Ordering Features by Category

P Ann Zimmer and Joanne M Atlee
David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

Abstract. Thousands of telephony features exist and combining them creates a multitude of feature interactions. Ordering these features is a costly process but is necessary to prevent undesirable interactions and ensure proper system behaviour. This paper introduces feature categorization as a means of significantly reducing the cost of determining an ideal set of feature orderings based on a set of principles used to identify acceptable category orderings. We conclude with a case study showing a savings of $1 - \frac{1}{10^{55}}\%$ over traditional methods. This paper also presents theorems that prove the resulting orderings determined by the category theory hold inside single and multiple address zones.

1. Introduction

Modern telecommunications systems are structured to enable the rapid development of new *features*. Each **feature** is an independent enhancement to the system that provides some new end-user functionality. Unfortunately, seemingly unrelated features may have hidden dependencies that cause them to interact with each other. As such, the naïve addition of a new feature may disrupt the correct workings of existing features. In general, a *feature interaction* occurs whenever the presence of one feature affects the behaviour of another feature [4]. As a classic example of an interaction among telephony features, a CallerID feature, which displays information about the origin of an incoming call, might use the callee's Personal Directory feature to lookup and display the *name* of the caller, if known, rather than his phone number. As can be seen from this example, feature interactions are not necessarily bad. The problem is that many interactions are unexpected and can result in undesired or unpredictable behaviour.

The key to ensuring interoperability among features lies in generic architectures, design rules, and protocols that constrain and control how features interact with each other. For example, a number of approaches, such as AT&T's Distributed Feature Composition (DFC)'s pipe-and-filter architecture [12], precedence rules for dispatching input events to features [28], call filters [6], and patterns [26] resolve interactions by serializing features' reactions to events. In each case, the system's architecture imposes a serial order on features' executions, such that when multiple features are enabled by an event or a call situation, the features respond sequentially. An important side effect of serial execution is that, as a feature executes, it may or may not leave the system in a state where subsequent features are still enabled. Consider what happens when a user subscribes to both Return Call and Voice Mail - Do Not Disturb. Both features react to incoming calls to their subscriber: Voice Mail asks the caller to leave a recorded message for the subscriber; whereas Return Call records the number of the incoming call, so that the subscriber can return the call at a later time. If the features are ordered such that Voice Mail reacts first, then the system automatically offers to record a message, and Return Call does not record the number of the incoming call. If instead Return Call is the first feature to react, then Return Call can record the number of the incoming call, then the call is presented to the Voice Mail feature and the caller gets the option of leaving a voice mail for the subscriber.

Of course, the success of these serialization-based approaches depends on our ability to devise a feature order such that sequential execution of the features results in desired system behaviour; and there are a large number of feature orders to consider. A set of *n* features has *n!* possible orderings, and many telephone switches have hundreds of features. Even if we consider only the problem of placing a new feature into an existing feature set, the existing set most likely induces a partial order on features, rather than a single serialization. Thus, ordering a new feature means examining the possible placements of that feature within every feature sequence that satisfies the partial order. As a result, the time to integrate a new feature into the system grows as the number of features increase. This affects not only development costs but also potential revenue streams and market share.

This paper presents a three-step process to serializing features: (1) each feature is categorized based on its goals and functionality, (2) the feature categories are serialized, and (3) the features within each category are serialized. The results of the various intra-category serializations (from step 3) can then be concatenated together according to the inter-category serialization (from step 2) to produce a single feature sequence.

The primary benefit of this approach is that it separates the task of resolving (via serialization) *expected* feature interactions from that of resolving *unexpected* interactions. Features within the same category are related to one another: they realize similar goals, or they perform similar functionality. It is not surprising that intra-category features interact (e.g., that two Redirect features want to reroute a call to different destinations). Moreover, because the goals and actions of such features are so similar, it would be difficult to determine automatically that one resolution is better than another. We believe that only a human expert can decide how to order the features within a category.

In contrast, features that belong to different categories are assumed to provide orthogonal functionality and are expected *not* to interact. We can take advantage of the fact that feature categories have distinct goals by formulating correctness criteria that express *principles* of proper system behaviour and feature behaviour. These principles can then be used to evaluate possible category orderings. In essence, we rank serializations of feature categories by the degrees to which they satisfy the correctness criteria. The evaluation and ranking is done automatically. The end result is a serialization of feature categories that effectively provides worked-out resolutions to unexpected interactions between categories of features.

A secondary benefit of this approach is that we reduce the overall cost of serializing features by decomposing the serialization problem into several smaller problems. As long as the number of feature categories is significantly smaller than the number of features, and the features are roughly distributed among the categories, the total number of feature orders that need to be analyzed, and the sizes of those orders, is significantly smaller. We show by way of a case study that large collections of telephone features can indeed be classified into a small set of feature categories. Although our experience in categorizing features is limited to telephony features, there is no reason to believe that our findings

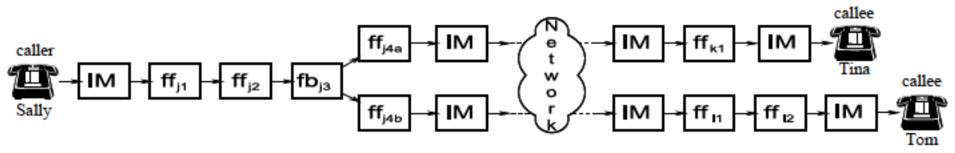


Figure 1. Multiple Active Calls

would not generalize to other domains where families of related features are common, such as in banking, insurance, and web services.

This paper is an extension of [30], which introduced the notion of feature categories and the serialization of categories based on principles of proper system behaviour. That paper also reported on a manual analysis of 11 feature categories and suggested a feature-category ordering that best adhered to the principles. In this paper, we extend that work by proving that feature-category ordering produces an optimal ordering, even in the presence of address-translation features, and by automating the serialization of feature categories. The remainder of this paper is organized as follows. Section 2 summarizes background terminology and AT&T's Distributed Feature Composition (DFC) architecture [12], on which we base our telephony model. Section 3 introduces the telephony case study that we use to evaluate our work. Section 4 describes the feature categories and principles originally presented in [30], and Section 5 presents arguments for why ordering features by categories works. In Section 6, we introduce our Prolog model for exhaustively exploring and evaluating possible orderings of feature categories. Specifically, the Prolog program takes as input a set of feature categories and a set of principles of correct behaviour, and it outputs a partial order of feature categories that best adheres to the principles. The program also produces a number of other outputs (lists of interactions, call trees, signal tables) that can help the analyst understand the cause of detected interactions. Section 7 describes optimizations that reduce the number of category orderings that the Prolog program considers. We revisit the case study in Section 8, review related work in Section 9, and conclude in Section 10.

2. Background and Terminology

In this section, we review the telephony model and terminology that is used throughout the rest of the paper. We base our telephony model on AT&T's Distributed Feature Composition (DFC) architecture because it supports modular feature development and because it resolves many feature interactions by serializing features' actions. Consequently, much of our terminology is based on publications on DFC.

