* translates the dialled code into the network address *nTina* and continues the call attempt towards the network. However, the dialled number *nTina* was on *BlockS*'s blocking list and the call should have been blocked. Consequently, the *abortion* principle is violated. The TP model terminates further exploration of this call scenario and all other call scenarios with this same ordering.

Example 5.7.3. A Call Tree with a Criterion Violation

Several *personalization* principle violations are found in the call tree shown in Figure 5.12, one in each of the call paths. These violations occur because the *present CRF* executes the *setup* signal before the *Alias CRF* and hence displays to the user the information *nTina*, instead of the personalized information *cTina*. Since this is a *criterion* principle violation, we continue exploring the remaining call paths.

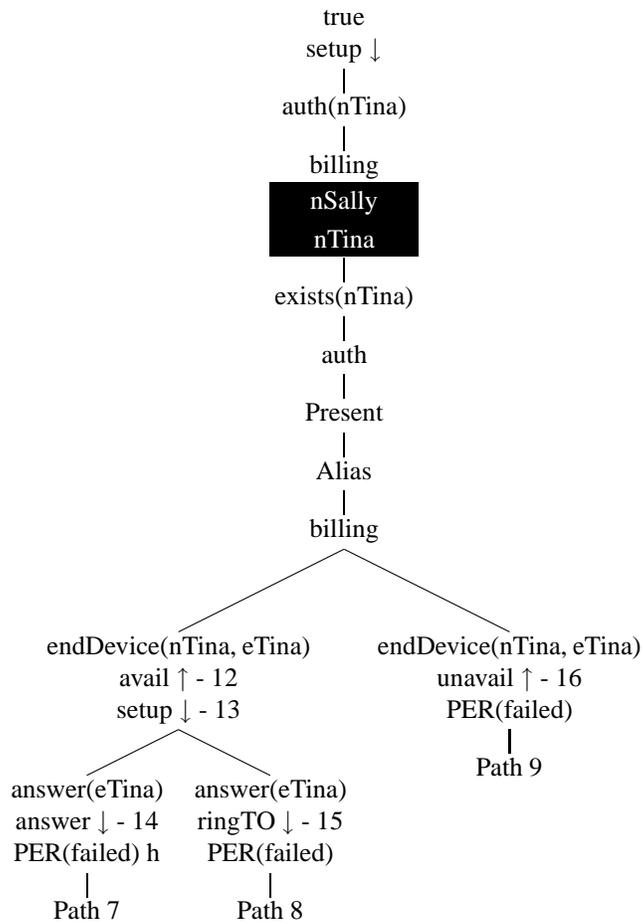


Figure 5.12. A call scenario, where the target address subscribes to Present features and Alias features.

The different paths depicted in a call tree are formed when a **transrule** has multiple execution options. Prolog chooses non-deterministically between the execution options and continues processing the call. When Prolog finishes simulating the call resulting from the first execution option, it backtracks the simulation and chooses a different execution option. In Figure 5.8, we see that the **transrule** executed at the target's end device has two possible execution options: one where the end device is available (*avail*), and the other where the end device is unavailable (*unavail*), resulting in two call paths 9 and 10.

Once the backtracking for the **transrules** are complete, TP model backtracks and selects another feature data set, which generates a new call tree. The end result is that TP model generates a call tree for every possible feature ordering combined with every possible set of feature data (except those orderings where violations are known to exist), where all possible feature data sets for a particular feature ordering are explored before a new feature ordering is selected. The TP model also explores all paths through the feature transition rules that simulate actions such as different callee responses, which together with exploring all feature ordering / feature data value combinations ensures that all possible call paths are simulated. In Section 5.8, we present optimization methods that reduce the actual number of call paths explored.

5.8 Optimizations

The previous sections describe the TP model in terms of searching all possible call paths. In fact, several optimizations were implemented in the TP model to reduce the search problem. Each optimization reduces the number of call paths searched by avoiding exploration of call paths whose sequence of feature categories contains a subsequence known to result in a violation.

5.8.1 Constraint Optimizations

When a feature ordering is known to cause a constraint violation, this ordering and any larger orderings containing this ordering as a subordering are deemed unacceptable. The **constraint optimization method** is designed to prevent the simulation of any future call paths that contain a subordering known to result in a constraint violation. The constraint optimization method tests the input feature ordering against the orderings found in **ConVioList** to determine whether the input feature ordering is unacceptable. When the ordering is deemed unacceptable, the call path is not simulated. The constraint optimization method is implemented immediately after a feature ordering is chosen for use in the call scenario and a feature data set is chosen to generate one of the possible call paths. If the ordering is found to be unacceptable during the exploration of the k^{th} feature data set, then none of the call paths corresponding to the remaining $k+1$ through n feature data sets need to be simulated. The TP model continues the call-path exploration by backtracking and selecting a different feature ordering.

If every ordering of a feature set results in a constraint violation, then the system is over constrained and must be adjusted either by relaxing one of the constraint principles (i.e., demote a constraint principle to a criterion principle) or by setting some of the features in the set to be mutually exclusive, thereby preventing all of the features from being subscribed to simultaneously.

5.8.2 Criterion Optimizations

We also designed and implemented a **criterion optimization method** to reduce the exploration of call paths whose feature ordering contains a subordering known to cause criterion violations (found in **CritVioList**). The criterion optimization method is performed immediately after the constraint optimization. The constraint and criterion optimization methods differ because criterion violations are not critical to proper system execution and may be possible in an optimal ordering.

A criterion violation is found in the optimal ordering only when there is no category ordering that is violation free (either constraint or criterion). For example, if both the category orderings [A, B] and [B, A] cause violations, then no ordering of any feature set containing A and B can satisfy all of the principles. We call such a feature set **unsatisfiable**. When a feature set is unsatisfiable, each of the optimal orderings will suffer from at least one known criterion violation.

As discussed in Section 5.7, when the system is unsatisfiable, an **allowable ordering** is chosen that minimizes the number of criterion violations that can occur and this ordering is placed in **AllowCritList**. During call simulation, an **unsatisfiability test** is performed whenever a new ordering is added to either **CritVioList** or **ConVioList**. If adding this new ordering indicates that the feature set is unsatisfiable, then at least one allowable ordering must be selected, although multiple equivalent orderings can be chosen. Once determined, the **allowable ordering(s)** is removed from the **CritVioList** and added to the **AllowCritList**, so that **AllowCritList**, which is now the set of optimal features, contains ordered feature sets that are known to cause criterion violations, but which have been deemed acceptable for inclusion in the optimal ordering.

The **criterion optimization method** is designed to reduce testing of call paths containing criterion violations, while still gathering enough information to accurately determine the allowable ordering, should the feature set be found unsatisfiable. The method references the global variables **FreeVioList**, **CritVioList**, **ConVioList**, and **AllowCritList** together with the feature ordering *FeatOrd*, which caused the criterion violation, and its corresponding criterion violation count, *CritCount*. The general algorithm for the criterion optimization method is given in Figure 5.13.

During call simulation, when a criterion violation is found, we cannot simply stop execution of a call path, as is done for constraint principles. Instead, we mark the feature ordering as having been violated and continue simulating the call. Our criterion optimization approach works by determining when to mark a feature ordering as being violated, while continuing call simulation,

```

critOptimization(FeatOrd, CritCount):-
1  member(permutation(FeatOrd), FreeVioOrder) ->
2      % there exists an acceptable permutation,
3      % add violation - set is not be unsatisfiable
4      ( addCrit(FeatOrder, CritCount)
5      );
6      % no acceptable permutations of FeatOrd
7      ( member(permutation(FeatOrd), CritVioList) ->
8          % other permutations already in CritVioList
9          ( permSmallCount(FeatOrd, SmallCount, CritVioList),
10             ( CritCount < SmallCount ->
11                 % mark criterion and continue call path exploration
12                 ( markCrit(FeatOrd)
13                 );
14                 % FeatOrd has a larger count than the smallest existing
15                 % add violation and perform unsatisfiability test
16                 ( addCrit(FeatOrd, CritCount)
17                 )
18             ) );
19          % no permutations - mark and continue call path exploration
20          ( markCrit(FeatOrd)
22          ) ).

```

Figure 5.13. Criterion Optimization Method

and determining when to accept a criterion violation as a failed ordering, so that it is unnecessary to explore further call scenarios for this ordering.

The criterion optimization method is applied at two points during call simulation, first immediately after the criterion violation is identified and second when all call scenarios for this feature ordering are complete. The algorithm works by considering the previously simulated feature orderings to determine if this set of features can be unsatisfiable or if this feature ordering could potentially be the allowable ordering should it become necessary to identify such an ordering. If the answer to either of these questions is no, then no further exploration of this feature ordering is required and we reduce the number of call scenarios simulated.

In Figure 5.13, the details of the criterion optimization approach are shown. The function takes as input a feature ordering, *FeatOrder*, that is known to violate *CritCount* criterion violations. The function starts on line 1 by determining whether or not it is possible for this feature set to be unsatisfiable: if there is a permutation of this feature set found in **FreeVioList**, then there is an ordering with no principle violations and hence *FeatOrder* can be added to **CritVioList** (line 4) without any further exploration required. Similarly, on lines 7-10, we test to see if there exists a feature ordering in **CritVioList** that is a permutation of *FeatOrder* and whose criterion violation count is strictly smaller than *CritCount*. If this is the case, then this permutation will be chosen as the allowable ordering, if such an ordering is required. This method is designed such that any ordering that might be chosen as the allowable ordering will always be completely simulated before being added to the **CritVioList**. Consequently, *FeatOrder* does not need to be explored any further and is added to **CritVioList** (line 16). Otherwise, *FeatOrder* is marked as violated and the call simulation is continued.

5.8.3 Pairwise Optimizations

Most feature interactions can be detected by analyzing pairs of features, hence a **pairwise optimization method** was constructed. This method splits the TP model analysis, so that before simulating call paths involving a full set of features, the TP model first generates all possible call paths that contain only pairs of the features. When a pairwise call path identifies a constraint or criterion violation, the information is added to the appropriate violation list while adhering to the above optimization methods. When the TP model completes the execution of the pairwise optimization method, **ConVioList** and **CritVioList** are populated with the majority of the orderings that result in constraint and criterion violations, respectively.

Now when the TP model begins to simulate the call paths comprising full feature sets, the basic violations identified through the pairwise analysis are used by the constraint and criterion optimization methods to greatly reduce the number of full-sized call paths simulated. No full-sized call path that contains a subordering (pair) found in either the **CritVioList** or the **ConVioList** will be simulated. The resulting reduction in the number of call paths explored varies in

effectiveness based partly on the order in which the call paths are generated. For example, if a violation is found when simulating feature data set 1 of m , then there is a savings of $m - 1$, as none of the remaining subsets are simulated. However, if the same violation is not found until executing the m^{th} set, then the savings are nonexistent.

Given a set of n features, the cost of adding the pairwise analysis is quadratic ($n * (n - 1)$), but if a single violation is found during this analysis, then the number of call paths simulated with the full set of n features ($n!$ orderings) will be reduced by half ($n!/2$ orderings explored), since exactly half of the full orderings will contain the subordering that leads to the violation. Each additional violation found will again reduce the number of full orderings that need to be simulated, usually by half. Thus for values of $n > 3$, the cost of the pairwise optimization method (that explores an extra $(n * (n - 1))$ pairwise orderings) is small compared to the savings in avoiding the exploration of related full-sized orderings.

5.9 Chapter Summary

In this chapter, we described the Prolog model designed to simulate a telephony environment in which we could implement our feature categories and principles to automatically generate a prioritized ordering for the categories. We began in Section 5.1 with a general overview of the design of our TP model. In the next section, we introduced the modelling abstractions used within our TP model. Section 5.3 outlines the execution behaviour of our TP model for a given input of feature categories. In Sections 5.4, 5.5 and 5.6, we examine the behaviour of the features, call state, and principles, respectively. The output generated by the TP model is described in Section 5.7. Finally, in Section 5.8 we describe the optimizations used by TP model to reduce the number of computations required.

Chapter 6

Evaluation

This chapter evaluates the effectiveness of our categorization approach and analyzes the results of the TP model presented in Chapter 5, using the categories and principles identified in Chapter 4. In Section 6.1, we present two case studies that generate priority orderings using the TP model. In Section 6.3, we analyze the cost savings of using the categorization approach to reduce the number of *category orderings* explored, while Section 6.4 details the savings with respect to the number of *call scenarios* explored. Finally, in Section 6.5, we summarize the analysis of the TP model and discuss the limitations of our categorization approach.

6.1 A Case Study in Telephony

To evaluate the TP model, we surveyed over 350 home-based features taken from different sources including the feature interaction benchmark [10], the second feature interaction contest [31], and from industry sources, such as Nortel and 3Com [1, 39]. Our goal was to prioritize the feature set subscribed to by traditional users. Consequently, we did not include in our evaluation those features that are involved in data services, such as faxes, or Internet-related features, nor did we evaluate those features used in call-centers. From this set of features, we were able to categorize 268 of the 352 features. The 84 uncategorized features fell into one of three main categories: emergency features; end-device features, which are encoded into the end device, such as on hook (hands-free) dialling [39]; and administrative features, such as the ability to add or remove features from a user's subscription list. The emergency features are not covered in this case study because emergency features are complicated and must be manually evaluated and prioritized to ensure that they adhere to the necessary local and national regulations. The end-device and administrative features are not covered in the case study because they are located in their own special address categories, distinct from the address categories containing the majority

of features.¹ Recall that a call is partitioned into multiple address categories, with the features ordered with respect to other features in the same address zone. This decomposes the prioritization problem into address zones, since each address zone has its own ordering.

Table 6.1 shows how the remaining 268 features are split into 11 of the categories described in Chapter 4. The features listed in the different sources often have overlapping functionalities or descriptions (e.g., each source describes at least one Call Waiting feature), although each feature will have different implementations. In the TP model, we chose not to test all the categories described, since some categories have no effect on their own address zone (e.g., the **Remote X** categories) and so our principles do not apply to these categories. Other categories, such as **Disable**, simply turn off the functionality of another category. The evaluation of these call scenarios is already simulated by our model, based on whether or not the category executes its functionality when it is found in the call path. Moreover, these features have no effect on the progress of the call attempt with the exception of how the effected category behaviour is modified. Table 6.1 also shows the breakdown of features from different sources and the categories into which we classified the features. When a category is partitioned into source and target features, the total number of features is expanded to show the breakdown between the two distinct categories (number of source features, number of target features). Note that the total number of features is higher than 268, because many features have multiple goals, and are thus “implemented” as two or more features, as described in Section 2.3. For example, some features that redirect call attempts can be both **Redirect** and **Delegate** features, while some features that block call attempts may be **Filter** features as well as **Blocking** features.

In the TP model, described in Chapter 5, we noted that **CRFs** are created to represent each of the 11 categories found in Table 6.1. Recall that all of the subscriber’s features must be present in every call because the role played by the subscriber (caller, callee) can change during a call, due to the presence of bound (i.e., **Multiplex**) features. Thus, the potential list of categories to be prioritized includes both the source and target versions of each category where the source and target versions of some categories are clearly distinct. For example, a feature in the **Source Redirect** category redirects outgoing calls through another address zone to add a layer of abstraction to outgoing calls, whereas a feature in the **Target Redirect** category redirects all incoming calls to an alternate address zone for the purpose of locating the subscriber at a different location. In other cases, the source and target version of a category can be represented by a single category and **CRF**. The **Multiplex** category is modelled as a single **CRF**, because this category represents bound features and because each feature in this category is implemented by a single instance of the feature. As such, this unique feature instance must be able to act as both the source and target version of the feature during any given call. However, the **Multiplex** feature

¹Category-based prioritization can be used to prioritize the end-device features within their own address zone. However, this would require identification of categories and principles for this set of features.

Table 6.1. Total Feature Count broken down by Category for different Sources.

Category / Source	Nortel [39]	3Com [1]	Centrex [40]	Centrex Plus [41]	Bellcore [10]	FIW Contest [31]	Total
Original Feature Count	58	59	35	171	17	12	352
Total Feature Count	64	62	41	199	17	12	395
Percent Uncategorized	14%	23%	14%	32%	6%	0%	
Uncategorized	8	14	5	56	1	0	84
Alias	6	3	3	13	1	1	27
Authenticate	4	0	0	2	0	0	6
Billing	7	2	2	2	5	2	20
Block	2 (0,2)	2 (2,0)	5 (2,3)	4 (0,4)	2 (1,1)	1 (0,1)	16
Delegate	5	9	5	36	2	2	59
Filter	2	0	0	2	0	1	5
Multiplex	7	4	4	24	2	3	44
Presentation	6	12	1	26	1	0	46
Redial	7 (2,5)	5 (4,1)	7 (4,3)	2 (1,1)	2 (1,1)	1 (1,0)	24
Redirect	5 (0,5)	6 (2,4)	4 (0,4)	24 (0, 24)	1 (0,1)	1 (0,1)	41
Set Outcome	1	2	1	5	0	0	9

has two choices for the basic **CRF** representations, one for handling a second call initiated by the subscriber (i.e., Three-Way Calling) and the other for handling a second incoming call received by the subscriber (i.e., Call Waiting), the chosen **CRF** will be identified as such in the following case studies. For the other categories, we chose to implement the **Alias** and **Present** categories as single **CRFs**, since it makes sense for the source and target **Alias** features to have access to the same **Alias**-translation list, and for the source and target **Present** features to share common communication functionality. All other categories that have features active in both regions were implemented as two distinct **CRFs**, where the **CRFs** are named «category»**S** or «category»**T**, where «category» represents the name of the category and the last letter indicates whether the feature operates in the source or target region, respectively. This is a design choice on our part, and we acknowledge that other categories, such as the **Blocking** category, could instead be implemented as a single **CRF**, or that the **Alias** and **Present** categories could instead be separated into distinct source and target **CRFs**. Our decision resulted in representing the 11 categories as 14 **CRFs**. We discovered during the creation of the **Authentication** and **Billing** **CRFs** that these feature categories are best implemented in a different call stage than the other categories. For this reason, these two categories are automatically prioritized with respect to the other categories. Therefore, although all 14 **CRFs** are implemented, we need to order only 12 **CRFs** with respect to one another. Thus, we will use 12 as the number of categories in our complexity analysis, since this represents the number of **CRFs** being ordered.