A **feature** is some add-on functionality that enhances a user's telephone service. Example features include Call Waiting, Call Forwarding, Caller ID, and Voice Mail. A feature is implemented as one or more **modules**. We use the terms *feature* and *feature module* interchangeably. Each feature module is modelled as a communicating finite state machine (CFSM) that sends and receives signals through two or more communication **ports**. A **signal** may be any inter-feature communication, such as a message, an event, a method invocation, and so forth.

A **call path** (or simply **call**) is a sequence of interconnected features, as shown in Figure 1. The feature modules are connected via communication **channels** between their respective ports. An end point of a call is normally a **user** communicating via a **telephone device** (although it could be a feature acting on behalf of a user). We distinguish between the **caller**, the user who initiates or whose features initiate the call, and the **callee**, the user who receives or whose features receive the call. Users are identified by their telephone numbers, which we call **addresses**. An address may have a number of feature subscriptions associated with it. We use the term **subscriber** to refer to the user who registers for and pays for a feature subscription.

When a call is made from one address to another, all of the caller's and callee's subscribed features are included in the call path, regardless of whether the features are invoked during the call. The modules are added to the call path one at a time as the call is being set up. When all of the caller's subscribed features have been successfully added to the call path, the **network** routes the call through the switching system to the callee's address. Then the callee's features are added one-by-one into the call path. The callee's telephone rings, notifying the callee of the incoming call. The call becomes **established** once the callee answers the call and a voice connection exists between the call's end points. A **call attempt** is an incomplete call that is in the process of being set up; it consists of a partial sequence of feature modules. We also sometimes refer to a **subcall**, which is a subsequence of modules in a call or call attempt.[1]

Normally, a whenever a feature is included in the call, a new module instance for that feature is spawned and added to the call path. However, there are a few special multi-user features that coordinate calls between three or more users (e.g., Call Waiting, Three-Way Calling). These features need access to all signals being sent to and from the subscriber, regardless of which call they affect. Such features are modelled as **bound features**, for which there is one static instance of the feature module for each user. In Figure 1, fb_j3 is a bound feature that coordinates two calls: one between Sally and Tina, and another between Sally and Tom. The presence of a bound feature in a call changes the traditional linear structure of the call path, as the subcall between the bound feature and its subscriber, Sally, is shared by all calls involving Sally. From the bound feature, the call path branches out to each of the other users involved in Sally's calls.

Figure 2 shows another call in more detail. Using DFC terminology [12,27], we say that each call is partitioned into a **source region**, which comprises the caller's part of the call (e.g., the caller's end device and features) and a **target region**, which comprises the callee's part of the call. Similarly, a feature is designated a **source feature** (**target feature**) if the feature's functionality affects the caller's (callee's) call experience. We use the informal terms **outgoing call** and **incoming call** to refer to a call attempt in the context of its source region or target region, respectively. One might think that we could be efficient and restrict a call to just the caller's source features and the callee's target features. However, an end-user can simultaneously be a caller and a callee if he is involved in multiple calls. For example, a user Tom might initiate a call to Alice, and then via Call Waiting accept an incoming call from Dave. It is precisely because a user may be both a caller and callee that each call comprises *all* of the subscribed features of all the end users.

Features that redirect and forward calls to new addresses add to the complexity of call paths. A call may be routed through many different locations (e.g., telephones, user

[1]The above terminology differs from that used in DFC papers. In DCF terminology, a call or call attempt is called a *usage* [12] or a *request chain* [27], and the subcall from one module to the next module is called a *call* [12] or a *request* [27].

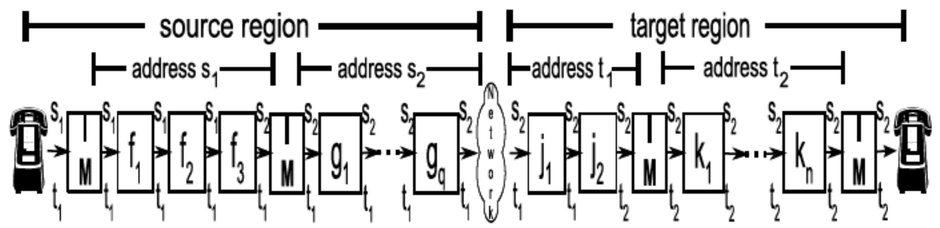


Figure 2. An established call, starting with an initial source address $s1$ and target address $t1$. The arrows between modules are annotated with the current source (above) and target (below) addresses. Feature f_3 redirects the call to a second source address $s2$. Feature j_2 redirects the call to a second target address $t2$.

addresses, trunk switches, network routers, PBX devices) before a connection is established. Each of these locations, can add features appropriate to the location. For example, device-specific features include Speed Dialing and Redial; service-provided features include Call Waiting, Call Forwarding, and Caller ID; and PBX features include Hunt Group (whereby calls to a single number are distributed among a group of phone lines, such as at call centres) and logging of call information for billing purposes.

To represent these different sets of features, we partition the source and target regions of a call path into **address zones**, where each address zone has its own feature set. For example, if the initial call request to one target address is redirected to another target address, then there will be multiple address zones in the target region. In such a case, the features $j_1, ..., j_m$ associated with the initial target address are included in the call path, but only up to the feature that, like j_2 in Figure 2, redirects the call to a new target address; after that, the features $k_1, ..., k_n$ associated with the new target address are incorporated into the call path. 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 $f_1, ..., f_p$ associated with the home number only up to the feature that, like f_3 in Figure 2, redirects the call; after that, the features $g_1, ..., g_q$ subscribed to by the work number are incorporated.[2] In our telephony model, we abstractly represent network/routing address zones as singleton **network modules**. This decision simplifies our call model without affecting our feature-ordering results.

Incorporating multiple address zones into a call path introduces an additional layer of complexity because each address zone contributes its own set of features that need to be ordered and included in the call path. Zave [27] introduces **Ideal Address Translation** (IAT), which is a set of design conventions that enable features in different address spaces to work together. IAT categorizes addresses according to how abstract the address value is, where the most concrete addresses identify specific physical telephone devices, a more abstract address might be a location (serviced perhaps by multiple telephone devices), an even more abstract address might be a personal address (regardless of where the person is or what telephone devices are nearby), and an even more abstract address might be a role-based address (e.g., the president of a corporation, whomever that might be). IAT places constraints on features' ability to redirect a call: for example, a source feature cannot redirect a call to a more concrete source address, and a target feature cannot

[2]If neighbouring address zones use different communication protocols, then an interface module can be added to the call path to translate the signals as they cross between the address zones.

redirect a call to a more abstract target address. The result is a call path that starts with the most concrete source address zone, passes through source address zones that are progressively more abstract, transitions to the most abstract target address zone, passes through target address zones that are progressively more concrete, and ends with the most concrete target address zone. This ordering of address zones prioritizes the zones so that the features closest to a subscriber have top priority over signals sent by the subscriber, and features closest to the network have top priority over signals received from a remote party. Zave proves that IAT guarantees a number of desirable properties of calls and users, such as privacy of address information, authenticity of users, and reversibility of calls [27].

IAT does not completely solve the problem of ordering features. It deals with how address zones should be ordered in a call path, but not how features should be ordered within an address zone. This paper focuses on the ordering of features within an address zone. In Section 5, we show not only that our orderings satisfy principles of correct feature behaviour within an address zone but that our orderings work together with IAT to produce a satisfactory ordering of features over the entire call path.