ConVioList:

[target, [BlockT, FilterT]]	[source, [RedirectS, BlockS]]
[target, [BlockT, RedirectT]]	[source, [BlockS, Alias]]
[target, [BlockT, DelegateT]]	[source, [RedirectS, Alias]]

CritVioList:

[target, [DelegateT, SetOutcome], [fail(3)]]	[target, [RedialT, RedirectT], [acc(1)]]
[target, [RedialT, SetOutcome], [log(1)]]	[target, [DelegateT, RedirectT], [acc(1)]]
[target, [FilterT, RedialT], [log(2)]]	[target, [SetOutcome, RedirectT], [acc(1)]]
[target, [FilterT, RedirectT], [acc(5)]]	[target, [Multi, RedirectT], [acc(1)]]
[target, [Present, RedirectT], [acc(3)]]	[target, [DelegateT, RedialT], [log(1)]]
[target, [Alias, RedialT], [per(2)]]	
[target, [Alias, RedirectT], [acc(6)]]	
[target, [Alias, Present], [per(2)]]	[source, [RedirectS, Multi], [acc(1)]]
[target, [BlockT, RedialT], [log(1)]]	[source, [RedirectS, SetOutcome], [acc(2)]]
[target, [BlockT, Present], [pres(1)]]	[source, [RedirectS, DelegateT], [acc(3)]]
[target, [redialS, RedirectT], [acc(3)]]	[source, [RedirectS, RedialT], [acc(1)]]
[target, [RedirectS, RedirectT], [acc(6)]]	[source, [RedirectS, FilterT], [acc(3)]]
[target, [BlockS, RedirectT], [acc(6)]]	[source, [RedirectS, Present], [acc(1)]]
[target, [FilterT, Present], [pres(1)]]	[source, [RedirectS, BlockT], [acc(2)]]
[target, [DelegateT, Present], [pres(1)]]	[source, [RedirectS, redialS], [acc(1)]]
[target, [SetOutcome, Present], [pres(1)]]	[source, [redialS, BlockS], [log(1)]]

AllowCritList: empty**FreeVioList:**

[RedirectT, Multi, Present, SetOutcome, RedialT, DelegateT, FilterT, Alias, BlockT, BlockS, redialS, RedirectS]
 [RedirectT, Multi, Present, SetOutcome, RedialT, DelegateT, FilterT, Alias, BlockS, BlockT, redialS, RedirectS]
 [RedirectT, Multi, Present, SetOutcome, RedialT, DelegateT, FilterT, Alias, BlockS, redialS, BlockT, RedirectS]
 [RedirectT, Multi, Present, SetOutcome, RedialT, DelegateT, FilterT, BlockT, Alias, BlockS, redialS, RedirectS]
 ... (399 acceptable orderings in total)
 [RedirectT, Present, SetOutcome, RedialT, Alias, BlockS, FilterT, DelegateT, redialS, BlockT, Multi, RedirectS]
 [RedirectT, Present, SetOutcome, RedialT, Alias, FilterT, BlockS, redialS, DelegateT, BlockT, Multi, RedirectS]
 [RedirectT, Present, SetOutcome, RedialT, Alias, BlockS, FilterT, redialS, DelegateT, BlockT, Multi, RedirectS]
 [RedirectT, Present, SetOutcome, RedialT, Alias, BlockS, redialS, FilterT, DelegateT, BlockT, Multi, RedirectS]

Figure 6.1. The results reported for an analysis of a single call-simulation. In this test the Multiplex CRF, *Multi*, is based on the generic behaviour of a Call Waiting feature.

The results of running TP model with all 14 **CRFs**, to determine the ordering of the 12 **CRFs** in the subscriber's address zone, and with all 9 of our principle assertions on a single call scenario, are shown in Figure 6.2. TP model is run with all optimizations (Constraint and Criterion) turned on and with the following simulation pattern:

1. Test all category pairs in target region for a single call simulation.
2. Test all category pairs in source region for a single call simulation.
3. Test all full-sized category orderings in target region for a single call simulation.
4. Test all full-sized category orderings in source region for a single call simulation.

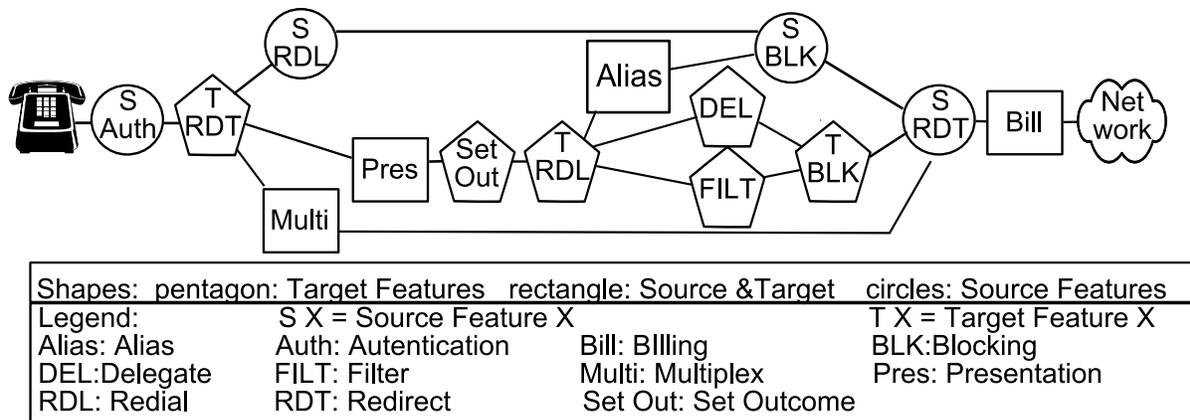


Figure 6.2. This figure shows the combined partial ordering determined using the results output by our Telephony Prolog Model.

where the principle violation result from each step is used to eliminate orderings that must be searched in subsequent steps. For example, a violation found in the target region during pairwise analysis will eliminate all full-sized orderings that contain this ordered pair during the testing of all full-sized categories in the target region. In this case study, the Multiplex CRF simulates Call Waiting in all call scenarios.

When we merge the acceptable orderings given in Figure 6.1, we derive the partial ordering shown in Figure 6.2. This partial ordering reflects most of the violations identified during manual analysis, whose results were reported in Chapter 4. The one feature that is not ordered according to the results of the manual analysis is the Multiplex feature. In this case study, we implemented a single-call simulation, where the subscribed set of features is tested first in the source region of the call and later in the target region of the call. Consequently, the Multiplex CRF behaviour is not fully tested, as only one call connection is present. To fully test this feature, it is necessary to test call scenarios in a multiple-call environment. We suggest analyzing a single-call simulation and using the results reported by this analysis, such as the principle violation results found in CritVioList and ConVioList, as input to a multiple-connection call simulation to limit the exploration required. The result of adding a multiple-call scenario is reported in Section 6.2.

Although this case study compares only a single set of feature categories, we believe that this is a representative example of the problem space, and hence the approach should be applicable to other feature sets.

6.2 An Advanced Case Study in Telephony

We continue our evaluation of the case study performed in Section 6.1 by adding a test that evaluates a multiple call scenario. In this second case study, we incorporate a test that evaluates

every non-Multiplex feature together with a Multiplex feature, where each ordered pair is implemented in a two-call scenario. Specifically, the feature pair is assigned to the source region of the first call, and then once the first call is established, the subscriber of the source region using a *feature-specific* signal to generate a new call attempt via the Multiplex feature (i.e., a Three-way Calling feature is simulated).

ConVioList:

[target, [BlockT, FilterT]]	[source, [BlockS, Alias]]
[target, [BlockT, RedirectT]]	[source, [RedirectS, Alias]]
[target, [BlockT, DelegateT]]	[source, [Alias, Multi]]
[source, [RedirectS, BlockS]]	

CritVioList:

[target, [DelegateT, SetOutcome], [fail(3)]]	[target, [RedialT, RedirectT], [acc(1)]]
[target, [RedialT, SetOutcome], [log(1)]]	[target, [DelegateT, RedirectT], [acc(1)]]
[target, [FilterT, RedialT], [log(2)]]	[target, [SetOutcome, RedirectT], [acc(1)]]
[target, [FilterT, RedirectT], [acc(5)]]	[target, [Multi, RedirectT], [acc(1)]]
[target, [Present, RedirectT], [acc(3)]]	[target, [DelegateT, RedialT], [log(1)]]
[target, [Alias, RedialT], [per(2)]]	
[target, [Alias, RedirectT], [acc(6)]]	[source, [redialS, BlockS], [log(1)]]
[target, [Alias, Present], [per(2)]]	[source, [RedirectS, Multi], [acc(1)]]
[target, [BlockT, RedialT], [log(1)]]	[source, [RedirectS, SetOutcome], [acc(2)]]
[target, [BlockT, Present], [pres(1)]]	[source, [RedirectS, DelegateT], [acc(3)]]
[target, [redialS, RedirectT], [acc(3)]]	[source, [RedirectS, RedialT], [acc(1)]]
[target, [RedirectS, RedirectT], [acc(6)]]	[source, [RedirectS, FilterT], [acc(3)]]
[target, [BlockS, RedirectT], [acc(6)]]	[source, [RedirectS, Present], [acc(1)]]
[target, [FilterT, Present], [pres(1)]]	[source, [RedirectS, BlockT], [acc(2)]]
[target, [DelegateT, Present], [pres(1)]]	[source, [RedirectS, redialS], [acc(1)]]
[target, [SetOutcome, Present], [pres(1)]]	[target, [Present, Multi], [pres(1)]]

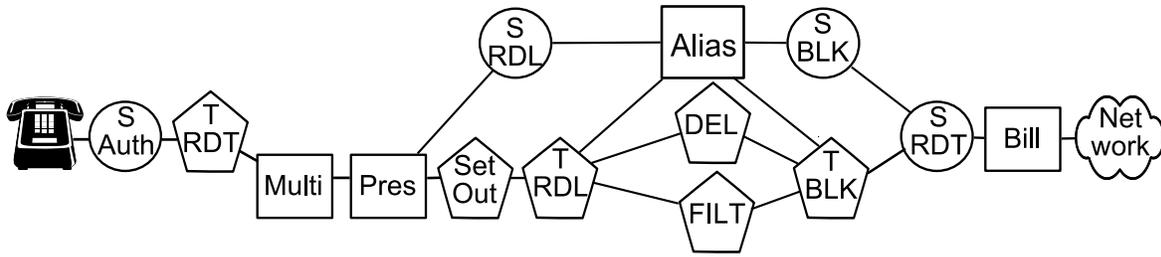
AllowCritList: empty

FreeVioList:

[RedirectT, Multi, Present, SetOutcome, RedialT, Alias, FilterT, DelegateT, BlockS, RedialS, BlockT, RedirectS]
 [RedirectT, Multi, Present, SetOutcome, RedialT, Alias, BlockS, FilterT, DelegateT, BlockT, RedialS, RedirectS]
 [RedirectT, Multi, Present, SetOutcome, RedialT, Alias, FilterT, BlockS, DelegateT, BlockT, RedialS, RedirectS]
 ... (37 acceptable orderings in total)
 [RedirectT, Multi, Present, SetOutcome, RedialT, DelegateT, FilterT, Alias, BlockS, RedialS, BlockT, RedirectS]
 [RedirectT, Multi, Present, SetOutcome, RedialT, DelegateT, FilterT, Alias, BlockS, BlockT, RedialS, RedirectS]
 [RedirectT, Multi, Present, SetOutcome, RedialT, DelegateT, FilterT, Alias, BlockT, BlockS, RedialS, RedirectS]

Figure 6.3. The results reported for an analysis of a multiple-call simulation. In this test the Multiplex CRF, *Multi*, is based on the generic behaviour of a Call Waiting feature for single call scenarios and in the target region of multiple-call simulation, while the generic behaviour of Three-way Calling is used in the source region for multiple-call simulations.

The results of running TP model in this second case study with all 14 CRFs, to determine the ordering of the 12 CRFs in the subscriber’s address zone, and with all 9 of our principle assertions on a single call scenario, are shown in Figure 6.4. TP model is run with all optimizations (Constraint and Criterion) turned on and with the following simulation pattern:



Shapes:	pentagon: Target Features	rectangle: Source & Target	circles: Source Features
Legend:	S X = Source Feature X	T X = Target Feature X	
Alias:	Auth: Autentication	Bill: Billing	BLK:Blocking
DEL:Delegate	FILT: Filter	Multi: Multiplex	Pres: Presentation
RDL: Redial	RDT: Redirect	Set Out: Set Outcome	

Figure 6.4. This figure shows the combined partial ordering determined using the results output by our Telephony Prolog Model.

1. Test all category pairs in target region for a single call simulation.
2. Test all category pairs in source region for a single call simulation.
3. Test all non-Multiplex - Multiplex pairs in source region for a two-call simulation, using Three-Way Calling.
4. Test all non-Multiplex - Multiplex pairs in target region for a two-call simulation, using Call Waiting.
5. Test all full-sized category orderings in target region for a single call simulation.
6. Test all full-sized category orderings in source region for a single call simulation.
7. Test all full-sized category orderings in source region for a two-call simulation, using Three-Way Calling.
8. Test all full-sized category orderings in target region for a two-call simulation, using Call Waiting.

where the principle violation results from each step are used to reduce the search space in the subsequent steps. In this case study, the Multiplex CRF simulates Call Waiting in all call scenarios except in the two-call simulation, where Call Waiting is used when the Multiplex feature is found in the target region and Three-Way Calling is used in the source region.²

Figure 6.3 shows the results output by TP model. The highlighted orderings identify new restrictions found during the multiple-call simulation. When we merge the acceptable orderings

²This choice shows a large portion of the flexibility of the features in the Multiplex category.

given in Figure 6.3, we derive the partial ordering shown in Figure 6.4. This partial ordering adds the newly identified violation that places **Multiplex** on the subscriber-side of **Alias**.

6.3 Cost Analysis: the Number of Feature Orderings

Our hypothesis is that, by categorizing the features into categories, we can reduce the cost of determining priority orderings for a set of features. We calculated the cost of our category-based prioritization technique and compared it against the cost of traditional prioritization methods, where the number of feature orderings evaluated is used to measure the cost. We use the results of the first case study in our evaluation to demonstrate the effectiveness of our approach. If we were to use the second case study in our evaluation, then the results of the evaluation would show an even greater reduction in cost, despite the extra testing required to generate the multiple-call simulations, because the number of full-sized ordering explored is reduced from 399 to 39 distinct orderings.

The traditional approach to prioritizing features is the brute force approach that involves the generation of $f!$ features orderings, where f is the number of features to be prioritized. Therefore, the traditional cost to prioritize 268 features is $268!$. This number is too large to be used effectively in our discussions; consequently, we use the size of the Nortel feature set [39] (second column of Table 6.1) for comparison purposes, so our "full set of" features is of size 58 and the traditional cost is $58! \approx 2.3 * 10^{78}$. Each ordering is tested in both the source and target region of a call to explore the ordering's effect in different regions.

Using our categorization approach, the cost to order the features is the cost to order the n **CRFs** ($n!$), plus the cost to categorize the features (f), plus the cost to order each of the category's feature sets: $\sum_{i=1}^n C_i!$, where C_i is the number of features in the i^{th} category. There is also the additional cost of identifying the categories and principles, but this cost is not assessed in this section. For our example, using 12 categories and 58 features, the number of orderings generated and analyzed is:

$$\begin{aligned}
&= \text{CRFs orderings} + \text{cost to categorize features} + \text{feature orderings} \\
&= 12! + 58 + \sum_{i=1}^{12} C_i! \\
&= (\approx 4.79 * 10^8) + \approx 58 + (\approx 1.19 * 10^4) \\
&= \approx 4.79 * 10^8
\end{aligned} \tag{6.1}$$

This is a savings of over $(100 - \frac{1}{10^{68}})\%$: a significant reduction in cost.

f	# of features
n	# of categories
v	# of violations found
C_i	# of features in the i^{th} category
S_i	# of subscriber-data values for the i^{th} category
$f!$	# of features orderings
$n!$	# of categories orderings
$\sum_{i=1}^n C_i!$	total # of feature orderings for all n categories
$\prod_{i=1}^n S_i!$	total # of subscriber-data values combinations for all n categories

Figure 6.5. This figure gives a quick reference to the variables and formulae used in this chapter.

The cost to generate orderings is factorial, therefore the cost of ordering the categories plus the cost of ordering feature sets for each category is approximately equal to $k!$, where k is equal to the largest number in the set $[C_1, C_2, \dots, C_n, n]$. Comparing $k!$ to $f!$ gives us

$$\frac{k!}{f!} = \frac{1}{\prod_{i=k+1}^f i} \quad (6.2)$$

Consequently, as long as k is significantly smaller than f (i.e., there is a fair distribution of features among the categories, and the number of categories is measurably smaller than the number of features), then our categorization approach will always result in a significant decrease in the amount of work needed to order a set of features. However, the number of orderings explored remains high at $4.79 * 10^8$.

6.3.1 Pairwise Optimization

In this section, we present the further savings that result when a pairwise optimization is performed to further reduce the number of orderings that need to be explored. The cost of the pairwise analysis is $\frac{n!}{(n-2)!}$, where n is the number of **CRFs**. If a single constraint or criteria violation is found during pairwise analysis, the number of full orderings explored is reduced by half, since no category orderings that contains this violation will be tested, and exactly half of the full feature orderings will contain such a pair. As a result of this optimization, if a single violation is found in our case studies, then the number of feature orderings evaluated becomes

$$\frac{n!}{(n-2)!} + \frac{n!}{2} + \sum_{i=1}^n C_i!. \quad (6.3)$$

For our example, one pairwise violation reduces the number of explored orderings to $\approx 2.395 * 10^8$, a savings of $\approx 50\%$ over the basic categorization approach.

Every new violation found during pairwise analysis reduces again by half the number of full feature orderings that need to be examined, as long as this ordering is not already implied by previously identified pairs. As seen in Figure 6.6, the addition of some features have no effect on the number of orderings explored since the newly identified ordering was already implied. However, if two violations share a common **CRF**, then the savings is even higher, or nonexistent. For example, if we have know (A, B) and add (B, C) , then the savings are much higher since no orderings with (A, C) will considered. However, if we then add the ordering (A, C) , no new savings will be seen as this was already implied by the previous orderings. We use the formula $GO(n, v)$ to represent the number of full category orderings generated given n **CRF**s and v violations. If pairwise analysis is used and $GO(n, v)$ full-sized **CRF** orderings that avoid pairwise violations are generated, instead of the original $n!$ orderings, then the total number of orderings evaluated becomes

$$\leq \frac{n!}{(n-2)!} + GO(n, v) + \sum_{i=1}^n C_i!. \quad (6.4)$$

For our first case study, we generated 400 full-sized **CRF**s instead of the original $4.79 * 10^8$, orderings at a cost of $\approx 8 * 10^{-5}\%$ of the original cost. Thus, in the first case study, combining categorization with all of the optimizations reduces the total number of orderings to $\approx 1.24 * 10^4$ or $\approx .003\%$ of the original cost. Table 6.6 shows how each new violation found during pairwise analysis reduces the number of **CRF** orderings explored. Notice that some new violations do not affect the number of orderings generated (e.g., violations # 24, 25, and 26 all evaluate the same number of orderings). This is because the new constraint imposed by the new violation is implied by existing constraints (e.g., adding the new restriction $[a,c]$ to the existing set of restrictions $[[a,b], [b,c]]$).

6.4 Cost Analysis: the Number of Call Scenarios

In the above section, we were able to show a reduction to $\approx .003\%$ of the original number of orderings that need to be explored when categorization is combined with the pairwise optimization technique. In this section, we compute the number of call scenarios that are explored with respect to each feature ordering based on the number of data values per feature in the call, and the number of scenarios avoided when criteria violations are detected. As previously described,

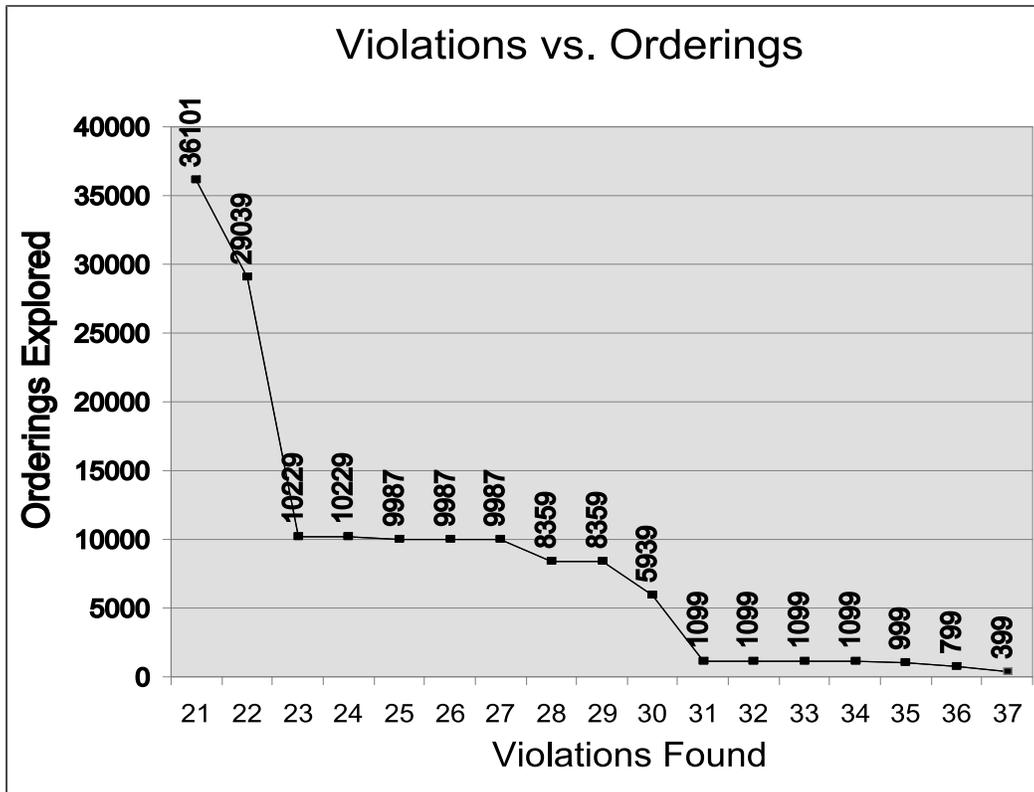


Figure 6.6. This data is generated using our first case study, where violations found during pairwise analysis are used to determine the number of category representative feature (CRF) orderings generated.

for each feature ordering, all possible variations of feature data (e.g., whether the dialled number is on a Call-Screening list, or whether a feature’s data indicates that the call should delegate on *ringTO* or *setup*) are explored as a separate call scenario.

Table 6.2 shows the number of different subscriber-data values that are associated with each **CRF**. Combining these possibilities, we find that there are $\prod_{i=1}^n S_i$ call scenarios for each feature ordering, where S_i is the number of possible subscriber-data values related to the i^{th} category. Both the constraint and criterion optimization methods are used to reduce the number of call scenarios explored; however, this reduction is hard to measure because the cost savings are directly related to when during the search the call scenario containing the criterion or constraint violation is detected.

Consider the situation where a constraint violation is identified: when this violation is detected, no more call scenarios corresponding to the currently simulated category ordering will be evaluated. Consider a situation where eight **CRFs** are implemented, and each has two possible subscriber-data values (e.g., number blocked and number not blocked for a **Blocking** feature).

CRF	# of Data Values
BlockS	2
BlockT	2
Alias	2
Present	1
Filter	3
RedirectT	4
RedirectS	2
RedialS	1
RedialT	1
DelegateT	3
SetOutcome	2
Multiplex	1

Table 6.2. This table shows the number of choices for feature data for each category representative feature.

There are $2^8 = 256$ call scenarios related to this one ordering. If we find the constraint violation during the first call scenario, then we immediately reject this ordering, without exploring the remaining 255 call scenarios, for a savings of $\approx 99.6\%$. However, if the constraint violation is not found until the i^{th} call scenario is explored, then the savings are:

$$\frac{\prod_{i=1}^n S_i - j}{\prod_{i=1}^n S_i} = 1 - \frac{j}{\prod_{i=1}^n S_i}. \quad (6.5)$$

The same is generally true for criterion violations, except that analysis of the remaining call scenarios will occasionally continue, since enforcing all criterion principles may over-constrain the system. Consequently, we continue a complete evaluation of an ordering that violates criteria principles, unless a permutation of the ordering has already been added to **FreeVioList** or until the criterion count for this ordering is higher than the criterion count for a permutation of this ordering.

In our first case study, this optimization affects only the number of pairwise call scenarios, since only pairwise orderings resulted in principle violations. In our first case study, the total number of simulated pairwise call scenarios was 367. If the criterion optimization had not been applied, 496 call scenarios would have been generated, thus using this optimization resulted in a savings of $\approx 26\%$. These are a small savings, but had pairwise optimization not been applied, then the savings would have been significantly higher. As well, this optimization results in significant savings whenever violations are found in larger orderings.

In our first case study, we did not reveal any new violations during the full-ordering analysis. However, this step is necessary, because the addition of a new feature or category of features can easily modify the behaviour of the system, such that multi-feature interactions will occur. As well, this step verifies that the system should execute without the presence of any principle violations.

6.5 Analysis Results

The final results from the implementation of our first case study in Prolog show that 399 full-sized category orderings were explored and the partial ordering in Figure 6.2 was generated. Given these category orderings and assuming traditional costs to order the features in each category, we have reduced the number of orderings explored from $\approx 2.3 * 10^{78}$ to $\approx 1.19 * 10^4$ (evaluation cost for the 399 full-sized category orderings plus the intra-category ordering cost), or $\approx 10^{-56}\%$ of the original cost. To prioritize all of the features found in the telephony environment, we could use the categorization approach on each type of address (e.g., device, system, router), and combine the results based on the inherent priorities assigned to each address category.

We acknowledge that there are some limitations to our categorization approach, but we submit that these limitations are not significant with respect to the savings generated. The first limitation is that not all features (e.g., 911 features) can be categorized or that some categories will contain a small number of features. These small categories can either be accepted and modelled in the TP model or treated as exceptions and manually analyzed with respect to the generated category orderings. The decision between these two choices should reflect the complexity of the feature and the total number of categories to the order. For example, 911 features are complex and have many principles for proper execution; hence these features should be manually prioritized. Alternatively, if we have a small number of categories and a few simple, uncategorized features (e.g., override feature), then it may be acceptable to model these in the TP model.

The second limitation is that the generated partial ordering may still need to be analyzed by a human expert. The partial ordering generated is guaranteed to adhere only to the principles used in the analysis, but the human expert may wish to check for more specialized interactions that are not represented by one of these principles. For example, if we wish to give **BlockT** the ability to block incoming calls based on aliases, then the human expert can assert this restriction. To search for these specialized interactions the human expert need only examine the set of optimal partial orderings, which greatly reduces the amount of work required. As well, a human expert should always validate the **AllowCritList** list to determine if the correct ordering for the over-constrained feature set was chosen. (We found no such ordering during our case studies.)

The last limitation we consider is the run time for TP model. Although our categorization approach greatly reduces the number of orderings explored, the results are still factorial with

CRFs	New CRF	# of Inferences	Run Time (seconds)	Heap Size (Bytes)	Con	Crit	Opt
5	Original Set	841,939	1.6	888,412	2	3	6
6	RedirectT	3,136,352	4.89	977,540	3	7	6
7	RedirectS	6,232,809	9.43	1,076,048	5	11	6
8	RedialS	10,877,028	17.70	1,223,036	5	15	10
9	RedialT	22,414,800	37.54	1,460,316	5	20	20
10	DelegateT	250,707,440	409.58	8,653,536	6	24	80
11	SetOutcome	231,676,899	373.19	8,179,104	6	29	40
12	Multiplex	20,009,084,662	3315.49	82,994,748	6	32	400

Table 6.3. This table shows how adding a new category representative feature, where the Original Set is [BlockS, BlockT, Alias, Present, Filter], changes the run time results.

respect to the size of the number of categories. This means that despite our many optimizations, there is a limit to the number of categories that can be automatically prioritized. The runtime analysis of the TP model is shown in Table 6.3. These results were generated using swi-Prolog (version 5.6.29) on a Windows XP platform using an Athlon XP X2 4400+ processor and 3G of RAM. Swi-Prolog's internal global and local stack settings were set at their maximum setting of 128 M. It is our belief that 15 to 20 categories can be easily ordered in a moderately constrained domain (i.e., 5-10 principles): we ran into stack overflow errors in Prolog when ordering 18 categories (all applicable categories where split into source and target categories for this simulation). However, not all principles were implemented in this case analysis, and the implementation of these principles which would likely have avoided the overflow issue. In more constrained system, where a large number of pairwise violations are found, a even larger number of categories can be ordered since each pairwise violation dramatically reduces the number of full sized feature category orderings that need to be evaluated. We believe that the other address categories, such as end-device address zones and network address zones, which are found in the telephony environment can also benefit from this approach. Each of these address zones would need to formulate its own set of categories that correspond to the different types of features in the address zone (e.g., screen display features, billing features, routing features).

The results of our analysis show that our categorization approach is successfully able to reduce the cost of prioritizing features, and that this prioritization can be automatically generated. This saves both time and money, freeing up time for human experts to do other tasks. The extra effort required to setup the categorization used in our approach is minimal in comparison to the increase in savings generated.

6.6 Model Reusability

The TP model is designed to test our approach with respect to the DFC-based telephony model described in Chapter 2. However, core components of this model could be reused to evaluate alternate “categorizable” domains. The conditions under which categorization is appropriate are 1) the features are independent of one another (i.e., modular feature development), and 2) communication between features can be modelled as an exchange of signals, with the option to share information via a database.

The new domain will require the creation of new feature transitions (i.e., a **CRFs** for each category in the new domain) and new principle assertions (i.e., an assertion representing each principle of proper system behaviour within the domain). If the features within the domain are applied in a serial ordering, then our TP model’s signalling/communication code used to propagate messages between features may be reusable. However, if the features are applied using a non-serialized communication model, then the entire underlying domain code may need to be redesigned. In either case, some of the underlying behaviour of the basic domain would likely need to be modified. For example, the new domain may require an alternate decomposition of the simulation into stages (i.e., the “call” stages may need to be modified to reflect the stages of a computation in the new domain).

The constraint and criteria optimizations should be usable in the new domain. The usefulness of the pair-wise optimization will need to be determined based on the types of feature interactions found within the new domain. However, with minor modification, this code can be used to test subsets of any size.

Another issue that affects reusability is that in our model, each feature module active and remains in the call path for the entire duration of the call which reflects how feature modules are treated in the DFC architecture. In other domains, features may be transient and remain active for only a portion of a simulation in the domain. However, we can simulate in our model the transient behaviour of a feature by designing the feature to behave transparently after it has completed its functionality.

Chapter 7

Conclusions and Future Work

Feature interactions pose a large problem in feature-rich domains, so much so that an entire research community (i.e., centered around the International Conference on Feature Interactions) has evolved to address this issue [6, 16, 17, 30, 9, 43]. In this thesis, we explore methods to reduce the cost of prioritizing a set of features for use in priority-based techniques (see Section 3.5.4) for resolving feature interactions.

Our categorization approach reduces this cost by using abstraction to divide the system's features into categories based on their main goal or functionality (i.e., block unwanted calls, present call information). The development of the feature categories follows a simple set of guidelines. To begin, each category should represent a single core functionality; these categories can then be further decomposed if a category has multiple goals. When determining the final set of categories, the designer should verify that the categories maintain a high level of cohesion among features within a category and a low level of cohesion between features in different categories. These design rules increase the effectiveness of the approach by increasing the ability to find the unexpected interactions that occur between seemingly unrelated features that are found in different categories.

These categories are then ordered with respect to one another using a set of principles that reflect the desired behaviour of the system and its features (i.e., no blocked calls will reach an end device). The principles provide the justification that the approach uses to effectively and efficiently generate feature categories orderings, so that the resulting orderings satisfies the requirements identified by the system designer and the end users.

The categorization of features is a critical step that must be carefully considered as it affects the success of our categorization approach, since incorrectly categorizing the features can result in high cohesion between features in different categories and/or low cohesion between features in the same category. When this occurs, performance using the approach will be poor, as the identified feature interactions will often generate an overly constrained set of categories that may require a human expert to re-evaluate the results and modify the output priority ordering. Once

the categories are selected, the principles for proper system behaviour are created. The principles are properties that hold regardless of which features are active in the system. In our experience, it is easier to identify the principles after the categories have been determined, since the categories give insight into what the system and features are trying to accomplish. In Chapter 4, we present a set of feature categories and principles for the telephony domain, which represent the set of features for residential clients.

By decomposing the prioritization problem into two phases, first ordering a set of categories and then ordering the intra-category features, we reduce the problem significantly, to the point where we can automatically generate the priorities for the feature categories. In Chapter 5, the results of programming our telephony domain, categories, and principles in Prolog are described. Using this Prolog model, two case studies are evaluated and the results are presented in Chapter 6. The case studies show that applying our categorization approach in our DFC-based telephony domain significantly reduces the cost to order a set of features. The chapter also discusses how the implemented optimizations further reduce the cost of generating optimal orderings.

The partial-ordering generated by the case studies is nearly equivalent to the partial-ordering identified by our manual analysis. In fact, when generating the results, we discovered the need for two new principles. These principles were then added to our Prolog model and the results of our manual analysis were updated to reflect these two new principles. The identification of these two new principles shows that automating the feature-ordering problem can help identify unknown principles and thus identify new restrictions on the feature orderings and increase the accuracy of identified feature orderings. Furthermore, our evaluation shows that as long as the number of categories is significantly smaller than the number of features to be ordered, then the cost reduction to determine priority orderings is significant. Our case study, implemented using an optimized Prolog model, shows that the cost is reduced to less than $\frac{1}{10^{55}}$ % of the traditional cost of testing all possible feature orderings. Given this significant reduction in cost and the ability of our model to accurately reproduce the manually identified partial-orderings, we can confidently argue that our categorization approach was successful.

The partial orderings, generated using the approach described in this thesis, were created for a DFC-based telephony environment. However, this does not limit the effectiveness of these orderings for use in resolution strategies designed for non-serialized telephony domains. The partial ordering results can be used in precedence- or priority-based resolution techniques without knowing how the underlying telephony architecture behaves. For example, in an alternate telephony architecture, all features triggered by the current state of the system could attempt to execute in parallel. A resolution strategy is used to determine when the set of executing features would interact and to resolve such interactions. An example of such a strategy is the priority-based resolution strategy developed by Tsang and Magill in [48]. In their approach, when two or more of the triggered features conflict, the feature with the smallest number of conflicts is chosen

for execution; in the case of a tie, the highest priority feature is chosen for execution. Although these partial orderings are generated using a serialized architecture, the priorities indicated by the partial orderings will correctly resolve these interactions (assuming the conflict type reflects one of our system principles) in this alternate architecture where features are executed in parallel, since the resolution strategy effectively serializes the execution of the conflicting features. For example, if feature *A* and feature *B* conflict in the parallel execution and *A* has priority over *B* based on the partial ordering, then when *A* is chosen for execution over *B*, the result is an optimal ordering, since either *A* will prevent *B* from executing by modifying the call state, or *A* will execute and leave the call state such that *B* is triggered again. In either case, the proper execution of the features is applied. Tsang and Magill's approach, along with other related work and techniques that use precedence or priorities to resolve interactions, are discussed in detail in Sections 3.6 and 3.5.4, respectively.

Below, we introspectively consider the use of the generated partial orderings within several different telephony architectures:

CPL: The Call Processing Language, CPL, is an architectural framework designed to simplify and standardize a method for creating features and services for Internet telephony [33]. Using CPL, the end user creates a script describing the features or service that the user wishes to be activated during call attempts. These scripts do not support modular feature development. Rather the scripts describe all the functionality for a specific end device or server. Each CPL script can represent multiple features and the combined behaviour of these features. CPL scripts give each end user the ability to create their own uniquely defined service behaviour. Its implementation as a single script removes ambiguity and resolves automatically feature interactions as only one action can be planned for each triggering signal / data combination. The CPL script is tested for unreachable code. The script is then uploaded to a server where it is implemented. Multiple servers may be reached during a call attempt, and each server, which acts as a CPL server, chooses the appropriate CPL script for execution based on the source and target address for the call attempt. This script is then executed and the call proceeds as the script specifies (e.g., is terminated, forwarded, redirected). When the call attempt is continued, the call may pass through more servers, where the end result is that each CPL script is executed based on the order in which the servers are added to the call path (i.e., follows address translation concepts and principles).

Due to the scripts being analogous to Ideal Address Translation (IAT) address zones (see Section 2.2) and due to the fact that the responses to signals are explicitly determined by the script, the categorization approach is not useful as a design rule for this architecture. However, the priority orderings identified using the categorization approach can provide script writers with advice on how to design scripts that combine features in a manner

that adheres to the principles for proper system behaviour. The use of IAT principles and conventions [27] can be used to resolve interactions that occur between features found in different CPL servers.

AIN: The Advanced Intelligent Network (AIN) architecture was developed by Bell Communications Research and is an industry standard in North American. The architecture separates service logic from the plain old telephone switching. Removing the service logic from the switching software allows new features and services to be added to the system without costly redesigns to the switching network. (AIN is more fully described in Section 3.4.2.) AIN supports modular feature development and the addition of features designed by multiple vendors. Each AIN feature can be represented by a communicating finite state machine, and is atomic, in that it cannot be interpreted by another AIN feature once execution of the feature has begun. Hence, the features are applied in a sequential order, and the orderings identified by the categorization approach can be used to help reduce the amount of work required to determine an optimal order for those features. Thus, using the categorization approach, which abstracts away the implementation details of the features and orders features based on their observable behaviour works well for combining different feature modules in AIN [35].