3. Telephony Case Study

As part of this work, we study the precedence ordering of a large collection of telephony features drawn from a variety of sources: the feature interaction benchmark [5], the second feature interaction contest [16], and from industrial documents, such as Nortel and 3Com reference guides to business services [1,19]. We consider all user-based features that would be associated with a single telephone address (i.e., a single subscriber), and exclude from the case study only data-services and call-center features. The result is a collection of 352 user-based telephony features to be categorized and ordered.

4. Categorizing and Prioritizing Features

In this section, we present the steps of our category-based approach to ordering features, drawing examples from the telephony case study. We first discuss the classification of features into distinct categories, according to their goals and functionality. We then identify principles for proper system behaviour. The principles are used to determine whether one feature ordering is more acceptable than another.

4.1. Feature Categories

The first step of the process is to cluster features into categories based on their goals and essential functionality. A feature's **goals** are user-defined or service-provider-defined objectives to be achieved by the feature. A feature's **functionality** is its behaviour as manifest by its possible executions. For example, we cluster together in a single *Redirect* category all features that re-route a call to another address without changing the intended participants of the call.

We clustered the case study's features into a total of 14 feature categories, listed in Figure 3. (As will be explained below, three entries in Figure 3 each represent two distinct categories.) The largest category is X, which contains over Y features. Most of

Authentication – a source (target) feature that verifies whether the caller (callee) is authorized to initiate (receive) a call from a particular source (target) address.

Blocking (S/T) – a source (target) feature that prevents the completion of an outgoing (incoming) call if the target (source) address is found on the feature's blocking list.

Filter – a target feature that selectively blocks or redirects incoming calls that are not meant for the callee.

Set Outcome – a target feature that asserts the outcome of a call attempt by issuing a signal, usually a *call-attempt-failed* signal, on behalf of the callee regardless of the callee's true call state.

Redirect (S/T) – a source (target) feature that re-routes a call attempt to another source (target) address without changing the intended call participants.

Delegate – a target feature that redirects an incoming call to an agent acting on the subscriber's behalf (e.g., a voice mail server or a secretary).

Redial (S/T) – a source or target feature that places a call to a previously logged address.

Alias – a source (target) feature that allows the user to employ an alias, such as a name or a Speed Dial code, to refer to a callee's (caller's) address.

Presentation – a source or target feature that presents information (via call display, ringtones, etc.) about the call to a user.

Multiplex – a source or target feature that allows the subscriber to be involved in multiple calls simultaneously (i.e., Conference Calling).

Billing – a source or target feature that records billing information for a subcall between adjacent address zones. Alternately, Billing features, like Collect Call, may change which user (address) is billed for a call.

Figure 3. The feature categories from the case study.

the categories are self-explanatory, although a couple are worth discussing in more detail, such as the difference between categories *Blocking* and *Filter*. *Filter* features handle misdirected or to-be-directed calls, whereas the *Blocking* features terminate undesired calls. Another important distinction is that *Filter* features may reveal callee information to help the caller complete her call appropriately. For example, a *Filter* feature can notify the caller when the person called has switched jobs, and can help direct the caller to either the original *person* called or to the new person who has taken over the *job*. The difference between *Delegate* and *Redirect* features is also rather subtle. Features in both of these categories reroute a call to a new address. The main difference is the reason for the rerouting: a *Delegate* feature reroutes the call to an *agent* of the callee (i.e., voice mail, assistant), whereas the *Redirect* feature is trying to reach the *callee* at an alternate location. The distinction is important because if the call is *redirected* to the callee, then the callee will expect to have access to all of his features regardless of his location; whereas if the call is *delegated* to another agent, there is no assumption of features subscribed to by the original callee. This distinction was identified during our original manual serialization of the case study features [30].

If source and target features have similar goals and functionality then we normally combine those features in the same category. However, there are a few categories whose goals and behaviours depend on whether they represent source-region or target-region features. These categories are designated in Figure 3 by the suffix *"(S/T)"*. For example, *Source Blocking* features prevent the caller from making certain outgoing calls (e.g., calls that incur long-distance charges), whereas *Target Blocking* features prevent certain incoming calls from reaching the subscriber. As another example, the goal of *Source Redial* features is to make it easier to repeatedly call the target of a previous call or call attempt, whereas the goal of *Target Redial* features is to facilitate callbacks to the callers of unsuccessful call attempts. As a special case, the *Redirect* category is also decomposed into separate source- and target-region categories because the *Target Redirect* features (only) must authenticate the callee. In general, *Authentication* features should always be located closest to the subscriber, so that the subscriber must verify his identify before he can access any of the rest of his features in the address zone. Whenever an incoming call is redirected, the *Target Redirect* feature is the last feature in the address zone to be added to the call path (i.e., closest to the subscriber), so it is responsible for authenticating the callee's access to that address zone.

Clustering is a somewhat iterative process, starting conservatively with just a few goal-based categories and splitting categories when needed. A category is typically split when it cannot be placed into a category sequence, but subcategories can be sequenced. For example, Redirect Source and Redirect Target features were originally together in the same category. The category was split when it became apparent that the features required both concrete addresses and aliasing information to make the appropriate call routing decision. Thus, the Redirect feature needs to occur after the Alias feature based on the direction of the incoming call signal, so the category needed to be split. Throughout the categorization process, it is important to maintain high cohesion among the features in each category and loose coupling between the different categories. Low coupling between categories reduces the risk of interactions between the categories, while high cohesion within each category ensures that during analysis a category can be represented by very few abstract features.

4.1.1. Multiple-Purpose Features

It is possible for a feature to have several purposes and thereby fall into more than one feature category. For example, Call Forwarding (CF) is designed to forward an incoming call to another address either automatically or as a failure treatment. When CF forwards a call to the *subscriber at an alternate location*, CF behaves as a *Redirect* feature; however, when CF forwards a call to an *agent*, such as an administrative assistant, then CF behaves as a *Delegate* feature. We recommend that such *Multiple Purpose* features be cloned and presented as distinct features, one to perform each purpose. In the above example, two CF feature clones are created: CF-Self as a Find-the-Subscriber feature, and CF-Delegate as a Delegate-Call-to-Agent feature.

Alternatively, a feature that performs multiple types of functionality may be placed in the feature category that matches the feature's more dominant purpose, as long as this placement does not violate any category-ordering restrictions imposed by the feature's other purposes. For example, every "pay-per-use" feature provides some end-user functionality plus some billing actions. In this case, "pay-per-use" features are categorized according to their end-user functionality. This categorization works because billing is modeled by sending call and feature charges directly to the billing database; the charges are commutative and associative, and thus are not affected by the order in which they are received by the database.

4.1.2. Special Cases

All of the feature categories discussed so far represent features that would be associated with the address (telephone number) of the subscriber and would be modules in the same address zone of a call. In addition, there are a few features whose classification defies the above categorization scheme, either because they are implemented within existing features or because they introduce new special address zones.