SIP: Session Initiation Protocol (SIP) is the Internet Engineering Task Force (IETF) protocol for Internet telephony. SIP runs as an end-to-end, client-server signalling protocol and is designed to be easily programmable, so that new features can be added and combined with existing features. (SIP is described more fully in Section 3.4.3.) Features are designed as extended finite state machines and implemented in proxy servers located along the call path. Several languages can be used to design features or services for SIP, including the Call Processing Language (CPL), described above, and the Specification and Design Language (SDL), which we will now consider. Using SDL, features can be implemented as modular features that reside on the same proxy server [14]. The designer can choose to either implement the features in a serial ordering or chose an alternate method for composing features, such as defining where an input signal should be directed as per scripts expressed in CPL. In the latter case, the same arguments that were made for using the categorization approach when using a CPL-based architecture are applicable here. In the first case, where features are serialized on the proxy server, the categorization approach is directly applicable and can reduce the cost of determining an optimal ordering for the features on that server.

These examples show that the categorization approach can be used to reduce the cost of prioritizing features in a variety of different domains. By modifying the basic system implemented in our Prolog model from the DFC-based telephony domain to another alternative architecture,

we would be able to prioritize features in a manner that avoids interaction among features interoperating with respect to the new architecture. The categorization of features, the category-representative feature, and the development of the principles of proper system behaviour would be reusable in other architectures, since the behaviour of the features and the principle assertions are independent of the underlying system (e.g., the feature tells the underlying system to redirect the call or to terminate the call attempt). Furthermore, the optimization methods would also be reusable in the new model as these optimizations are based solely on the discovery of a principle violation. The majority of the required changes will be to the underlying system design: the restructuring of code that determines when a feature should execute (e.g., do features execute sequentially? in parallel? based on results determined by a feature interaction manager?), the mechanism for routing signals through the features, and any other means by which the architecture coordinates feature execution (e.g., when does redirection of a call occur?, how does redirection affect which features are members of the call path?). Thus, by redesigning the underlying model and how this model interfaces with the feature modules and principle assertions, the majority of the work identified through the categorization process (i.e., the categories and the principles) can be reused to determine a priority ordering for a domain based on the alternate architecture, assuming modular feature development is supported by the new architecture.

7.1 Future Directions

Our future research interests include expanding this work to explore possible extensions to the categorization approach and exploring other domains in which feature interactions are a problem.

7.1.1 Intra-category Prioritization Cost Reduction

The use of our categorization approach dramatically reduces the cost of prioritizing a set of features. However, the cost to prioritize intra-category features remains high. We would like to research strategies to reduce this cost, perhaps by considering the feature's triggering event or subscription restrictions to decrease the number of feature combinations that need to be explored.

The first strategy considers how different triggering events (i.e., signals in telephony) that cause a feature to execute its functionality might be used to reduce the number of feature comparisons required. We hypothesize that if two features are triggered by different events, it is less likely that these features will interact. Consequently, it should be possible to order features based on their triggering event (i.e., the signal that triggers the features main functionality), and if we design feature modules so that they have only one triggering event, then each feature module needs to be ordered only with respect to other feature modules that have the same feature-related triggering event. In telephony, the different triggering events are the core set of DFC signals (*avail*, *setup*, *unavailable*, *ringTO*, *teardown*) plus *feature-specific* signals. While each feature

may respond to multiple input signals (e.g., initialization on *setup*), the feature can be designed to perform its main functionality (e.g., redirect the call) only on receipt of one specific signal type. Normally, a feature will receive only one triggering signal, although exceptions arise due to race conditions when signals are issued by different users (e.g., *teardown* is sent from one user and a *feature-specific* signal is sent from another user), but this should not affect the correctness of the generated priority ordering. Our expectation is that all categories, with the possible exception of **Multiplex**¹, will benefit from this strategy, since most features have their main functionality triggered by one specific type of signal (e.g., Call Forward on Busy, Call Forward on No Answer, and Call Forward on Setup). This strategy requires further investigation to validate whether the hypothesis holds and to determine if the cost reduction is significant when compared to the extra work that might be involved to separate features into modules based on their triggering event.

A second approach to reducing the cost of prioritizing intra-category features is to identify any subscription restrictions. The subscription restrictions are implemented by the service-provider to resolve some types of interactions by preventing the subscriber from subscribing to a specific subset of features. Thus, the subscription restriction reduces the cost of prioritizing features, since it eliminates the need to order each feature with respect to every other feature in the category. For example, some features can be subscribed to as a base feature or as an enhanced version of the base feature (e.g., Call Waiting vs. Call Waiting with Incoming Caller ID). The user would subscribe to only one of these features, hence we do not need to order these features with respect to each other. Depending on how many enhanced features are found within the category, this could result in a significant decrease in the prioritization cost. For example, the number of Call-Forwarding based features found in the different service providers, which we explored in Chapter 6, are significantly higher than the number of "usable" Call-Forwarding based features that can be active for a user at a given instance.

Another type of subscription restriction that can be explored for potential cost reduction, is that of a feature extension. There are some features that will not be subscribed to unless the user also subscribes to another feature (e.g., a user will not subscribe to Cancel Call Waiting unless she already subscribes to Call Waiting). The difference between a feature enhancement and a feature extension, is that a feature enhancement extends the functionality of an existing feature by adding new functionality, while a feature extension is the creation of a new feature that can only function properly when in the presence of another feature. We need to research this concept further, as it may be possible that most features extensions of this type are **Remote Control** features, which are implemented as extensions of the base feature (see Section 4.1). In such a case, this approach reverts to that of an enhanced feature in a subscription set as described above.

¹Multiplex features are bound features that need to coordinate between multiple connections, hence they may not be easily decomposed into different feature modules based on triggering events.

7.1.2 Other Feature Rich Domains

Currently, our research focuses on solving the feature-interaction problem with respect to the telephony domain. We would like to explore whether the categorization approach would be applicable to other feature-rich domains that suffer from feature-interaction problems. In particular, we might expect the categorization approach to work in other communication domains, such as network routing, instant messaging, and e-mail systems, since these domains share the same basic goal of providing communication between end users. We would also like to explore our approach in a non-communication-based domain, such as banking or insurance systems, because successful application in such a domain would confirm that our categorization approach is applicable to a wider variety of feature-rich domains.

The new domain must meet certain criteria in order for our categorization approach to be applicable: the domain must be able to coordinate feature modules based on a priority ordering, and the features must be designed as independent feature modules. These two conditions usually hold true for the newer components of a large system, but there may be legacy issues. Another criterion is that the features share common goals and functionality (i.e., that the feature is categorizable), and that the features are fairly distributed among the categories.

7.1.3 User Defined Priority Schemes

We have also considered how our prioritization approach might address Human-Computer Interaction (HCI) issues. An HCI interaction occurs when the user interacts with the system causing an unexpected result. For example, a user might input a new signal or cause a change to a system parameter that conflicts. One HCI issue that our categorization approach may be able to help address is whether or not we should allow a user the ability to personalize system behaviour by modifying the execution order (i.e., priorities) for a set of features.

The first step, to decide whether or not a user should be given control, is to determine whether giving the user such control can damage the functionality of the system. For example, our categorization approach has two types of principles: constraint principles that must always hold and criterion principles that should hold whenever possible. Because the criterion principles are not necessary to ensure proper system functionality, they could be violated without risking system failure. Consequently, we could allow a user to modify feature priorities in such a way that introduces criterion violations, but disallow any changes that could result in a constraint violation.

Instinctively, it might seem unwise to allow the user the freedom to reprioritize her features. However, further exploration into this possibility is warranted, because users desire the ability to personalize their systems. This research would explore how much freedom the user should have: how would we restrict the user's final priority ordering to avoid potentially dangerous (i.e., constraint violations) and undesirable (i.e., multiple criterion violations) results? What would be

the service provider's responsibility when a user changes the priority ordering of new features instead of using the default ordering: For overly constrained systems, should the company allow users to remove certain criterion principles and generate alternate, semi-personalized feature orderings? This might be a good compromise, but it restricts the user's ability to personalize individual feature orderings. Is the company required to identify potentially undesirable interactions caused by changes to the feature orderings requested by the user? If so, how would such information be communicated to the user? The potential problems versus a potential increase in user satisfaction (or dissatisfaction) warrant further research into this problem. However, our categorization approach can help answer some of the above questions and provide a mechanism to check the priority orderings suggested by the user to prevent dangerous feature orderings.

Appendix A

Feature Names and Definitions

Below is a set of features, compiled from the feature interaction benchmark [10], feature interaction contexts [24, 31], and customer-service manuals [1, 39, 41] are separated into their appropriate feature category. When appropriate, unique feature-specific signals were substituted to remove signalling ambiguities.

Alias features

Area Number Calling (ANC) determines the terminating address based partly upon the originating number [10].

Parallel Dialling (PARA) locates the called party by contacting multiple addresses simultaneously. When one of the call attempts is answered, all other calls are torn down [53].

Personal Directory (PD) is an advanced version of Speed Calling. PD contains detailed contact information, such as phone numbers, and email address, and is able to determine the best number to call for an alias based on time of day [52]. The PD feature is reversible, such that incoming calls can be identified by their corresponding alias.

Sequential Dialling (SEQ) is similar to Parallel Dialling: multiple addresses are called in an attempt to reach the callee, however only one call attempt is made at a time. If a call is not answered after a reasonable amount of time, then the call is torn down and the next number in the list is dialled [53].

Speed Calling (SC) places a call using a dialled short cut code [39]. AKA Abbreviated Dialling [1].

Billing features

Collect Call is used to reverse the long-distance charges associated with a call from the caller to the callee. AKA Reverse Charging [31].

Toll Free Call is used to allow the caller to dial a long-distance number, where the charges associated with a call are automatically charged to the callee.

Blocking features

Incoming Toll Restrictions (ITR) blocks any incoming call that will result in a charge to the subscriber [1].

Originating Call Screening (OCS) screens all outgoing calls against an outgoing-call screening list, aborting calls to numbers found on the screening list [10]. The Disabling feature **Originating Call Screening Override (OCSO)** is available.

Outgoing Toll Restrictions (OTR) aborts any outgoing call that will result in a charge to the subscriber [1].

Teen Line (TL) restricts outgoing calls based on the time of day. This restriction can be overridden with an authorization code [24].

Terminating Call Screening (TCS) screens all incoming calls against an incoming-call screening list, aborting call from numbers found in the screening list [10]. The Disabling feature **Terminating Call Screening Override (TCSO)** is also available.

Delegate features (also see Redirect features)

Call Transfer (CT) allows the subscriber to transfer the current call to another party. [31] The two remote parties can be linked “blindly” (Blind Call Transfer), so that the third party merely receives an incoming call, or the subscriber of CF can put the first party on hold, connect with the third party and then link the two remote parties together.

Send Mail (SM) is similar to Voice Mail, however the caller is given the option of leaving either a voice message or sending an email [53].

Voice Mail (VM) connects the calling party to an answering service when the subscriber is not available [31]. AKA Answer Call [10].

Filter features

Automated Role Identification (ARI) reacts to an incoming call by playing a message, which determines if the call is meant for the subscriber or another party. This feature is used when individuals change roles inside a company.

Multiplex features

Call Waiting (CW) generates a call-waiting tone to alert the subscriber to a second incoming call. The subscriber is then able to answer the incoming call by placing the current party on hold [10]. The Disabling feature **Cancel Call Waiting (CCW)** is also available.

Conference Calling (CC) sets up a call between multiple parties [1].

Three-Way Calling (TWC) allows the subscriber to add a third party to an established call. The first party is put on hold while a connection with the third party is established, then all parties can be linked together [10].

Presentation features

Call Display (CD) gathers the phone number of an incoming call and displays it on the subscriber's phone during the ringing cycle. AKA Calling Number Delivery [10].

Distinctive Ringing (DR) uses special ring tones to identify calls from specially assigned numbers [1].

Group Ringing (GR) allows an incoming call to ring at multiple phones. The first phone that is answered is connected to the calling party, and the remaining phones stop ringing [31].

Redial features

Automatic CallBack (ACB) is triggered if the subscriber calls another party whose line is busy. The feature records the number of the called party. The subscriber can then choose to activate ACB so that the called party's line is tested and the subscriber is notified when the line is no longer busy [10]. AKA Ring Back when Free and Automatic Busy Redial [1].

Return Call (RC) records the addresses of incoming calls, if the subscriber is unavailable, so that the subscriber can later automatically return a missed call [24]. AKA Automatic ReCall and Ring Back When Free [10].

Redirect features

(these features can also be Delegate features)

Call Forwarding (CF) allows incoming calls to be redirected to another address, based on some triggering event [10]. For example, the *setup* signal triggers CF Universal, the *unavail* signal triggers CF Busy Line [24], and the *ringTO* signal triggers CF No Answer. A variant is CF Universal per Key, which redirects all incoming calls, except those from numbers found in the "key-exception" list [1].

Remote-Control Invoking features

911/Emergency Services (911) prevents anyone involved in an emergency call, except the 911 operator, from either ending the call or placing the 911 operator on hold [10].

Busy Override (BO) overrides a busy signal, so that the subscriber can break into an existing conversation. A warning tone is played to notify the parties of the existing call about the impending break-in [1]. AKA Executive Busy Override. The Disabling feature **Busy Override Exempt (BOE)** is also available.

Call Park (CP) is used to "park" the current call at a different location, allowing the subscriber to change phones. AKA Mid-Call Move [52]. Variations include Call Park with Recall (CPR): if the "parked" call is not answered in a timely fashion, then the call is redirected back to its original location [1].

Call PickUp (CPU) allows authorized users to answer a call that is ringing at another location [39]. Variations include Call PickUp with Barge-in (CPUB): if a subscriber starts to pick up a call that is subsequently answered, then the subscriber will "barge-in" to the call [1]. The Disabling features **Call PickUp Exempt (CPUE)** and **Call PickUp with Barge-in Exempt** are also available.

Call Waiting Originator (CWO) gives the callee Call Waiting for the duration of the call, if the callee's line is busy and CWO is invoked [10]. The Disabling feature **Call Waiting Exempt** is also available.

Malicious Call Hold (MCH) allows the subscriber to put the call on hold so that the remote party cannot terminate the call. This feature is intended to prevent abusive calls, and allows a trace to be performed on the held call [39].

Remote Control Override

Call Display Blocking (CDB) is a directory-service feature designed to allow a subscriber to keep their number private, and prevents their number from being displayed to the called party [10]. AKA Unlisted Number.

Set Outcome features

Do Not Disturb (DND) prevents incoming calls from reaching the subscriber and causing their device to ring [1]. A variation is Do Not Disturb Screening, which blocks incoming calls, except calls from numbers on the Screening list. **Do Not Disturb Override (DNDO)** is available [1].

Make Set Busy (MSB) makes the subscriber appear unavailable by returning a busy notification, regardless of whether or not the subscriber is actually on the phone. A variant is Make Set Busy Group Exempt (MSBGE), which returns a busy notification to all incoming calls, except those from numbers are on the group exemption list.

Source Authentication features

Remote Access Authentication (RAA) verifies that a user is authorized to access a remote address category.

Appendix B

Continuation of Correctness Property

Validation 4.6.1

Correctness Property 4.6.1 Correctness of Combined Address Zones and Category Prioritizations

Given a prioritized set of feature categories, C^ , and a set of address zones ordered according to Ideal Address Translation (IAT) principles, the resulting composition of all feature categories within all address zones in the call path correctly resolves interactions with respect to the categorization and IAT principles.*

Problem Setup:

See initial problem setup for this correctness property.

Suppose a category $C_{p_n} \in A_p$ and a category $C_{q_m} \in A_q$ violate the ordering imposed by C^* . That is:

$A_p < A_q \in A^*$: satisfies IAT principles

$C_m < C_n \in C^*$: satisfies categorization principles

$C_{p_n} < C_{q_m} \in A^* \oplus C^*$: violates categorization principles

Validation:

Assumption: This property uses the terminology defined in Chapter 4 and is validated with respect to a DFC-based architecture in which features are serially composed, as defined in Chapter 2.

This validation is performed using case analysis.

We decompose the validation into ten cases, one for each principle of proper system behaviour with respect to categorization. Three of these cases are found in 4.6.1, the remaining are found below. We further decompose each of the ten principle-based cases by identifying the pairs of feature categories that can potentially result in a principle violation, which we know from our manual analysis of the categories in Section 4.5, and we examine both orderings of each pair.

Case 1: Logging Principle: Already completed in Correctness Property 4.6.1

Case 2: The Abortion Principle: Already completed in Correctness Property 4.6.1

Case 3: The Failure Principle: Already completed in Correctness Property 4.6.1

Case 4: The Accessibility Principle

The *Accessibility* principle states that all features associated with any address zone in the call will be included in every established call (i.e., voice connection is formed). We know from our manual analysis of the categories in Section 4.5, that only features that redirect a call attempt can change or prevent features from being added to the call path and still allows the call to be established. Consequently, we explore how these other categories behave in the presence of the category that can redirect a call attempt. The subcases to consider are:

1. RedCall - All Other Categories

where RedCall is any feature category that can redirect a call attempt to an alternate location (i.e., Source Redirect, Delegate, Filter, Target Redirect, and Target Redial, when triggered as it's source counterpart), and All Other Categories includes every other category found in the address zone¹.

Case 4.1: RedCall - All Other Categories

Assume the features are ordered as shown in Figure B.1. A new call attempt can be redirected in the source or target region upon receipt of the *setup* signal or any *failure* signal. The *Accessibility* principle can be violated only during the initialization of a call attempt, so we consider only call scenarios that involve the *setup* signal (either generated by the caller to initialize the call attempt or by the RedCall feature in response to a *failure signal*²), separating the subcases into incoming and outgoing *setup* signal.

In all of the cases below, we explore only those execution paths that can possibly lead to an established call. This is due to the fact, that the *Accessibility* principle can only be violated, if the call is established.

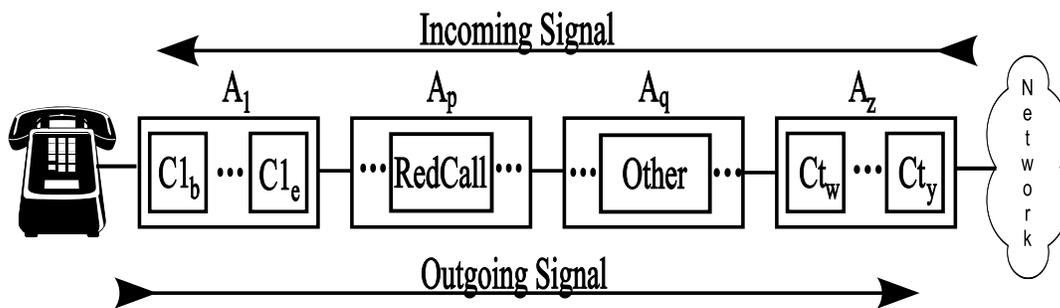


Figure B.1. This figure shows that a feature category RedCall is in address zone A_p , while the Other category is in the address zone A_q .

Incoming *setup* signal (target region): The Other feature in zone A_q is entered first and is initialized as designed. If the call attempt is continued by this feature, then the *setup* signal continues towards the RedCall feature in A_p , which redirects the call attempt to another target address zone. If the

¹This does not include Billing, which is found in the Network's address zone

²The generation of the *setup* signal, by a RedCall in response to a *failure* signal, generates a path equivalent to the RedCall reacting to the *setup* signal, since the remainder of the original call is torndown.

feature continued call attempt is established, no violation of the *Accessibility* principle will be found, since both features are initialized and remain part of the call path.

Outgoing setup signal (source region): The RedCall feature in zone A_p is entered first and redirects the call attempt to another address zone A_q . The *setup* signal is continued and enters the Other feature in zone A_p , which is initialized and continues (or prevents) the call attempt per its design. If the feature continues the call attempt and the call is established, no violation of the *Accessibility* principle will be found, since both features are initialized and remain part of the call path.

Next, we consider the alternate feature ordering shown in Figure B.2. Once again, we separate our analysis of the cases based on the direction (incoming or outgoing) of the *setup* signal and the region (target or source) of the interaction.

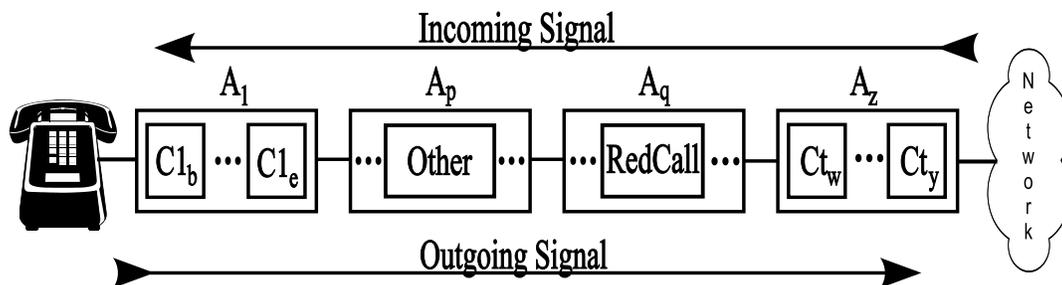


Figure B.2. This figure shows that the Other category is in address zone A_p , while a feature category RedCall, which can be any feature category that can redirect a call, is in the address zone A_q .

Incoming setup signal (target region): The RedCall feature in zone A_q is entered first and redirects the call attempt to another address zone A_p . The *setup* signal is continued and enters the Other feature in zone A_p , which is initialized and continues (or prevents) the call attempt per its design. If the feature continues the call attempt and the call is established, no violation of the *Accessibility* principle will be found, since both features are initialized and remain part of the call path.

Outgoing setup signal (source region): The Other feature in zone A_p is entered first, and the feature is initialized and continues (or prevents) the call attempt per its design. If the call attempt is continued, then the *setup* signal enters the RedCall feature in zone A_q , which continues or redirects the call attempt accordingly. If the call attempt is an established call, no violation of the *Accessibility* principle will be found, since both features are initialized and remain part of the call path.

Case 5: The Authorization Principle

The *Authorization* principle states when authentication is required, the user's identity will be verified before the user is allowed access to any of her features. (i.e., a voice connection is not fully formed until user authentication is complete). We know from our manual analysis of the categories in Section 4.5, that

authorization must occur during as part of the initialization of the call and that Target Authentication features are implemented as part of Target Redirect features. Consequently, we explore how the Auth categories behave during the initialization of a call attempt. The subcases to consider are:

1. AuthCall - All Other Categories

where AuthCall is any feature category that can authenticate a user’s ability to access a set of features (i.e., Source Authentication, and Target Redirect with Authentication), and All Other Categories includes every other category found in the address zone³.

Case 5.1: AuthCall - All Other Categories

Assume the features are ordered as shown in Figure B.3. A new call attempt can be authenticated in the source or target region upon receipt of the *setup* signal. The *Authorization* principle can be violated only during the initialization of a call attempt, so we consider only call scenarios that involve the *setup* signal, separating the subcases into incoming and outgoing *setup* signal.

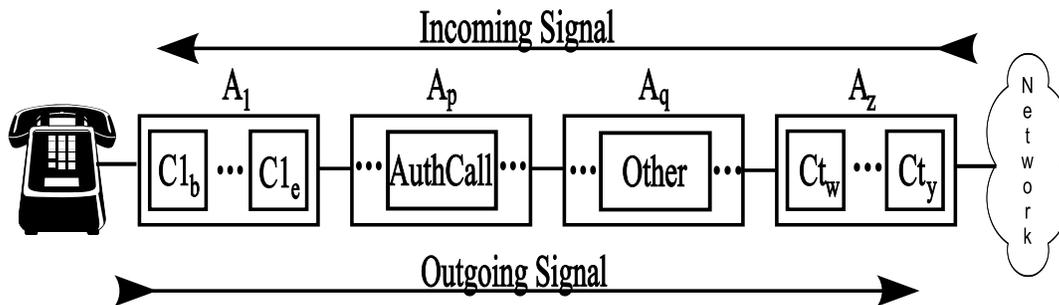


Figure B.3. This figure shows that a feature category AuthCall, which authenticates end users, is in address zone A_p , while the Other category is in the address zone A_q .

Incoming *setup* signal (target region): The Other feature in zone A_q is entered first and is initialized as designed. If the call attempt is continued by this feature, then the *setup* signal continues towards the AuthCall feature in A_p , which continues the call attempt. If the call attempt is established, the AuthCall feature reacts to the establishing of the call by prompting the callee for authentication. If the authentication is given, then the call is established and the callee is given access to all the features in the call path. Thus, neither of the features are accessed without the callee being authenticated, and thus no violation of the *Authorization* principle occurs.

Outgoing *setup* signal (source region): The AuthCall feature in zone A_p is entered first and the feature immediately requires the caller to provide authentication, so that access can be allowed to the remaining features in the call path. If authentication is provided, the *setup* signal is continued and enters the Other feature in zone A_p , which is initialized and continues (or prevents) the call attempt

³This does not include Billing, which is found in the Network’s address zone

per its design. No violation of the *Authorization* principle occurs, since both features are accessed only after the callee has provided the necessary authentication.

Next, we consider the alternate feature ordering shown in Figure B.4. Once again, we separate our analysis of the cases based on the direction (incoming or outgoing) of the *setup* signal and the region (target or source) of the interaction.

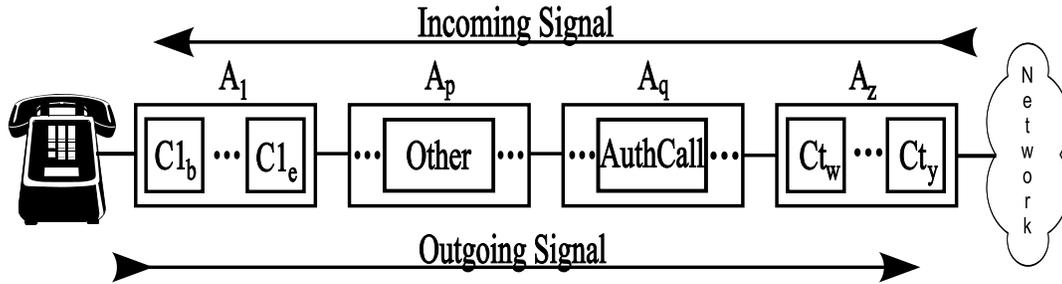


Figure B.4. This figure shows that the Other category is in address zone A_p , while a feature category AuthCall, which can be any feature category that authenticates end users, is in the address zone A_q .

Incoming *setup* signal (target region): The AuthCall feature in zone A_q is entered first which continues the call attempt. The *setup* signal is continued and enters the Other feature in zone A_p , which is initialized and continues (or prevents) the call attempt per its design. If the feature continues the call attempt and the call is established, then the AuthCall feature reacts to the establishing of the call by prompting the callee for authentication. If the authentication is given, then the call is established and the callee is given access to all the features in the call path. Notice that in this case, the callee is automatically granted access to the features in A_p , before the authentication is required in address A_q . This is a violation of the *Authorization* principle will be found, since one of the features is accessed by the callee before the authentication is given. However, because zone A_p is more concrete than zone A_q in the target region, it has priority with respect to responding to outgoing signals (the *answer* signal is sent in reverse), according to IAT principles, and thus this is an acceptable violation.

Outgoing *setup* signal (source region): The Other feature in zone A_p is entered first, and the feature is initialized and continues (or prevents) the call attempt per its design. If the call attempt is continued, then the *setup* signal enters the AuthCall feature in zone A_q , which immediately requires the caller to provide authentication, before access can be allowed to the remaining features in the call path. If authentication is provided, the *setup* signal is continued. This ordering allows the caller to access the feature in A_p before the user has been authenticated, thus the *Authorization* principle is violated. However, because zone A_p is more concrete than zone A_q in the source region, it has priority with respect to responding to outgoing signals, according to IAT principles, and thus this is an acceptable violation.

Case 6: Concretization Principle

The *Concretization* principle states that all source features that make decisions about a call's progress should have access to both the target's network address and any available alias information. Therefore, we need to consider 1) source features that can modify the call attempt (e.g., redirect, block) or 2) source features that record information to be used in subsequent call attempts (i.e., redial attempts). Thus, the subcases to consider are:

1. ModCall - Alias
2. Source Redial - Alias

where ModCall is any source feature category (i.e., Source Blocking and Source Redirect) that can modify a call attempt.

Case 6.1: ModCall - Alias

Assume that the features are ordered as shown in Figure B.5. The *Concretization* principle can be violated only during the initialization of a call attempt and in the source region, so we consider call scenarios with an outgoing *setup* signal, where the caller initiates the call with either a dialled code or network address.

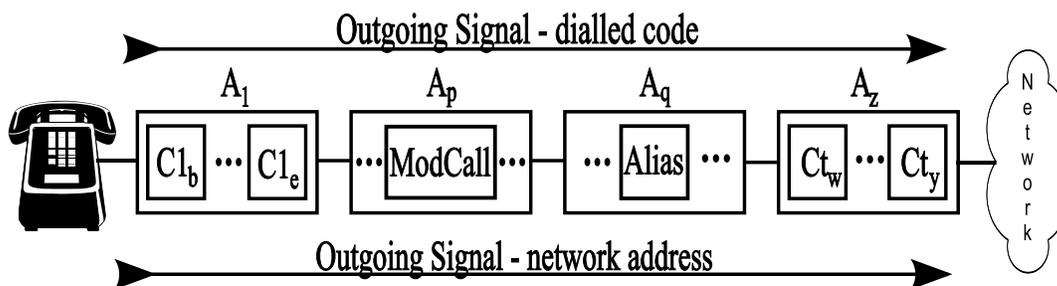


Figure B.5. This figure shows that a feature category Alias is in address zone A_q , while the ModCall category, which is any source feature that can modify a call attempt, is in the address zone A_p .

Outgoing setup signal to dialled code (source region): The ModCall feature in zone A_p is triggered by receipt of the *setup* signal and uses the dialled code to determine if the call should be modified. However, the feature does not have access to the target address that corresponds to the dialled code and hence the *Concretization* principle is violated. According to the principles of IAT, this is the correct solution, since A_p is more concrete than A_q and it has priority with respect to responding to outgoing signals, according to IAT principles, and thus this is an acceptable violation. Furthermore, the existence of the alias is determined when the caller is acting in the role of A_q , and thus it is acceptable that more concrete address A_p should not use information generated in a more abstract address zone.

Outgoing setup signal to network address (source region): The ModCall feature in zone A_p is triggered by receipt of the *setup* signal and uses the network address to determine if the call should be modified. However, the feature does not have access to the dialled code or alias that corresponds to the network address and hence the *Concretization* principle is violated. According to the principles of IAT, this is the correct solution, since A_p is more concrete than A_q and it has priority with respect to responding to outgoing signals, according to IAT principles, and thus this is an acceptable violation. Furthermore, the existence of the alias is determined when the caller is acting in the role of A_q , and thus it is acceptable that more concrete address A_p should not use information generated in a more abstract address zone.

Next, we consider the alternate feature ordering shown in Figure B.6. Once again, we separate our analysis of the cases based on the whether or not the ModCall feature can be effected by the presence of an Alias feature. Hence, we explore the subcases where an outgoing *setup* signal is used to initiate a call attempt to either 1) a dialled code or 2) a network address.

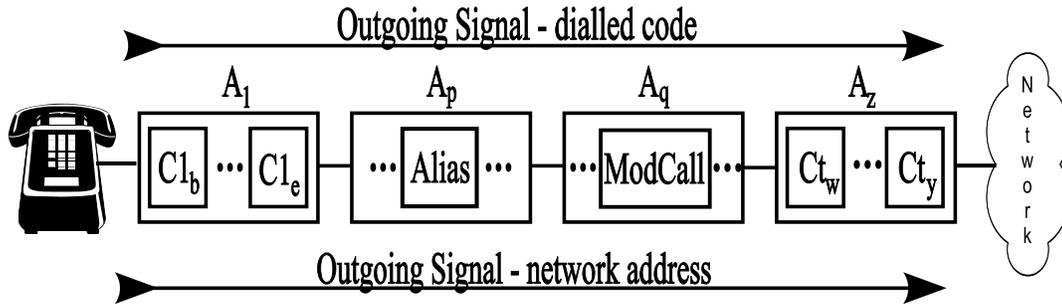


Figure B.6. This figure shows that a feature category *Alias* is in address zone A_p , while the *ModCall* category, which is any source feature that can modify a call attempt, is in the address zone A_q .

Outgoing setup signal to dialled code (source region): The *Alias* feature in zone A_p is triggered by receipt of the *setup* signal and translates the dialled code into its corresponding network address before continuing the *setup* signal towards the *ModCall* feature in zone A_q . When the *ModCall* feature receives this signal, it has access to both the network address and its corresponding dialled code for comparison against the modification list to determine whether or not the call attempt should be modified. Hence, no violation of the *Concretization* principle occurs.

Outgoing setup signal to network address (source region): The *Alias* feature in zone A_p is triggered by receipt of the *setup* signal and translates the network address into its corresponding dialled code before continuing the *setup* signal towards the *ModCall* feature in zone A_q . When the *ModCall* feature receives this signal, it has access to both the network address and its corresponding dialled code for comparison against the modification list to determine whether or not the call attempt should be modified. Hence, no violation of the *Concretization* principle occurs.

Case 6.2: Source Redial - Alias

Assume that the features are ordered as shown in Figure B.7. The *Concretization* principle can be violated only during the initialization of a call attempt and in the source region, so we consider call scenarios with an outgoing *setup* signal, where the caller initiates the call with either a dialled code or network address.

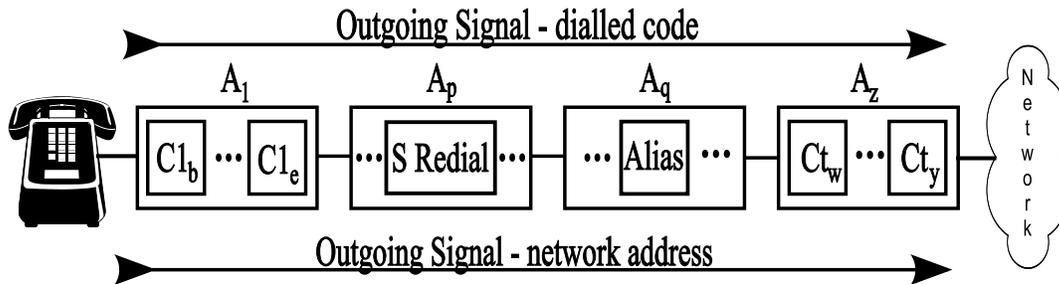


Figure B.7. This figure shows that a feature category *Alias* is in address zone A_q , while the *Source Redial* category is in the address zone A_p .

Outgoing *setup* signal to dialled code (source region): The *Source Redial* feature in zone A_p is triggered by receipt of the *setup* signal and records the dialled code to use in future call attempts. However, the feature does not have access to the target address that corresponds to the dialled code and hence the *Concretization* principle is violated. According to the principles of IAT, this is the correct solution, since A_p is more concrete than A_q and it has priority with respect to responding to outgoing signals, according to IAT principles, and thus this is an acceptable violation. Furthermore, the existence of the alias is determined when the caller is acting in the role of A_q , and thus it is acceptable that more concrete address A_p should not use information generated in a more abstract address zone.

Outgoing *setup* signal to network address (source region): The *Source Redial* feature in zone A_p is triggered by receipt of the *setup* signal and records the dialled code to use in future call attempts. However, the feature does not have access to the dialled code or alias that corresponds to the network address and hence the *Concretization* principle is violated. According to the principles of IAT, this is the correct solution, since A_p is more concrete than A_q and it has priority with respect to responding to outgoing signals, according to IAT principles, and thus this is an acceptable violation. Furthermore, the existence of the alias is determined when the caller is acting in the role of A_q , and thus it is acceptable that more concrete address A_p should not use information generated in a more abstract address zone.

Next, we consider the alternate feature ordering shown in Figure B.8. Once again, we separate our analysis of the cases based on the whether or not the *Source Redial* feature can be effected by the presence of an *Alias* feature. Hence, we explore the subcases where an outgoing *setup* signal is used to initiate a call attempt to either 1) a dialled code or 2) a network address.

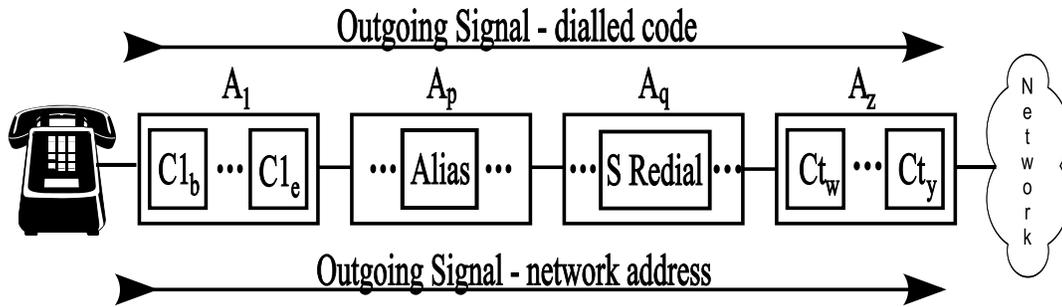


Figure B.8. This figure shows that a feature category **Alias** is in address zone A_p , while the **Source Redial** category is in the address zone A_q .