A *Disabling* feature stops another feature in the address zone from fulfilling its purpose. For example, Cancel CallWaiting (CCW) allows the subscriber to temporarily disable Call Waiting (CW) for the duration of a call. A *Disabling* feature is subscribable and offers a distinct service, but it is most easily modelled by extending the feature that it disables to respond to a feature-specific disabling signal. This extension of a feature does not affect its categorization or its serial ordering, because the fact that a feature can be disabled has no effect on its purpose or its interactions with other categories. Hence, *Disabling* is not a distinct feature category.

A *Remote-Control Invoking* feature allows the subscriber to give the remote user temporary access to an unsubscribed-to feature. For example, the caller can invoke 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 functionality associated with the feature. The Remote-Control Action module resides in a special address zone in the remote user's region of the call, and thus is not ordered with respect to the subscriber's features; we rely on Ideal Address Translation [27] to order this special address zone with respect to the call's other address zones. The Remote-Control Invoking module is the only part of the feature that resides in the subscriber's address zone, and it poses no serialization constraints.

A *Remote-Control Override* feature allows the subscriber to disable a remote user's feature. For example, the feature Make Set Busy Override (MSBO) can override the reported unavailability of the callee, if the unavailability is due to the callee's Make Set Busy (MSB) feature. A *Remote-Control Override* feature can be thought of as a combined *Remote-Control Invoking* and *Disabling* feature, where the remote-control command is a disabling signal and the remote action module is an extended feature module that responds to the disabling signal. Hence, each *Remote-Control Override* feature is composed of two modules: a Remote-Control Override module that resides in the subscriber's address zone and that sends a feature-specific disable signal, and an extended version of the remote-user's feature module, which resides in the remote user's address zone. As with the previous category, *Remote-Control Override* features pose no serialization constraints.

4.2. Principles of Proper System Behaviour

There are certain principles of proper system behaviour that must be upheld so that the system executes in the manner that the users expect. These principles reflect both system and feature requirements: the system requirements specify global properties of a correctly working system, and feature requirements specify properties that the users expect of correctly working features. Using these requirements, a set of hard and soft princi-

Abortion: – Calls made to or from numbers that appear on the caller's or callee's blocking list should be aborted.

Authorization: – A user's identity must be verified before the user can interact with any of his features. A call is not established until the users are authenticated.

Invoicing: – The cost of each subcall from one address zone to another is charged to some user.

Network: – When a call attempt is routed to a new address zone, if the new zone is denoted by an *alias*, the alias is resolved into a network *address* before the call attempt reaches the Network.

Figure 4. Hard principles from the case study

Accessibility: – All of the features associated with each end users' address(es) will be included in every established call. Each end user expects that her full set of features is accessible to her for the duration of the call.

Failure: – Any feature that is triggered by the receipt of a *call-attempt-failed* signal is ordered with respect to other features such that it will intercept all such failure signals before they are propagated to the next address zone in the call path.

Logging: – Information about all successful and unsuccessful calls should be recorded, with the exception of blocked or delegated calls. Blocked calls are treated as if they never occurred, and delegated calls are treated as if they had never involved the subscriber.

Personalization: – Aliases should be usable whenever they exist, so that human users can work as much as possible with aliases rather than network addresses. In particular, features that record network addresses or that initiate calls on behalf of the subscriber should also record and use aliases, to take advantage of changes in alias assignments.

Presentation: – When a presentation feature is subscribed to, the callee's end device will present information about each and every call that reaches the end device, but will not present information about calls that are blocked, redirected, or delegated before reaching the end device.

Concretization: – Any feature that affects call routing (i.e., change dialed number, redirect to another source region) requires a concrete address as well as any available alias information relating to that address. This allows either the concrete address or the alias information to be used to determine how the call should be routed (i.e., blocked, continued, forwarded).

Figure 5. Soft principles from the case study.

ples[3] are identified that express the expected behaviour of the basic telephony system, as well as properties imposed by feature categories. **Hard principles** express behaviour that must always hold, while **soft principles** express properties that should hold whenever possible. Figures 4 and 5 describe the hard and soft principles identified in our case study from Section 3.

Principles are correctness criteria, and they can be used to evaluate potential feature-category orderings on the basis of how well the resulting system behaviour adheres to these principles. Because hard principles are required to hold, a feature-category ordering must satisfy all hard principles to be acceptable. Soft principles, on the other hand, are properties to be optimized. Thus, an optimal feature-category ordering is one that satisfies all hard principles and violates a minimal number of soft principles.

[3]Hard and soft principles are referred to as constraint and criteria principles in [30] and [29].

Recall the feature interaction example given in the paper's introduction in which the ordering of features Voice Mail and Return Call determines whether Return Call is even activated. The Logging principle can be used to distinguish between acceptable and unacceptable orderings of the Delegate and Target Redial categories: by placing the Return Call before the Voice Mail feature the logging information about the incoming call is recorded.

The example in Section 1 shows of how features from the Delegate and Multiplex categories can interact and be correctly resolved using a variant of the Personalization principle. Below are some other examples of how the principles can be used to deduce the correct ordering(s) of pairs of feature categories:

Example 4.1 ***Source Redirect versus Alias***

A Source Redirect feature allows a caller to redirect an outgoing call through another source address zone (e.g., to redirect a call placed from home through an address at work, so that his work address is billed for the subcall to the target address). An Alias feature allows the user to employ an alias, such as a name or a code, instead of dialing a full address. If a caller dials an alias and the call attempt is redirected by the Source Redirect feature before the dialed alias is translated into an address by the Alias feature, then the Network principle is violated because the network does not know how to use the alias to route the call. Thus, Alias features must be added to the call path before Source Redirect features are added.

Example 4.2 ***Redial versus Set Outcome***

A Redial feature can be used to call the originator of the last incoming unanswered call. A Set Outcome feature can be used to reject all incoming calls. If an incoming call attempt reaches the Set Outcome feature and is rejected before it reaches the Redial feature, then the Redial feature never has a chance to log the call information so that the subscriber can later return the call. This is a violation of the Logging principle. Thus, Redial features must be added to the call path before Set Outcome features are added.

We have decided to only consider feature interactions that occur within the same region, since there are no defined set of accepted desirable behaviour for most interactions that occur between a feature in the target region and a feature in the source region of the same call. Thus, the above principles will be applied within a single call region to determine the ideal ordering of feature categories.

Using the feature categories and the principles for proper system behaviour we analyze how the system works when the feature categories are serialized. Our analysis generates a set of best orderings for feature categories, but does not consider how features should be individually ordered within each feature category.

4.3. Category Based Orderings

The principles of proper system behaviour are used to determine if one category ordering is more acceptable than another. To determine the acceptability of category orderings, we create a Category Representative Feature (**Category Representative Feature** (CRF)), a set of feature transitions rules, that represent the expected behaviour of each category. A CRFis designed to model the goals and functionality of its associated category. We assume that the application of a set of non-interfering features within a category is represented the category's CRF. This assumption means that the resolved goals of a set of features in a category is represented by a CRF, otherwise the features in the category form an interaction feature set.[4]

Using the principles and the CRFs, we apply the CRFs in a serial ordering and check for principle violations. A sequence of CRFs is considered an optimal ordering if no hard principle violations occur and the minimum number of soft principle are violated.

5. Correctness of Ordering Categories

Our theory thus far has used principles to evaluate and rank category orderings, applying these principles generates an optimal ordering of categories. In this section, we address how optimal category orderings can be used to find optimal feature orderings. We prove this theory, beginning with a proof of correctness with respect to features found in different categories and follow-up with a proof of correctness with respect to features found in different address zones. As previously mentioned, we do not consider feature interactions that occur between source and target features in the same call, because there are few principles that indicate how these regions should behave with respect to one another. Due to space restrictions, we provide only sketches of our proofs. More formal proofs, which rely on precise definitions of multiple concepts, are presented in [29].

Theorem: Correctness of Category Prioritization
Given an optimal category ordering, $O = [C_1 \ldots C_u]$, for a set of categories $\mathcal{C}$ of size u, any sequence of features that adheres to this category ordering is also an optimal ordering with respect to the principles used to derive O.

Proof sketch: Proof by contradiction
Let f be the smallest sequence of features $[f_1^\star \ldots f_u^\star]$, such that

- each $f_i^\star$ is a sequence of zero or more features from category $C_i \in \mathcal{C}$
- there exists an execution of the feature sequence f that violates some principle P
- there is no subsequence of features in f whose execution violates principle P

Case 1: f contains more than one feature from the same category C_x.
Given that no smaller set of features violates principle P, the violation depends on the presence of multiple features from the same category (C_x) in the presence of features from other categories.[5] We know from Section 4.3 that the application of multiple features within a category will either leave the category in a CRFrepresentative state or an interaction been the intra-category features has been detected. In the first case, the application of the features exit the sequence with the same behaviour as a single feature, however given that no smaller subsets of feature are know to cause this interaction, this is impossible. In the second case, the sequence of features within the category do not adhere to the expected CRFbehaviour for the category and an intra-category interaction is detected. Since intra-category interactions are outside the scope of the theory on cat-

[4]We would like to explore this assumption more in future work: by identifying features within a category that do not conform to the a CRF transition rule, we identify features at a higher chance for feature interactions.
[5]If only one category is involved, then this is clearly an intra-category interaction and outside the scope of the theory on category orderings.

egory orderings, this interaction has no effect on whether a feature ordering is optimal with respect to the category-based principles used to derive O.

Case 2: f contains at most one feature from each category in $\mathcal{C}$.
We construct the feature sequence f_g from f, by replacing each feature f_i in f with the Category Representative Feature (CRF) for f_i's category. Recall that a CRFmodels the goals and essential functionality of its associated category, and that principles reflect the expected behaviour of the feature categories. Thus, if f violates some principle P about the behaviour of categories, then its corresponding sequence of CRFs, g, also violates principle P. Given that f adheres to the optimal ordering O, we know that g is a subsequence of the CRFsequence that was used to evaluate ordering O.

Because O is an optimal ordering, we know also that O is either violation free or it violates a minimum number of soft principles:

Case 2a: O is violation free.
If O is violation free, then its sequence of CRFs does not violate principle P, which means that the subsequence g should also not violate P, which leads to a contradiction.

Case 2b: O is known to violate a minimum number of soft principles, excluding P.
If the sequence of CRFs used to evaluate ordering O does not violate principle P, then the subsequence g should also not violate P, which leads to a contradiction.

Case 2c: O is known to violate a minimum number of soft principles, including P.
$\mathcal{C}$ is an overly constrained set of feature categories. The violation of principle P is a known and accepted solution to resolve interactions $\mathcal{C}$'s categories. Thus, although feature ordering f violates P, it is still an optimal ordering with respect to the principles used to derive O. $\square$

5.1. Correctness Across Address Zone

In the work presented thus far, we have ordered categories within a single address zone, in this section, we expand our focus across address zones. As discussed, Ideal Address Translation (IAT) [27] is a theory used to order entire address zones with respect to other address zones in the call path. IAT also imposes an ordering on the features in the call path. We combine IAT and our principles of category ordering to order features, the results are the features in different address zones may not adhere to an optimal ordering (according to our principles) but the ordering is acceptable due to IAT orderings and principles.

IAT principles were developed by AT&T for DFC-based telephony systems to determine how features should behave in the presence of multiple address zones. These IAT principles work by considering the purpose of each address zone and how this purpose should effect when a feature is implemented and how to override default behaviours.

Our category-based approach generates the ordering for features within a single address zone, we combine address zones into our approach by nesting the feature categories inside each address zone, so that as each new address zone is added to the call path the features subscribed to by that address zone are added based on the order determined by our feature category prioritization.

Nesting feature categories within several different address zones can potentially introduce undesirable interactions. The undesirable interactions include the presence of features which are out of order with respect to the category orderings due to their presence in different address zones or due to the presence of multiple instances of the same feature in different address zones. Given these situations, the question arises: When are two features subscribed to in different address zones ordered correctly with respect to one another?

We claim that the nested combination of feature category orderings within multiple address zones results in an acceptable ordering that correctly resolves feature interactions based on the combined set of IAT principles and our proper-system behaviour principles, where IAT principles have priority over proper-system behaviour principles.

Theorem: 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 IAT principles, the resulting serial composition of all feature categories within all address zones inside a DFC-based architecture correctly resolves interactions with respect to our category-based principles and IAT principles.

Proof sketch: Proof by case analysis
The proof of this theorem is accomplished by decomposing the proof into different cases. We need to consider each category ordering that is known to cause an interaction within the single address zone and show that when present across address zones the interaction between the categories is acceptable. There are technically many cases to consider, however these the cases can be reduced to 4 types: optimal ordering is maintained; no proper-system behaviour principle is violated; a proper-system behaviour principle is violated and IAT design conventions indicate that ordering is correct; and a proper-system behaviour principle is violated and IAT design conventions do not resolve the interaction. A proof for each of these sub-cases will be sketched below. A full formal proof for this theorem can be found in [29].

Case 1: Optimal Ordering Maintained.
The ordering of the pair of feature categories in the different address zone matches the optimal ordering. The ordering of the feature categories is optimal and no further analysis is needed.

Case 2: Optimal Ordering Violated.
The ordering of the pair of feature categories in different address zones does not match the optimal ordering. The ordering is not optimal, hence further analysis is required.

Case 2a: No proper-system behaviour principle is violated.
Analysis does not detect any violations, hence ordering is optimal.

Case 2b: IAT design conventions indicate that ordering is correct.
One or more of the proper-system principles are violated. However, this violation is deemed acceptable due to IAT design conventions, which have precedence over proper-system principles. IAT design conventions state that for features in different address zone, the more abstract (concrete) category has priority with respect to incoming (outgoing) signals in the target (source) region. Since the category has precedence due to the placement of its address zone, it is expected to respond to the signal before features in other categories have the opportunity to respond, and hence this is an acceptable violation.