Outgoing setup signal to dialled code (source region): The **Alias** feature in zone A_p is triggered by receipt of the *setup* signal and translates the dialled code into its corresponding network address before continuing the *setup* signal towards the **Source Redial** feature in zone A_q . When the **Source Redial** feature receives this signal, it has records both the network address and its corresponding dialled code to use in future call attempts. Hence, no violation of the *Concretization* principle occurs.

Outgoing setup signal to network address (source region): The **Alias** feature in zone A_p is triggered by receipt of the *setup* signal and translates the network address into its corresponding dialled code before continuing the *setup* signal towards the **Source Redial** feature in zone A_q . When the **Source Redial** feature receives this signal, it has records both the network address and its corresponding dialled code to use in future call attempts. Hence, no violation of the *Concretization* principle occurs.

Case 7: Invoicing Principle

The *Invoicing* principle states that the cost of all subcalls is billed to some user. The *c* for this principle does not follow the pattern of the remainder of these cases, because the *Invoicing* principle's only concern is that the **Billing** category be added to the call path each time the call attempt transitions from one subcall (e.g., address zone) to another. By design, we know that the **Billing** category is added to the call path whenever the Network's address zone is entered, and that any transition between subcalls is managed by the Network's address zone. Thus, the **Billing** category is added each time a transition between subcalls is initiated and the cost of the subcall is billed to some user, so no violation of the *Invoicing* principle exists.

Case 8: Network Principle

The *Network* principle states that all dialled codes must be translated into their corresponding network address before reaching the Network's address zone, which uses this address to direct the call attempt to the next address zone. The *Network* principle is violated when the **Alias** feature fails to update the dialled code before it reaches the network. The subcases to consider are:

1. Multiplex - Alias
2. RedCall - Alias

where RedCall is any feature category that can redirect a call attempt to an alternate location (i.e., Delegate, Filter, Target Redirect, Source Redirect and Target Redial).

Case 1.1: Multiplex - Alias

In this case, we explore the possibilities of an interaction between the Multiplex and Alias categories. The *Network* principle can be violated only during the initialization of a call attempt in the source region, when a dialled code is entered by the caller. Therefore, we consider call scenarios with an incoming or outgoing *setup* signal. Multiplex features add another layer to our analysis as we must consider whether or not the subscriber's Multiplex feature is already active.

Assume that the features are ordered as in Figure B.9.

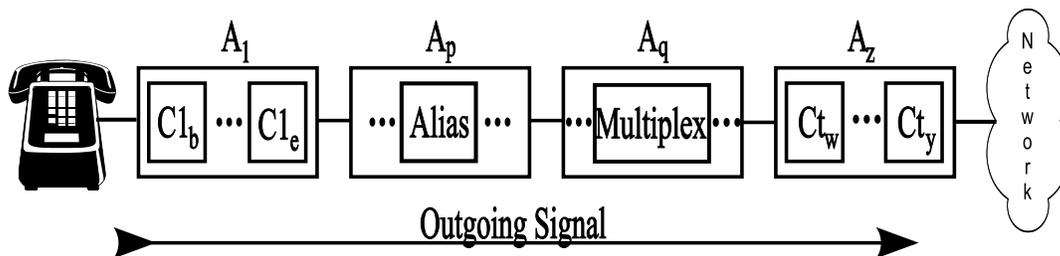


Figure B.9. This figure shows that a feature category Alias is in address zone A_p , while the Multiplex category is in the address zone A_q .

Non-active Multiplex

Consider the case in which Multiplex is not active when the call attempt is initialized.

Outgoing *setup* signal to dialled code (source region): The Alias feature in zone A_p is triggered by receipt of the *setup* signal and translates the dialled code into its corresponding network address. The Alias feature continues the call attempt, which enters and activates the Multiplex feature. When the Multiplex feature continues the call attempt, the *setup* signal eventually reaches the Network address that directs the call attempt into the target region and uses the translated network address to continue the call attempt. Since the network address is available, no violation of the **Network** principle exists.

Active Multiplex

Consider the case in which Multiplex is already active.

Outgoing *setup* signal to dialled code (source region): Suppose that the Alias feature is on the subscriber side of Multiplex in the source region, as shown in Figure B.9, and the user initiates a new call attempt by triggering a Multiplex feature that exists in an already established call in which

the user is currently involved. In this situation, the *Alias* feature is bypassed because the new call attempt is generated by the *Multiplex* feature, which means the *setup* signal does not pass through the *Alias* feature, and the dialled code is not updated by the *Alias* feature. This call attempt will fail and a violation of the *Network* principle occurs, because the call attempt is continued by the *Multiplex* feature and the *Network*'s address zone is reached without the dialled code being translated. According to the principles of IAT, this is the correct solution, since the caller chooses to act in the role of A_q when using A_q 's *Multiplex* feature to initiate a new call attempt. That is a call initiated by a more abstract source address zone, should not have access to features in a more concrete address zone.

Next, we consider the alternate feature ordering shown in Figure B.10. Once again, we analyze the generation of the *setup* signal issued with a dialled code in the source region.

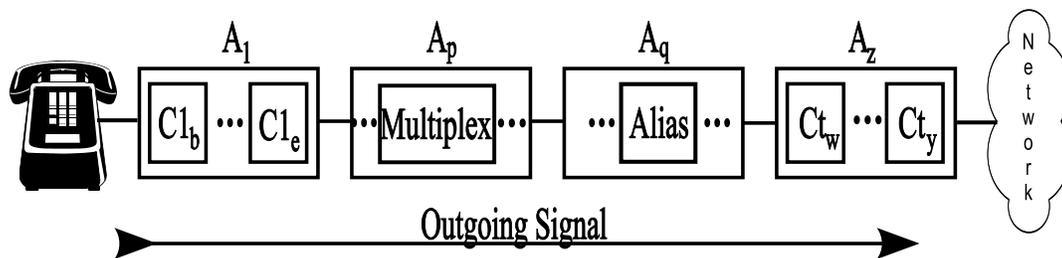


Figure B.10. This figure shows that the *Multiplex* category is in address zone A_p , while a feature category *Alias* is in the address zone A_q .

Non-active *Multiplex*

Consider the case in which *Multiplex* is not active when the call attempt is initialized.

Outgoing *setup* signal to dialled code (source region): The *Multiplex* feature in zone A_p is triggered by receipt of the *setup* signal and becomes active. The *Multiplex* feature continues the call attempt, which enters and the *Alias* feature. The *Alias* feature translates the dialled code into its corresponding network address and continues the call attempt. When the *setup* signal eventually reaches the network address that directs the call attempt into the target region is reached, it uses the translated network address to continue the call attempt. Since the network address is available, no violation of the *Network* principle exists.

Active *Multiplex*

Consider the case in which *Multiplex* is already active.

Outgoing *setup* signal to dialled code (source region): Suppose that the *Multiplex* feature is on the subscriber side of *Alias* in the source region, as shown in Figure B.10, and the user initiates a new call attempt by triggering a *Multiplex* feature that exists in an already established call in which the user is currently involved. In this situation, the *Multiplex* feature generates a new call attempt, and the

setup signal passes through the *Alias* feature in A_q . The *Alias* feature translates the dialled code into its corresponding network address, before continuing the call attempt. When the Network address that directs the call attempt into the target region is reached, it uses the translated network address to continue the call attempt. Since the network address is available, no violation of the *Network* principle exists.

Case 8.2: RedCall - Alias

This is a subcase of the one we explored in Case 4.1 of the *Accessibility* principle. That is if we replace the *Other* feature with an *Alias* feature, we discover that the *Alias* feature is always accessible. This means that the *Alias* feature is applied during a call attempt regardless of where it is located with respect to the *RedCall* feature, thus any dialled code is correctly translated into its corresponding network address before the Network address zone, which directs the call attempt into the target region, is reached. Therefore, there is no violation of the *Network* principle.

Case 9: Personalization Principle

The *Personalization* principle states that aliases should be used, when they exist, whenever user-related information is presented to or recorded by the subscriber. The *Personalization* principle is violated when an *Alias* feature translates an incoming *setup* signal after the call information has been 1) presented or 2) or recorded. The subcases to consider are:

1. Presentation - Alias
2. Redial - Alias

Case 9.1: Presentation - Alias

We consider the receipt of a *setup* signal in the either region. Furthermore, since an *Alias* feature is present in the call path, an outgoing call attempt can be made to either 1) a dialled code or 2) a network address.

Assume the features are ordered as shown in Figure B.11.

Outgoing *setup* signal to dialled code (source region): The *Presentation* feature in zone A_p is triggered by receipt of the *setup* signal and uses the dialled code to present information back to the caller. The call attempt is continued, and the *setup* signal enters the *Alias* feature in zone A_q , which translates the network address into its corresponding code and stores it in the alias field if such a code exists.⁴ The information presented to the user includes the dialled code, and therefore the *Personalization* principle is not violated.

Outgoing *setup* signal to network address (source region): The *Presentation* feature in zone A_p is triggered by receipt of the *setup* signal and uses the network address to present information back

⁴If the dialled code is not translated, then a violation of the *Network* principle occurs and the call attempt cannot be completed.

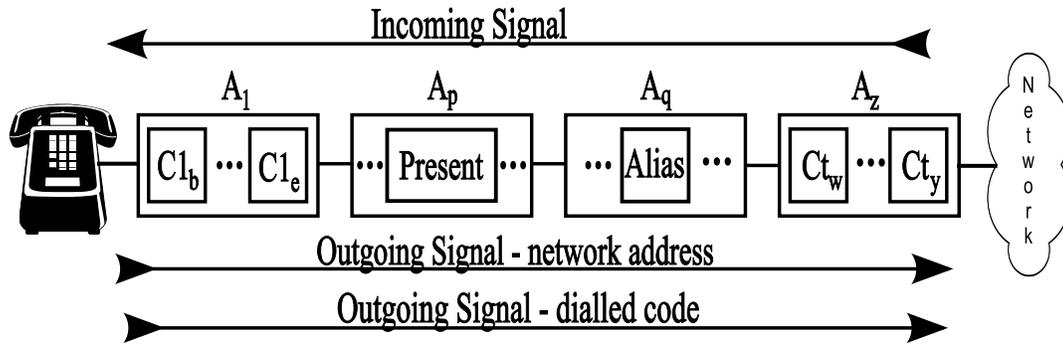


Figure B.11. This figure shows that a feature category Presentation is in address zone A_p and the Alias category is in the address zone A_q .

to the caller. The call attempt is continued, the *setup* signal continues towards the Alias feature in zone A_q , which determines and records the alias mapping for the network address, if one exists. If a mapping does not exist for the network address, then the *Personalization* principle is not violated since there is not alias mapping to present to the caller.

However, if a mapping does exist for the network address, then the *Personalization* principle is violated, since there is an alias mapping for this network address and it was not to present to the caller. However, because zone A_p is more concrete than zone A_q in the target region, it has priority with respect to responding to outgoing signals, according to IAT principles, and thus this is an acceptable violation. Furthermore, the existence of the alias is determined when the caller is acting in the role of A_q , and thus it is acceptable that more concrete address A_p should not use information generated in a more abstract address zone.

Incoming *setup* signal (target region): The Alias feature in zone A_q is triggered by receipt of the *setup* signal and translates the incoming network address to its corresponding alias, if one exists. The call attempt is continued, the *setup* signal continues towards the Presentation feature in zone A_p , which presents both the network address and any alias information about the incoming call to the callee. Since the information presented to the user includes the dialled code, the *Personalization* principle is not violated.

Next, we consider the alternate feature ordering shown in Figure B.12. Once again, we separate our analysis of the cases based on the direction (incoming or outgoing) of the *setup* signal and the region (target or source) of the interaction, and whether or not a dialled code or network address was used to initialize the call.

Outgoing *setup* signal to dialled code (source region): The Alias feature in zone A_p is triggered by receipt of the *setup* signal and translates the dialled code into its corresponding network address before continuing the *setup* signal towards the Presentation feature in zone A_q . When the Presentation feature receives this signal, it uses the network address and dialled code when presenting

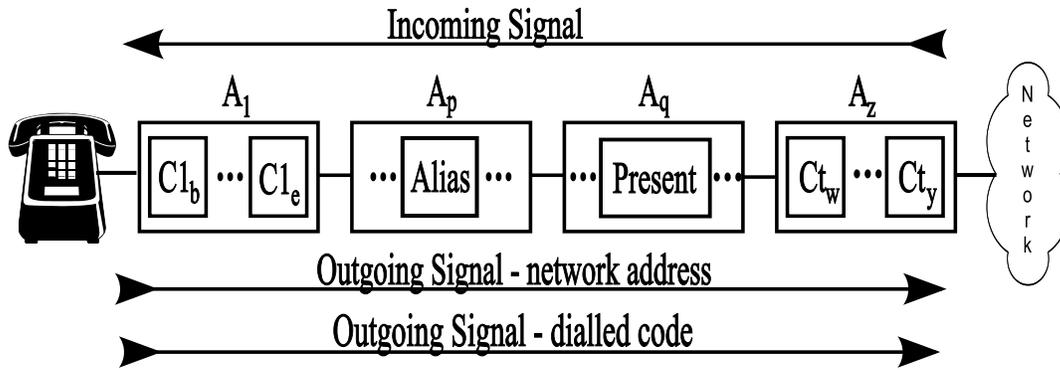


Figure B.12. This figure shows that a feature category *Alias* is in address zone A_p , while the *Presentation* category is in the address zone A_q .

information to the caller. Since the information presented to the user includes the dialled code, the *Personalization* principle is not violated.

Outgoing *setup* signal to network address (source region): Similarly, there is no violation of the *Personalization* principle in this call scenario. The *Alias* feature in zone A_p is triggered by receipt of the *setup* signal and translates the network address into its corresponding code, which is stored in the alias field if such a code exists. The *setup* signal is then continued towards the *Presentation* feature in zone A_q , which uses the network address and dialled code to present information to the caller. Since the information presented to the user includes the dialled code, the *Personalization* principle is not violated.

Incoming *setup* signal (target region): The *Presentation* feature in zone A_q is triggered by receipt of the *setup* signal and presents the network address of the incoming call attempt to the callee, and continues the call attempt. The *setup* signal continues towards the *Alias* feature in zone A_p , which determines and records the alias mapping for the network address, if one exists. If a mapping does not exist for the network address, then the *Personalization* principle is not violated since there is not alias mapping to present to the callee.

However, if the a mapping does exist for the network address, then the *Personalization* principle is violated, since there is an alias mapping for this network address and it was not to present to the callee. However, because zone A_q is more abstract than zone A_p in the target region, it has priority with respect to responding to incoming signals, according to IAT principles, and thus this is an acceptable violation. Furthermore, the existence of the alias is determined when the callee is acting in the role of A_p , and thus it is acceptable that more concrete address A_p should not supply information to a more abstract address zone, such as A_q .

Case 9.2: Redial - Alias

We consider the receipt of a *setup* signal in the either region. Furthermore, since an *Alias* feature is present in the call path, an outgoing call attempt can be made to either 1) a dialled code or 2) a network address.

Assume the features are ordered as shown in Figure B.13.

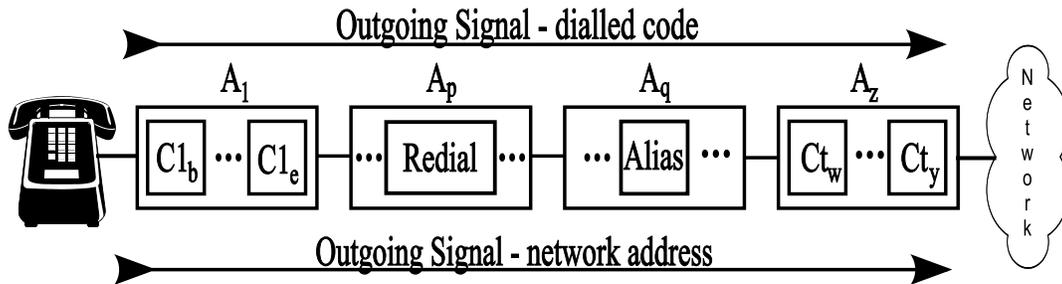


Figure B.13. This figure shows that a feature category *Redial* is in address zone A_p and the *Alias* category is in the address zone A_q .

Outgoing *setup* signal to dialled code (source region): The *Redial* feature in zone A_p is triggered by receipt of the *setup* signal and uses the dialled code record call back information for the caller. The call attempt is continued, and the *setup* signal enters the *Alias* feature in zone A_q , which translates the network address into its corresponding code and stores it in the alias field if such a code exists. ⁵ The information recorded by the user includes the dialled code, and therefore the *Personalization* principle is not violated.

Outgoing *setup* signal to network address (source region): The *Redial* feature in zone A_p is triggered by receipt of the *setup* signal and uses the network address to record information for the caller. The call attempt is continued, the *setup* signal continues towards the *Alias* feature in zone A_q , which determines and records the alias mapping for the network address, if one exists. If a mapping does not exist for the network address, then the *Personalization* principle is not violated since there is not alias mapping is recorded by the caller.

However, if the a mapping does exist for the network address, then the *Personalization* principle is violated, since there is an alias mapping for this network address and it was not recorded by the caller. However, because zone A_p is more concrete than zone A_q in the target region, it has priority with respect to responding to outgoing signals, according to IAT principles, and thus this is an acceptable violation. Furthermore, the existence of the alias is determined when the caller is acting in the role of A_q , and thus it is acceptable that more concrete address A_p should not use information generated in a more abstract address zone.

⁵If the dialled code is not translated, then a violation of the *Network* principle occurs and the call attempt cannot be completed.

Incoming *setup* signal (target region): The Alias feature in zone A_q is triggered by receipt of the *setup* signal and translates the incoming network address to its corresponding alias, if one exists. The call attempt is continued, the *setup* signal continues towards the Redial feature in zone A_p , which records both the network address and any alias information about the incoming call to the callee. Since the information recorded by the user includes the dialled code, the *Personalization* principle is not violated.

Next, we consider the alternate feature ordering shown in Figure B.14. Once again, we separate our analysis of the cases based on the direction (incoming or outgoing) of the *setup* signal and the region (target or source) of the interaction, and whether or not a dialled code or network address was used to initialize the call.