Case 2c: IAT design conventions do not resolve the interaction.
One or more of the proper-system principles are violated and this violation is not correctly resolved using the IAT design conventions. IAT design conventions do not resolve

all interactions that occur between different address zones, thus the IAT solution was extended to allow these special cases to be correctly resolved.[6] The interactions that fall into this category can be resolved using IAT design convention solution. □

6. Generating Feature-Category Orderings Automatically

The task of serializing a set of feature categories entails identifying a category ordering that satisfies all of the identified principles of proper system and feature behaviour. To do a thorough job, one would evaluate all possible feature-category orderings – a time-consuming and error prone activity, if done manually. Thus, we formulate this search problem as a Prolog program. The Prolog model evaluates all possible category orderings, simulates all possible call scenarios, and outputs the set of orderings that satisfy all of the principles. If no such ordering exists, then it outputs all orderings that satisfy all hard principles and violate the smallest number of soft principles.

6.1. General Design Overview

Prolog, short for Programming in Logic, is a declarative programming language based on predicate calculus [23]. A Prolog program comprises a database of facts, relations, and inference rules plus a set of queries over the database. The program executes by evaluating a query. The query is also expressed a relation, so finding a solution to the query means finding a set of variable values that cause the query, when instantiated with those values, to be a true statement about the database's contents. The Prolog engine uses resolution, backtracking, term rewriting, and pattern matching to find an answer to a query. If there is more than one solution to a query, then the Prolog engine can be configured to return all solutions.

We chose to model our serialization problem in Prolog because there is a clear correlation between our telephony model and the database of facts and relations that make up a Prolog program: 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 assertions. We refer to the encoding of our telephony model in Prolog as the **Telephony Prolog Model** (TP Model). A query in our TP Model is designed to return, for a given set of categories and principles, a list of orderings of feature categories that exhibit the fewest principle violations.

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 CRF's behaviour of reacting to input signals. Each principle is expressed as an assertion over variables representing the current state of a call or call attempt. Whenever an assertion evaluates to false, the corresponding principle is violated, indicating an undesired feature interaction.

The five main concepts used in simulating the execution of a call attempt in the TP Model are **call path**, **call segment**, **call state**, **feature transition rules**, and **principle assertions**. A **call path**, in a variation of the definition given in Section 2, is a sequence

[6]The extended IAT solution involves the use of a special signal sent between the address zones to override default behaviour of the conflicting features.

of CRFmodules, ordered according to some feature-category ordering. The TP Model simulates communication between the CRFmodules by mapping the output signals from one module to the input(s) of its neighbouring modules. If a call path includes a *Multiplex* CRF, then it is decomposed into several **call segments**, where each segment is delimited by an end device or a *Multiplex* CRF [13]. Call segments simplify the modelling when there are sections of the call path that are shared by several calls, as shown in Figure 1. A **call state** represents the current execution state of one call attempt and the associated call information in the database. One or more **feature transition rules** are defined for each feature and simulate the feature's reactions to input signals. **Principle assertions** represent feature interactions, or the absence of a desired feature interaction, in the TP Model. In Subsection 6.3 and 6.4, we discuss the latter two concepts in more detail.

The TP Model performs an exhaustive simulation of all possible feature-category orderings. A category ordering is combined with a specific set of feature data (e.g., a redirect address, Personal Directory data, Blocking list) to form a distinct **call scenario**. For each call scenario, all potential outcomes are generated, where each **call outcome** represents one possible execution path of a call. A call outcome depends not only on the call scenario, but also on input from the call's environment (e.g., whether the callee answers the 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 outcomes, and the TP Model explores all possibilities.

6.2. Modelling Abstractions

In this section, we identify and describe some of the data structures used to model our DFC-based telephony model in Prolog. The data structures below store **dynamic** or **static** data, where *static data* are information that persist after a call is torndown, whereas *dynamic data* are information that pertain to the current structure of a call and are removed when the call is terminated.

- **Call Database** (**CallDB**) contains dynamic call-specific information logged by CRFmodules in the call segments and call outcomes currently being explored. When a call segment (outcome) terminates, any related call information is removed from **CallDB**. Examples of call database information include the composition of the call path, the list of CRFmodules still to be appended to the call path (if the call path is still under construction), as well as feature-related data such as the caller ID presented to the callee and the dialled alias.
- **System Database** (**SysDB**) holds static database information, recorded or required by the CRFmodules and the main telephony system, that persist beyond the life of a call. Examples of system database information include billing data, the last number dialled, and call-blocking lists. **SysDB** information is permanently retained until explicitly removed or updated by a CRF.
- **Call List** (**CList**) is the list of all active and complete call segments that form the call paths for the calls currently being explored.
- **Call State** is the current execution state of the call outcome being explored. It includes the stage of the call (e.g., under construction, established, ending), the current signal being processed, and the current state of the call's environment. Although our approach refers to the call state as if it were a concrete entity, it is

```
     BlockS_trans1(CallState, InputSig)
     % Block call if target address of call attempt is on the blocking list
1      % Check preconditions
2      CallState.stage is under construction
3      CallState is in the source region
4      Source Block is the last module added to the call path
5      InputSig = setup
6      % Preconditions satisfied; check target address
7          target := CallState.target
8          subscriber := CallState.source
9          blocking list := SysDB(subscriber, BlockS_list)
10         if target ∈ blocking list
11         then % block the call
12             OutputPort := reverse
13             OutputSig := callBlocked
14             CallState.stage := ending
15         else % continue the call
16             OutputPort := forward
17             OutputSig := InputSig
18         endif
19     endif
```

Figure 6. Transition rule for Source Block CRF.

in fact a derived entity comprised of information distributed in **CallDB**, **SysDB**, and **CList** accessible via the call's **Call Identifier** (**CallID**).

6.3. Feature-Transition Rules

In the TP Model, each category is represented using a CRF. A CRF is encoded as a set of feature-transition rules that model the essential behaviour of a feature category with respect to the signals it receives.

Information extracted from the TP Model databases are tested against the preconditions of each transition rule to determine whether the rule is triggered in the current call state. A triggered transition rule can extract further details from a database, such as feature data values, to determine the appropriate actions to be applied by the rule. The actions of a transition rule can modify the call state, output signals to neighbouring features, and update information stored in the **CallDB** and **SysDB** databases.

An example feature transition rule is shown in Figure 6. Each feature transition rule is designed with a set of pre-conditions (lines 2-5): a feature is enable when all the preconditions are satisfied. If the feature is enable, then temporary variables are used to extract information from the database (lines 7-9) and this information is used to determine the feature's post-condition (lines 10-18). The post-conditions reflect the feature actions and requests in changes to the call state and call database.

Example 6.1 ***Source Blocking CRF***

In this example, we consider the Source Blocking transition rule that executes when the CRFis added to the call path's source region. This transition executes only if (1) the call path is under construction, (2) the part of the call path under construction is the source region, (3) the most recently added CRFis Source Blocking, and (4) the input signal to be processed is setup. These constraints are listed in lines 1-4 of Figure 6, and comprise the rule's preconditions.

```
    Network_Principle(CallState)
    % Check Network Principle
1       % Check preconditions
2       CallState.stage is linking
3       % Preconditions satisfied; check target address
4       CallState.target is not formatted as a network address
5       % process hard-assertion violation
6       hardVioDetected(CallState)
```

Figure 7. Two principle assertions.