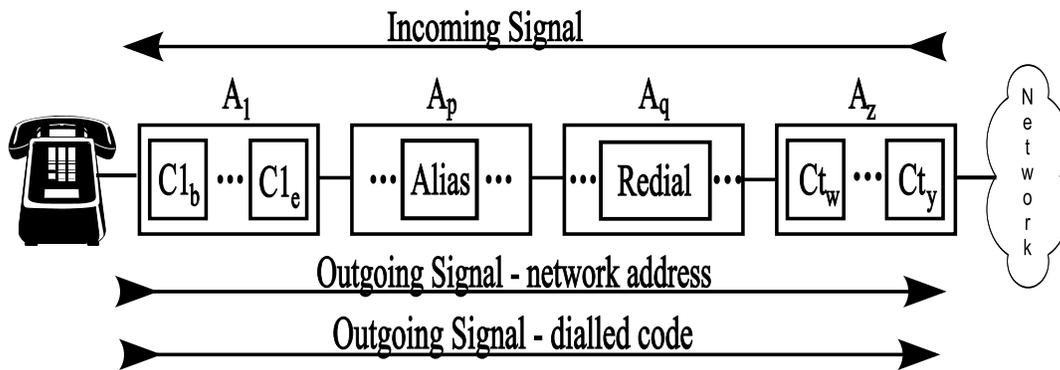


Figure B.14. This figure shows that a feature category Alias is in address zone A_p , while the Redial category is in the address zone A_q .

Outgoing *setup* signal to dialled code (source region): The Alias feature in zone A_p is triggered by receipt of the *setup* signal and translates the dialled code into its corresponding network address before continuing the *setup* signal towards the Redial feature in zone A_q . When the Redial feature receives this signal, it uses the network address and dialled code when recording information for the caller. Since the information recorded includes the dialled code, the *Personalization* principle is not violated.

Outgoing *setup* signal to network address (source region): Similarly, there is no violation of the *Personalization* principle in this call scenario. The Alias feature in zone A_p is triggered by receipt of the *setup* signal and translates the network address into its corresponding code, which is stored in the alias field if such a code exists. The *setup* signal is then continued towards the Redial feature in zone A_q , which uses the network address and dialled code when recording information. Since the information recorded includes the dialled code, the *Personalization* principle is not violated.

Incoming *setup* signal (target region): The Redial feature in zone A_q is triggered by receipt of the *setup* signal and records the network address of the incoming call attempt to the callee, and continues the call attempt. The *setup* signal continues towards the Alias feature in zone A_p , which

determines and records the alias mapping for the network address, if one exists. If a mapping does not exist for the network address, then the *Personalization* principle is not violated since there is no alias mapping to be recorded.

However, if an alias mapping does exist for the network address, then the *Personalization* principle is violated, since there is an alias mapping for this network address and it was not recorded. However, because zone A_q is more abstract than zone A_p in the target region, it has priority with respect to responding to incoming signals, according to IAT principles, and thus this is an acceptable violation. Furthermore, the existence of the alias is determined when the callee is acting in the role of A_p , and thus it is acceptable that more concrete address A_p should not supply information to a more abstract address zone, such as A_q .

Case 10: Presentation Principle

The *Presentation* principle states that only information about incoming calls that can reach the subscriber's end device should be presented to the subscriber, and all calls that reach the subscriber's end device should be presented, when a presentation feature is subscribed to. The *Presentation* principle is violated when a Presentation feature presents information about an incoming *setup* signal and the call attempt is then 1) terminated or 2) redirected, or when 3) a call reaches the end device without first being presented. The subcases to consider are:

1. TRedCall - Presentation
2. TermCall - Presentation
3. Multiplex - Presentation

where TRedCall is any target feature category that can redirect a call attempt to an alternate location (i.e., Delegate, Filter, and Target Redirect) and TermCall is any target feature category that can terminate a call attempt (i.e., Filter, Set Outcome, and Target Block),

Case 10.1: TRedCall - Presentation

Assume the features are ordered as shown in Figure B.17. The *Presentation* principle can be violated only during the initialization of a call attempt in the target region, so we consider the subcase where an incoming *setup* signal is received.

Incoming *setup* signal (target region): The Presentation feature in zone A_q is entered first, and the call attempt is presented to the subscriber's end device, and then the call attempt is continued towards the TRedCall feature in A_p . Address A_p 's TRedCall feature causes the call to be redirected to another address zone, which may or may not involve the subscriber. A violation of the *Presentation* principle occurs, since the call attempt is redirected to an alternate address zone and the call attempt does not reach the end device corresponding to this address zone, despite the fact that the call attempt has already been presented. However, because zone A_q is more abstract than zone A_p

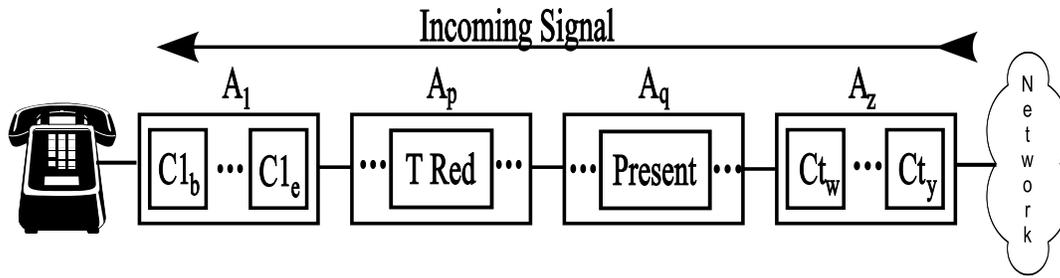


Figure B.15. This figure shows that a feature category TRedCall (T Red) is in address zone A_p , while the Presentation (Present) category is in the address zone A_q .

in the target region, it has priority with respect to responding to incoming signals, according to IAT principles, and thus this is an acceptable violation.

Next, we consider the alternate feature ordering shown in Figure B.18. Once again, we analyze the cases where an incoming *setup* signal is received.

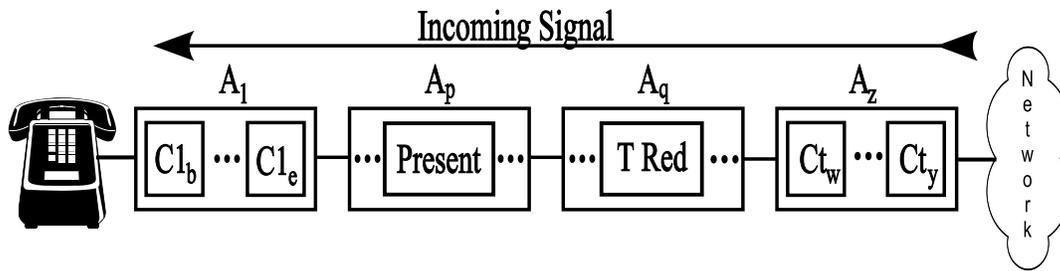


Figure B.16. This figure shows that the Presentation (Present) category is in address zone A_p , while a feature category TRedCall (T Red), which can be any target feature category that can redirect a call, is in the address zone A_q .

Incoming setup signal (target region): The TRedCall feature in zone A_q is entered first and redirects the call attempt to another address zone A_p . The *setup* signal is continued and enters the Presentation feature in zone A_p . The call attempt is then presented to the subscriber’s end device, before continuing the call attempt. This call attempt eventually reaches the end device and there is no violation of the *Presentation* principle.

Case 10.2: TermCall - Presentation

Assume the features are ordered as shown in Figure B.17. The *Presentation* principle can be violated only during the initialization of a call attempt in the target region, so we consider the subcase where an incoming *setup* signal is received.

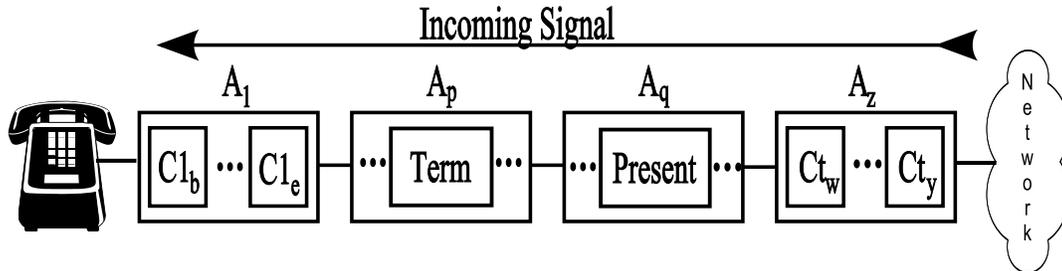


Figure B.17. This figure shows that a feature category TermCall (Term), which is any feature category that can terminate a call attempt, is in address zone A_p , while the Presentation (Present) category is in the address zone A_q .

Incoming setup signal (target region): The Presentation feature in zone A_q is entered first, and the call attempt is presented to the subscriber’s end device, and then the call attempt is continued towards the TermCall feature in A_p . Address A_p ’s TermCall feature terminates the call attempt. This results in the violation of the *Presentation* principle, since the call attempt has been presented to the subscriber’s end device, but the call attempt is prevented from reaching the end device. However, because zone A_q is more abstract than zone A_p in the target region, it has priority with respect to responding to incoming signals, according to IAT principles, and thus this is an acceptable violation.

Next, we consider the alternate feature ordering shown in Figure B.18. Once again, we analyze the cases where an incoming *setup* signal is received.

Incoming setup signal (target region): The RedCall feature in zone A_q is entered first and terminates the call attempt. There is no violation of the *Presentation* principle, since the call attempt is not presented to the subscriber’s end device.

Case 10.3: Multiplex - Presentation

In this case, we explore the possibilities of an interaction between the Multiplex and Presentation categories. The *Presentation* principle can be violated only during the initialization of a call attempt in the target region. Multiplex features add another layer to our analysis as we must consider whether or not the subscriber’s Multiplex feature is already active.

Assume that the features are ordered as in Figure B.19 and consider the case in which Multiplex is already active.

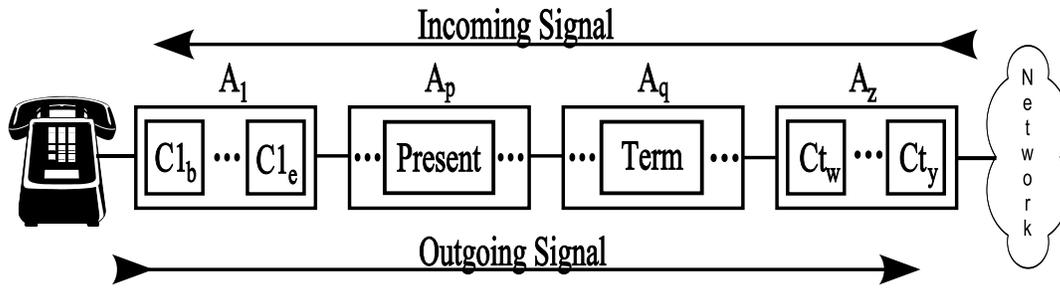


Figure B.18. This figure shows that the Presentation (**Present**) category is in address zone A_p , while a feature category TermCall (**Term**), which can be any target feature category that can terminates a call attempt, is in the address zone A_q .

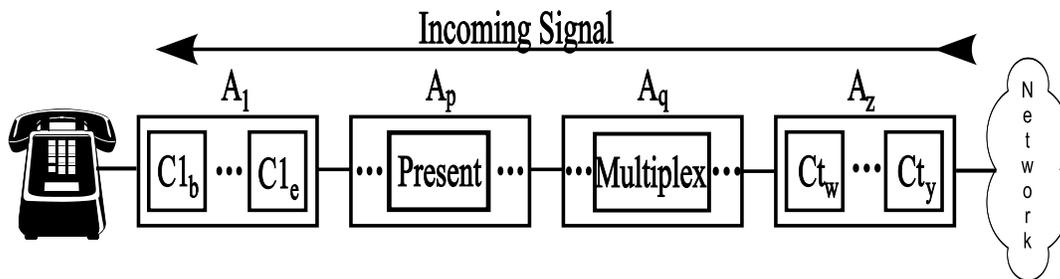


Figure B.19. This figure shows that a feature category Presentation (**Present**) is in address zone A_p , while the Multiplex category is in the address zone A_q .

Non-active Multiplex

Incoming setup signal (target region): When the Multiplex in address A_q is not in use and receives a *setup* signal, it initializes the feature and continues the *setup* signal unchanged. The **Presentation** feature in zone A_p presents the call attempt to the subscriber's end device, before continuing the call attempt. This call attempt eventually reaches the end device and there is no violation of the *Presentation* principle.

Active Multiplex

Incoming setup signal (target region): Suppose that the **Presentation** feature is on the subscriber side of **Multiplex** in the target region, as shown in Figure 4.8, and the **Multiplex** feature is already in use, when the incoming *setup* is received. The **Multiplex** feature in zone A_q reacts to this call attempt by performing an action that allows the subscriber to interact with the new call attempt. However, because of the manner in which **Multiplex** features are designed, the *setup* signal is not propagated through the remaining features in the call path; instead the **Multiplex** feature notifies the subscriber of the new call using either the existing voice connection or a *feature-specific* signal along the call path. If the existing voice connection is used, instead of the typical signal routing through

the features' ports, then the *Presentation* feature does not receive any signal. Alternatively, if the *Presentation* feature receives a *feature-specific* signal, it may not recognize the signal and may behave transparently. In either case, the *Presentation* feature in zone A_p does not present the call attempt regarding this incoming call, and the *Presentation* principle is violated. However, when the user accepts this incoming call, she is doing so in the role of the address zone A_q , interacting with the address A_q 's *Multiplex* feature. Hence, it is acceptable that the features in the more concrete address zone, A_p , do not record information about this call.

Next, we consider the alternate feature ordering shown in Figure B.20. Once again, we separate our analysis of the cases based on whether or not the *Multiplex* feature is active.

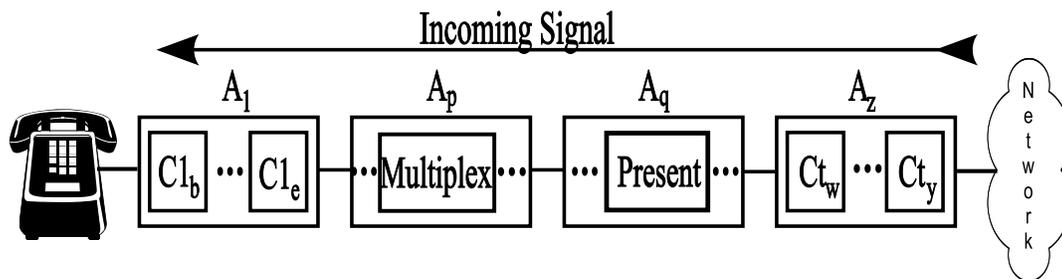


Figure B.20. This figure shows that the *Multiplex* category is in address zone A_p , while a feature category *Presentation* (*Present*) is in the address zone A_q .

Non-active *Multiplex*

Incoming *setup* signal (target region): The *Presentation* feature in zone A_q is entered first and presents the call attempt to the subscriber, and then continues the *setup* signal. When the signal reaches the *Multiplex* feature in zone A_p , the feature becomes active and continues the incoming call attempt. Eventually, the call attempt reaches the end device. Hence, no violation of the *Presentation* principle occurs, since the call attempt has been presented before the call reaches the end device.

Active *Multiplex*

Incoming *setup* signal (target region): The *Presentation* feature in zone A_q is entered first and presents the call attempt to the subscriber, and then continues the *setup* signal. When the signal reaches the active *Multiplex* feature in zone A_p , the incoming call attempt will be processed and presented to the user as designed. Hence, no violation of the *Presentation* principle occurs, since the call attempt has been presented before the call reaches the end device.

Bibliography

- [1] 3com Technical Guide. *IP Telephony Jargon Buster and Glossary*. http://www.3com.com/voip/assets/3com_200251-003.pdf.
- [2] A. Aho, S. Gallagher, N. Griffeth, C. Schell, and D. Swayne. Scf3TM/sculptor with chisel: Requirements engineering for communications services. In *Feature Interactions in Telecommunications and Software Systems V*, 1998.
- [3] M. Amer, A. Karmouch, T. Gray, and S. Mankovskii. Feature-Interaction Resolution Using Fuzzy Policies. In *Feature Interactions in Telecommunications Systems VI*, pages 94–113, 2000.
- [4] D. Amyot, L. Charfi, N. Gorse, T. Gray, L. Logrippo, J. Sincennes, B. Stepien, and T. Ware. Feature description and feature interaction analysis with use case maps and lotos. In *Feature Interactions in Telecommunications Systems VI*, page 274, 2000.
- [5] M. P. O. Blumenthal and P. Russo. Addressing feature interactions during service creation. In *Feature Interactions in Telecommunications and Distributed Systems IV*, 1997.
- [6] L. G. Bouma and H. Velthuijsen, editors. *Feature Interactions in Telecommunications Systems*. IOS Press, 1994.
- [7] J. F. Brule. Fuzzy Systems /- A Tutorial. <http://www.ortech-engr.com/fuzzy/tutor.txt>.
- [8] R. J. A. Buhr, D. Amyot, M. Elammari, D. Quesnel, T. Gray, and S. Mankovski. Feature-interaction visualization and resolution in an agent environment. In *Feature Interactions in Telecommunications and Software Systems V*, 1998.
- [9] M. Calder and E. Magill, editors. *Feature Interactions in Telecommunications and Software Systems VI*. IOS Press, 2000.
- [10] E. J. Cameron, N. D. Griffeth, Y.-J. Lin, M. E. Nilson, W. K. Schnure, and H. Velthuijsen. Feature Interaction Benchmark for IN and Beyond. In *Feature Interactions in Telecommunications Systems II*, pages 1–23, 1994.
- [11] J. Cameron, K. Cheng, S. Gallagher, F. J. Lin, P. Russo, and D. Sobik. Next generation service creation: Process, methodology, and tool integration. In *Feature Interactions in Telecommunications and Software Systems V*, 1998.
- [12] J. Cameron, K. Cheng, F. J. Lin, H. Liu, and B. Pinheiro. A formal ain service creation, feature interactions analysis and management environment: An industrial application. In *Feature Interactions in Telecommunications and Distributed Systems IV*, 1997.
- [13] D. Cattrall, G. Howard, D. Jordan, and S. Buj. An Interaction-Avoiding Call Processing Model. In *Feature Interactions in Telecommunications Systems III*, pages 85–96, 1995.