On lines 6-9, the rule checks the ***System Database*** *(****SysDB****) to see if the subscriber blocks outgoing calls to the intended target address. If the call should be blocked, then an error message is sent back along the established call path and the stage of the call changes to ending (lines 11-13). Otherwise, the feature behaves as if it were transparent (i.e., as if the module had no effect on the call) and simply propagates the input signal on to the next (future) feature in the call path (lines 15-16). Another rule adds the next module to the call path.*

6.4. Principle Assertions

Whether a feature-category ordering is deemed acceptable depends on whether the call outcomes for all of its call scenarios satisfy the system principles (see Section 4.2). In the TP Model, principle violations are expressed as Prolog assertions. If an assertion holds in the TP Model, then a feature interaction can occur among the modelled feature categories.

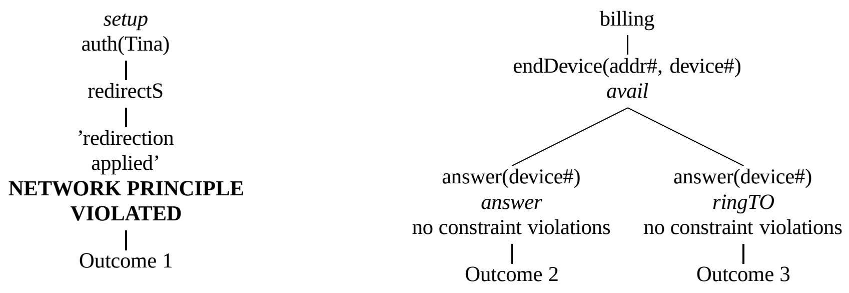
An example principle assertion is shown in Figure 7. Each principle assertion is designed with a set of pre-conditions (line 2) and a set of post-conditions (line 4). If the pre- and post-conditions are all satisfied, then the principle has been violated and we capture the interaction (line 6) as either a hard or soft violation.

A **hard violation** is a Prolog assertion that when true indicates a hard principle is violated. A hard principle expresses required or prohibited behaviour. So if the simulation reaches a call state that violates a hard principle, then the category ordering currently being explored is unacceptable and further exploration of the ordering is terminated. The explored prefix of the category ordering, which was sufficient to realize the violation, is added to the **Hard-Assertion Violation List** (**HardVioList**).

A **soft violation** is a Prolog assertion that when true indicates a soft principle is violated. Because a soft principle expresses desirable rather than required properties, the presence of a soft violation is not as severe as a hard violation and is treated more like a warning. If the simulation reaches a call state that violates a soft principle, the category ordering being explored is not immediately deemed unacceptable – if, in the end, no category ordering is violation free, then orderings that violate the least number of soft principle violations will be the optimal orderings. Thus, soft principle violations are noted by adding the explored prefix of the category ordering to the **Soft-Assertion Violation List** (**SoftVioList**), and exploration of the call scenario continues.

Example 6.2 ***Network Principle: Hard Violation***

The Network Principle can be violated only when a call attempt reaches the network before the dialed alias is translated into a network address. Thus, the corresponding



(a) Call Tree with Hard-Assertion Violation (b) Partial Call Tree

Figure 8. (a) A complete call tree that terminates after a hard-assertion violation is detected. (b) A small portion of the end of a call attempt, in which the incoming call reaches the end device, finds it available to accept the call, and explores both the *answer* and *ring-timeout* outcomes for this call.

Prolog assertion, outlined in Figure 7, need only be checked whenever the stage of the simulated call is linking, meaning that the call is being routed to a new address zone (line 2); and the property to check is that the target address is not formatted as a network address (line 4). If all of the assertion's statements are true, then the Prolog program reaches line 6, indicating that the principle has been violated.

6.5. Execution Model

This section briefly explores the different steps of simulating the TP Model: selecting the next feature transition rule to be applied, updating the call state based on the applied rule's actions, and testing the updated call state for principle violations. The process repeats until all possible execution paths have been explored.

1. Initialize the call state.
2. Select a unique ordering of feature categories.
   (a) Identify feature transition rules whose pre-conditions are enabled.
   (b) Select an enabled feature transition rule, instantiate possibility for any required but currently undetermined variations on the call scenario or call outcome (i.e., membership in blocking list, caller availability) and update the call state.
   (c) Test for principle violation.
   (d) Continue call exploration by: 1) continuing from step a) with the new call state until a principle violation is found or no unexplored enable features transition rules remain. Once option 1 is exhausted, backtrack to the most recent instance of b) with an unexplored feature transition rule and call scenario combination and continue call exploration.
3. Explore next possible ordering of feature categories and repeat above sub-steps.

6.6. Model Output

After simulating all of the possible call scenarios and identifying acceptable feature-category orderings, the TP Model simulation outputs:

1. **Violation-Free Orderings** (**VioFreeOrder**) - a list of feature orderings that are violation free.
2. **Optimal Orderings** (**OptimalOrders**) - this list is output only if **VioFreeOrder** is empty (i.e., there are no violation-free orderings). It lists the feature orderings that violate no hard principles and violate the least number of soft principles. These are the optimal orderings.
3. **SoftVioList** - a list of orderings that violate some hard principle
4. **HardVioList** - a list of feature orderings that violate some soft principle

In the above, **VioFreeOrder**, or **OptimalOrders**, is the simulation's recommendation for how to order feature categories to minimize undesirable interactions between inter-category features.

The latter two lists, of feature orderings that violate principles, are output for information only. To help feature designers to better understand why a particular ordering of feature categories violates a principle, the TP Model simulation can output at the user's discretion the set of *call trees* and corresponding *signal tables*. A **call tree** is created for each call scenario of each call segment, and represents the possible call outcomes that could execute in that scenario. Figure 8(a) shows a call tree whose path violates the Network principle, which is a hard-assertion. The tree begins with the receipt of the *setup* signal. The caller is authenticated and dials an alias code, *Tina*. The call then passes to a *redirectS* feature, which in this call scenario instantly redirects the call to another source region. When the redirection occurs, a violation of the Network principle is detected: the dialed alias was not translated into a network address before the call is routed to the new call region. Figure 8(b) shows a small portion of a call tree that has multiple outcomes: one for each possible callee response to an incoming call.

A **signal table** (not shown) can also be output for each call tree. The table displays all of the signals that are generated by the features in the call outcomes, the order in which the signals occur, the direction that they flow among the features in the call tree.

7. Optimizations

The previous section describes the TP Model simulation as if it searches all possible call outcomes. In this section, we introduce several optimizations that significantly reduce the search problem by avoiding exploration of orderings that contain a subordering known to result in a violation.