- [14] K. Y. Chan and G. V. Bochmann. Methods for designing sip services in sdl with fewer feature interactions. In *Feature Interactions in Telecommunications and Software Systems VII*, pages 59–76, 2003.
- [15] Y.-L. Chen, S. Lafortune, and F. Lin. Design of nonblocking modular supervisors using event priority functions. *IEEE Transactions on Automatic Control*, 45(3):432–452, March 2000.
- [16] K. Cheng and T. Ohta, editors. *Feature Interactions in Telecommunications Systems III*. IOS Press, 1995.
- [17] P. Dini, R. Boutaba, and L. Logrippo, editors. *Feature Interactions in Telecommunications and Distributed Systems IV*. IOS Press, 1997.
- [18] C. D. Elfe, E. C. Freuder, and D. Lesaint. Dynamic constraint satisfaction for feature interaction. In *BT Technology Journal*, volume 16, number 3, pages 38–45, July 1998.
- [19] M. Faci and L. Logrippo. Specifying feature and analysing their interactions in a lotos enviroment. In *Feature Interactions in Telecommunications Systems*, page 136, 1994.
- [20] N. Fritsche. Runtime resolution of feature interactions in architectures with separated call and feature control. In *Feature Interactions in Telecommunications Systems III*, 1995.
- [21] J. P. Gibson. Feature requirements models: Understanding interactions. In *Feature Interactions in Telecommunications and Distributed Systems IV*, 1997.
- [22] J. P. Gibson. Towards a feature interaction algebra. In *Feature Interactions in Telecommunications and Software Systems V*, 1998.
- [23] N. Griffeth and H. Velthuisen. The negotiating agents approach to runtime feature interaction resolution. In *Feature Interactions in Telecommunications Systems*, 1994.
- [24] N. Griffith, R. Blumenthal, J.-C. Grégoire, and T. Ohta. Feature interaction detection contest. In *Feature Interactions in Telecommunications and Software Systems V*, 1998.
- [25] R. J. Hall. Feature Interaction in Electronic Mail. In *Feature Interactions in Telecommunications Systems VI*, pages 67–82, 2000.
- [26] Y. Harada, Y. Hirakawa, and T. Takenaka. A design support method for telecommunication service interactions. In *GLOBECOM '91. Countdown to the New Millennium. Featuring a Mini-Theme on: Personal Communications Services.*, volume 3, pages 1661–1666, 1991.
- [27] M. Jackson and P. Zave. “Distributed Feature Composition: A Virtual Architecture for Telecommunications Services”. *IEEE Transactions on Software Engineering*, 24(10):831–847, October 1998.
- [28] A. L. Juarez Dominguez and N. A. Day. Compositional reasoning for port-based distributed systems. In *ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 376–379, New York, NY, USA, 2005. ACM Press.
- [29] S. Kawauchi and T. Ohta. Mechanism for 3-way feature interactions occurrence and a detection system based on the mechanism. In *Feature Interactions in Telecommunications and Software Systems VII*, pages 313–328, 2003.
- [30] K. Kimbler and L. G. Bouma, editors. *Feature Interactions in Telecommunications and Software Systems V*. IOS Press, 1998.
- [31] M. Kolberg, E. H. Magill, D. Marples, and S. Reiff. Second feature interaction contest results. In *Feature Interactions in Telecommunications and Software Systems VI*, 2000.

- [32] J. Kuthan and D. Sisalem. Session initiation protocol (sip) tutorial. <http://www.cs.columbia.edu/sip/papers.html>.
- [33] J. Lennox and H. Schulzrinne. *Call Processing Language Framework and Requirements*. Columbia University, 2000. Network Working Group Memo.
- [34] J. Lennox and H. Schulzrinne. Feature interaction in internet telephony. In *Feature Interactions in Telecommunications and Software Systems VI*. IOS Press, 2000.
- [35] F. J. Lin, H. Liu, and A. Ghosh. A methodology for feature interaction detection in the ain 0.1 framework. *IEEE Transactions on Software Engineering*, 24(10):797–817, October 1998.
- [36] D. Marples and E. H. Magill. The use of rollback to prevent incorrect operation of features in intelligent network based systems. In *Feature Interactions in Telecommunications and Software Systems V*, 1998.
- [37] M. Nakamura, T. Kikuno, J. Hassine, and L. Logrippo. Feature interaction filtering with use case maps at requirements stage. In *Feature Interactions in Telecommunications Systems VI*, page 163, 2000.
- [38] M. Nakamura and Y. Tsuboi. A method for detecting and eliminating feature interactions using a frame model. In *Communications, 1995. ICC 95 Seattle, Gateway to Globalization, 1995 IEEE International Conference on*, volume 1, pages 99–103, 1995.
- [39] Nortel Networks. *Centrex Feature Library*. http://products.nortel.com/go/product_index.jsp.
- [40] Nortel Networks. *Centrex Feature Library - Glossary*. www.nortelnetworks.com/products/01/centrex/library/list.html.
- [41] Nortel Networks. *Centrex Feature Library - Voice Features*. www.nortelnetworks.com/products/01/centrex/library/voice/index.html.
- [42] RADVision white paper. *What is SIP*. <http://www.cs.columbia.edu/sip/papers.html>.
- [43] S. Reiff-Marganiec and M. Ryan, editors. *Feature Interactions in Telecommunications and Software Systems VIII*. IOS Press, 2005.
- [44] R. W. Sebesta. *Concepts of Programming Languages*. Addison-Wesley, third edition, 1996.
- [45] S. Shapiro and Y. Lesperance. Modeling multiagent systems with casl - a feature interaction resolution application. In *Intelligent Agents Volume VII – Proceedings of the 2000 Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, 2001.
- [46] B. Stepien and L. Logrippo. Representing and verifying intentions in telephony features using abstract data types. In *Feature Interactions in Telecommunications Systems III*, 1995.
- [47] J. Thistle, R. Malhame, H.-H. Hoang, and S. Lafortune. Feature interaction modeling, detection and resolution: A super supervisory control approach. In *Feature Int. in Tele. and Dist. Sys. IV*, pages 93–107, 1997.
- [48] S. Tsang and E. H. Magill. Behaviour based run-time feature interaction detection and resolution approaches for intelligent networks. In *Feature Interactions in Telecommunications and Distributed Systems IV*, 1997.
- [49] G. Utas. “A Pattern Language of Feature Interaction”. In *International Workshop on Feature Interactions in Telecommunications Systems V*, pages 98–114, 1998.
- [50] G. Utas. A pattern language of feature interactions. In *Feature Interactions in Telecommunications Systems V*, 1998.

- [51] M. Weiss, T. Gray, and A. Diaz. Experiences with a service environment for distributed multimedia applications. In *Feature Interactions in Telecommunications and Distributed Systems IV*, 1997.
- [52] P. Zave. Address translation in telecommunication features. *ACM Trans. Softw. Eng. Methodol.*, 13(1):1–36, 2004.
- [53] P. Zave, H. Goguen, and T. M. Smith. Component coordination: a telecommunication case study. *Computer Networks*, 45(5):645–664, 2004.
- [54] I. Zibman, C. Woolf, P. O’Reilly, L. Strickland, D. Willis, and J. Visser. “Minimizing Feature Interactions: An Architecture and Processing Model Approach”. In *International Workshop on Feature Interactions in Telecommunications Systems III*, pages 65–83, 1995.

Glossary

Abstract Address Zone

using IAT terminology, an address zone that is closer to the network is more abstract than an address zone nearer the subscriber. 15

Address Zones

the different locations that make up the call path. This includes locations such as telephones, user addresses, network routers, and PBX devices. 14

Applied Feature List (AFL)

a list containing the features currently applied in this call path. 101, 107, 108

Allow Criterion Violation List

a list of all feature orderings known to cause criterion violations but whose presence is deemed acceptable for inclusion in the optimal ordering list. 93

Allowable Criterion Violation List (AllowCritList)

the list of known feature ordering that can result in a criterion violation, but whose presence is deemed acceptable for inclusion in the Optimal Feature Ordering. Allowable criterion violation elements are due to the presence of a criterion limitation. 93, 95, 102, 109, 111, 120, 128, 130, 137, 187

Bound Feature

a feature such that a single instance of the feature exists for each user, and all calls involving the user are routed through this instance of the feature. 16

Call Path

an execution path that represents the progress of a call/call attempt, which contains the feature modules that are added one at a time during initiation the setup of the call. 12

Call Scenario

is a specific feature ordering together with a specific set of feature data that is used to simulate a call attempt. 93

Call Segment

a portion of the call path that can be linked to other call segments to form a complete call path[28].
93

Call State

represents the current execution state of each call attempt and the call attempt's effect on the model's environment. 93

Call Tree

a tree representing all possible call paths for a call segment given a specific ordering of features and a set of subscriber data. 93

Call Database (CallDB)

a list of information recorded by features in response to the current call attempt. All recorded values relating to a call will be lost when the call is torn down. 102, 104, 105, 109

Callee

the user who receives a call. 11

Caller

the user who initiates, or whose features initiate, a call. 11

Call Identifier (CallID)

a list of variables that together represent the current instance of a call. 101–104, 107, 108

Categorization Approach

an approach that separates features into categories as an initial step to determining a priority ordering for the full set of features. 1

Categorization

the process of grouping together items (e.g., features) based on a given set of properties (e.g., functionality). 3

Call List (CList)

a list of calls and their corresponding signals that are currently simulated in our modelled telephony environment. 101, 103, 104, 107, 108

Call Name (CN)

the name of the current call segment. 101, 108

Concrete Address Zone

using IAT terminology, an address zone that is closer to the subscriber is more concrete than an address zone nearer the network. 15

Constraint Violation List

a list of all feature orderings known to cause constraint violations. 93

Constraint-Violation List (ConVioList)

the list of known feature ordering that can result in a constraint violation. 93, 95, 103, 108, 111, 119, 120, 122, 128–130, 187

Category Representative Feature (CRF)

a feature whose design simulates the basic underlying behaviour of any feature in this category. 95, 101, 103, 112, 117, 126–130, 132–135, 139, 187

Criterion Violation History List (CritHis)

a list of all criterion violations that have occurred thus far in this call attempt. 101, 107, 108

Criterion-Violation List (CritVioList)

the list of known feature ordering that can result in a criterion violation. 93, 95, 103, 109, 111, 120, 122, 128–130, 187

Call Stage (CS)

a data structure representing the different stages of a call, each call stage instance contains a specific call stage and the current address information, taking the form CallStat(Region, Zone, Source, Target, Alias). 99–101, 107, 108

Detection

a technique that identifies feature sets that can interact when the features are combined in the system. 31

Device-specific Interface Module

a module associated with a device and that translates between protocol signals understood by protocol signals understood by features and the device signals issued and understood by the device. 9

Distributed Feature Composition (DFC)

is AT&T's pipe-and-filter architecture is a virtual architecture for telecommunications used for feature specification and composition, which supports feature-interaction management 2

Dynamic Data

records information that pertains to the structure of the call and stores information such as the composition of the call (e.g., a feature has been added, a voice connection has been established), state information for each feature in the call, and the set of signals found in the message queues for each feature. 22

Feature

is any add-on functionality that extends the basic service. A feature is made up of several feature modules that execute the feature's functionality. The term feature is used to refer to either a feature or its modules. 9

Feature Interaction

a feature interaction occurs when the presence of one feature in the system modifies the expected behaviour of another feature in the system. Feature interactions are often intentional and may be either desirable or undesirable. 1

Feature Transition Rule

a feature transition rule that simulates the behaviour of a feature upon receipt of any input signal. 93

Filtering

a technique used to reduce the number of subsets of features that need to be tested or analyzed for interactions. 30

Free Feature

a feature such that a new instance of the feature is created for each call. 16

Ideal Address Translation(IAT)

a methodology introduced by Zave in [52] to determine how features in different address zones should work together. 15

Input Signal (InSig)

a data structure containing the input signal and its direction. 101, 103, 104, 107, 108

Intra-category

the set of features located within a single category. 4

Modular Development

is used to separate complex systems into smaller, more manageable pieces called modules. The code within each module is written in isolation from the remainder of the system, and access to information in other modules is obtained through the modules' interfaces. 1

Module

a component that is part of a feature's implementation. 9

Network

the location in the call path, where the routing of the call through the switching system of the basic telephony system is represented. 12

Ordered Feature List (OFL)

an ordered list of the features that are used to form this call path. 101, 104, 107, 108

Violation-Free List (FreeVioList)

a list of all feature orderings that do not contain any constraint or criterion principles. 93, 95, 102, 108, 109, 111, 120, 122, 128, 130, 136, 189

Output Signal (OutSig)

a data structure containing the output signal and its direction. 104, 107, 108

Ports

the access point through which each feature communicate by sending and receiving signals. 18

Prevention

a technique avoid interactions by coordinating features such that interactions are prevented from occurring. 33

Principle Assertions

the assertions defining the required and acceptable system behaviour. 93

Principles of Proper System Behaviour

a set of principles that define acceptable (or unacceptable) behaviours in the global system. 4

Prioritize

a set of items (e.g., features) are ordered with respect to one another according to precedence. 1

Resolution

a technique that determines an acceptable integrated behaviour for a set of interacting features. 36

Service

is the core functionality of a system. 9

Signal

any inter-feature communication, which may be a message, an event, a method invocation, and so forth. 18

Signal Table

a table representing the history of signals travelling along a given call path. 93

Source Region

the caller's portion of the call path. 13

Segment Stage (SegStg)

the stage of the current call segment, can be one of the following: sApp, sStartCon, sConnected, vLink, sAnswer, sComplete. 99–101, 107, 108

Static Data

records information that persists beyond the lifetime of a call, such as feature data (e.g., who subscribes to which set of features) and system data (e.g., billing records). 22

Subcall

any continuous portion of the call path. 18

Feature Data Database (FeatData)

is a database that holds subscription information for all users in the system, including the list of features subscribed to by each subscriber and subscription information relating to each of those features. For example, when a subscriber uses a call blocking feature, then FeatData contains a list of numbers that this blocking feature uses to determine when a call should be blocked. 102, 104, 106, 109

Subscriber

the user who signs up for and pays for the feature, which may or may not be the user who invokes or is affected by the feature. 11

System Database (SysDB)

a list of information recorded by features to the main system database. All recorded values are permanently retained unless specifically removed or updated by a feature. 102, 104, 105

Target Region

the callee's portion of the call path. 13

Telephony Prolog Model (TP model)

our representation of the Telephony environment in Prolog 92, 93, 95, 97–109, 111, 112, 117, 119, 122, 123, 125, 126, 128, 130, 131, 137–139, 187

Transparent

an execution propagates the incoming signal unchanged to the next feature in the call path without changing the state of the feature's CFSM and with no changes to the database data. Such transparent

execution gives the appearance that this feature does not exist and that a direct connection exists between the feature module's neighbours. 22

Index

- CRF**, 92, 126
- ConVioList**
 - definition, 95
- CritVioList**
 - definition, 95
- TP model
 - definition, 92
- address zone, 14
 - abstract, 15
 - concrete, 15
- address zones, 14
- allowable criterion violation list, 93, 102, 111
 - also see* **AllowCritList**
- allowable ordering, 111, 120
- applied feature list, 101
 - also see* AFL
- call
 - established, 11
 - incoming call, 13
 - outgoing call, 13
- call database, 102
 - also see* CallDB
- call identifier, 101
 - also see* CallID
- call list, 101
 - also see* CList
- call name, 101
 - also see* CN
- call path, 12, 93
- call scenario
 - definition, 93
- call segment, 17
 - active, 100, 103
 - definition, 93
 - executing, 100
- call stage, 101
 - also see* CS
 - cAdd, 97
 - cAnswer, 99
 - cDone, 99
 - cEnd, 99
 - cValid, 97
 - cVLink, 99
 - netLink, 98
- call state, 93, 102
 - definition, 93
- call tree, 112
 - definition, 93
- callee, 11
- caller, 11
- categorization, 3
- category
 - categorization approach, 1
 - intra-category, 4
- category representative feature, 103
 - also see* CRF
- constraint-violation list, 93, 103, 111
 - also see* **ConVioList**
- criterion-violation history list, 101
 - also see* CritHis
- criterion-violation list, 93, 103, 111
 - also see* **CritVioList**
- data
 - dynamic data, 22
 - static data, 22

- dynamic data, 22
- static data, 22
- detection, 31
 - CHISEL, 32
 - formal methods, 32
 - Finite State Machine (FSM), 33
 - LOTOS, 33
 - requirements model, 32
 - SCF3TM/Sculptor, 32
- device-specific interface modules, 9
- Distributed Feature Composition, 9
- Distributed Feature Composition (DFC), 2
- feature, 9, 27, 104–106
 - bound feature, 16
 - free feature, 16
 - module, 9
 - transition rule, 93, 104
 - also see* transrule
 - definition, 93
- feature data database, 102
 - also see* FeatData
- feature interaction, 1, 2, 23, 27
 - shared-variable interaction, 27, 34, 37, 40
 - constraint-violation interaction, 28, 35, 37
 - data-modification interaction, 28, 35, 40
 - global-invariant interaction, 4, 28, 35
 - race-condition interaction, 30
 - reachability interaction, 29, 35
 - resource-contention interaction, 29, 35, 37
 - user/feature interaction, 29, 35
- feature privacy, 43
- filtering, 30
 - Use Case Maps (UCM), 31
- Ideal Address Translation, 15, 69
 - principles, 44, 69, 72, 153
- modular development, 1
- module, 10
 - device-specific interface, 11
 - feature, 10
 - bound feature, 17
 - free feature, 17
 - network, 12
- network, 12
- optimal ordering, 91
- optimization
 - constraint, 119, 134
 - criterion, 120, 134
 - pairwise, 122, 133
- ordered feature list, 101
 - also see* OFL
- ports, 18
- prevention, 33
 - advanced Intelligent Network (AIN), 35, 144
 - Distributed Feature Composition, 34
 - also see* Distributed Feature Composition
 - Session Initiation Protocol (SIP), 36, 144
- principle, 91
 - of proper system behaviour, 4
- assertion, 93
 - definition, 93
- constraint, 108
- criterion, 108
- definition, 108
- prioritization, 2, 3
- prioritize, 1
- Prolog, 91–119
 - definition, 92
- region
 - source region, 13
 - target region, 13, 14
- resolution, 36
 - agent-based architectures, 37
 - fuzzy logic, 38
 - negotiation, 38
 - arbitration, 38

- direct, 38
- indirect, 38
- priority, 39
- rollback, 42

segment stage, 101

- also see* SegStg
- definition, 99
- sAdd, 100
- sAnswer, 100
- sComplete, 100
- sConnected, 100
- sStartCon, 100
- sVLink, 100

service, 9, 27

signal, 18

- table
 - definition, 93, 94

signals

- DFC core set, 19
- feature-specific signals, 20

subcall, 18

subscriber, 11

system database, 102

- also see* SysDB

transparent, 22

transrule, 104–106

- code, 104
- semantics, 104

unsatisfiable, 120

violation-free list, 102, 111

- also see* **FreeVioList**, FreeVioList