7.1. Optimizations For Hard Violations

A feature-category ordering that can result in a hard principle violation is **unacceptable**. Moreover, a larger ordering is unacceptable if includes a subsequence of categories that is unacceptable. The **Hard-Violation optimization method (HardVio)** is designed to avoid the simulation of feature-category orderings that contain an unacceptable subsequence. There are two aspects of this optimization: (1) Whenever a hard-assertion is detected (found to be true) during exploration of a category ordering, the categories explored before the violation, which may be a prefix of the category ordering under exploration, is rejected: this prefix is added to **HardVioList** and the simulation of all active call paths ends. Simulation resumes with the next category ordering to be explored.

```
softVioDetected(CallState)
% called when a soft violation is detected during simulation of CategoryOrdering
    if CallState.SoftVioCount == CallDB.MinViolations
        then % violation count exceeds minimum; ordering rejected
        add CallState.callpath to SoftVioList
        end simulation of CallState.ordering
    else % continue simulation
        inc(CallState.SoftVioCount)
        continue simulation of CallState.ordering
    endif

softVioOptimization(CallState)
% called when simulation of category ordering finishes
    if CallState.SoftVioCount < CallDB.MinViolations
    then % we have a new minimal number of violations
        CallDB.MinViolations = CallState.SoftVioCount
        forall entries $o \in$ OptimalOrders
            remove $o$ from OptimalOrders
            add $o$ to SoftVioList
        endforall
    endif
    add CallState.ordering to OptimalOrders
```

Figure 9. Two SoftVio optimization routines.

(2) Before the simulation of a new category ordering, the ordering is checked to see if it includes an unacceptable subsequence (listed in **HardVioList**). If so, the simulation rejects the ordering outright and instead starts exploring another ordering.

The optimization benefits primarily from avoiding the exploration of entire feature-category orderings. The reduction in search time depends on the number and sizes of the prefixes reported as being unacceptable: the smaller the prefix, the larger the number of full category orderings that can be rejected without examination. There is also some reduction in search time that comes from prematurely ending the simulation of a category ordering. The size of this reduction depends on what percentage of the ordering's search space remains unexplored when the violation is detected.

If every ordering of feature categories results in a hard violation, then there are no acceptable category orderings. In this case, the system is over constrained. System designers must either relax one of the hard principles (i.e., demote a hard principle to a soft principle) or must disallow some feature-category combinations.

7.2. Optimizations For Soft Violations

Soft violations are not as serious as hard violations, because soft assertions are considered to be *desirable* rather than *required* properties. As such, when a soft-assertion is detected (found to be true), the acceptability of the category ordering being explored is diminished but not necessarily rejected. The goal of the **Soft-Violation optimization method (SoftVio)** is to explore only category orderings that have a chance of being an optimal ordering. An optimal ordering is one that exhibits the *minimum* number of soft-principle violations. In the ideal case, an optimal ordering exhibits no violations.

The method works by keeping track of the minimum number of soft violations detected within any of the already-explored category orderings, keeping a list of the orderings that have only this number of violations (**OptimalOrders**), and keeping a list of the orderings that exceed the minimum number of soft violations (**SoftVioList**). There are three aspects of this optimization method. Figure 9 shows the first two aspects: (1) Whenever a soft violation is detected during simulation, the program increments the count of violations associated with the category ordering being explored; if the new count exceeds the minimum number of violations (lines 1-4 of softVioDetected), then the category ordering is rejected: simulation of all active call paths ends, and the categories explored (possibly a prefix of the category ordering) is added to **SoftVioList**. Simulation resumes with a new category ordering. (2) Whenever the exploration of a category ordering finishes successfully, its count of soft violations is compared to the current minimum number of soft violations known for any ordering. If the count is smaller, then it becomes the new known minimum number of soft violations (lines 1-8 of softVioOptimization). **OptimalOrders** all have higher than the (new) minimum number of violations, so its contents are moved to **SoftVioList**. The category ordering just explored becomes the first element of a new **OptimalOrders**. Simulation resumes with a new category ordering. (3) Before the simulation of a new category ordering, the ordering is checked to see if it includes a subsequence known to exceed the number of soft violations (listed in **SoftVioList**); if so, the simulation rejects the ordering outright and instead starts exploring another ordering.

7.3. Pairwise Optimizations

Most feature interactions are interactions between pairs of features. Thus, we can significantly reduce the search space by first analyzing all pairs of feature categories; populating **HardVioList** and **SoftVioList** with category pairs that exhibit hard- and soft-principles violations, respectively; and then exploring larger category orderings. The violations detected during the analysis of pairs of categories are used by the HardVio and SoftVio optimization methods to reduce the number of full-size category orderings that are explored. We refer to this search strategy as **Pairwise Optimization**.

Given a set of n feature categories, the cost of performing the pairwise analysis is quadratic: $n * (n-1)$ analyses of category pairs. But if a single violation is found during this analysis, then the number of full category orderings is reduced by half $n!/2$ analyses of n-category orderings are eliminated because exactly half of the orderings contain the subordering that exhibits the violation. The HardVio and SoftVio optimization methods reject these orderings before their simulation even starts. Each additional violation reduces again the number of full orderings that need to be simulated, usually by half. Thus for sets of feature categories of size $n > 3$, the cost of the pairwise analysis is small compared to the savings realized by avoiding the search of full-size orderings.

8. Evaluation

To evaluate how well our approach works for decomposing features into categories and to evaluate the effort of prioritizing these feature categories, we constructed a case study using home-based telephony features from several different sources. As shown in Table 1, we surveyed over 350 features taken from sources including the feature interaction benchmark [5], the second feature interaction contest [16], and from industry sources, such as Nortel and 3Com [1,19].

Table 1. Total Feature Count broken down by Category for different Sources.

		Source						
		Nortel [19]	3Com [1]	Centrex [20]	Centrex Plus [21]	Bellcore [5]	FIW Contest [16]	Total
Count	Original Features	58	59	35	171	17	12	352
	Total Features	64	62	41	199	17	12	395
	Percent Uncategorized	14%	23%	14%	32%	6%	0%	
Category	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 (S,T)	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 (S,T)	7 (2,5)	5 (4,1)	7 (4,3)	2 (1,1)	2 (1,1)	1 (1,0)	24
	Redirect (S,T)	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

Of the initial 352 features, we were able to categorize 268. The 84 uncategorized features fell into three categories: emergency features; end-device features, which are implemented in the physical end device, such as hands-free dialing [19]; and administrative features, such as the ability to add and remove features from a user's subscription list. We excluded emergency features from our study because they must be manually evaluated and serialized to ensure that they adhere to the necessary local and national regulations. The end-device and administrative features are not covered because they reside in their own special address zones, and this paper is concerned with the ordering of end-user features. Our category-based approach could be used to find suitable orderings of the end-device and administrative features within their respective address zones; this may require the identification of new categories and principles for these special address zones.

Table 1 shows how the remaining features are clustered. Features from different sources often have overlapping functionalities or descriptions (e.g., every source describes at least one Call Waiting Feature), although each feature has a different implementation. Note that the total number of features is higher than 268 because many features have multiple goals and are thus cloned into two or more features, as described in Section 4.1.1. For example, some features that redirect call attempts can be both *Redirect* and *Delegate* features, and some features that block call attempts can be *Filter* features as well as *Blocking* features.

Altogether, this clustering results in 14 feature categories. This list of categories is not promoted as being complete. In fact, it is not possible to create a complete list, as new features are continuously being added to the base system. Moreover, determining categories is a subjective process, and there may be other acceptable ways of clustering features into categories. The effect of decomposing feature categories coarsely is that there may be fewer (or no) category orderings are interaction free. The effect of decomposing feature categories finely is reduced savings from using our category-based approach.

In the first case study, all possible call scenarios involving two users were simulated. In the second case study, the call scenarios were expanded to include three users, where after a traditional two user connection is established, a *Multiplex* feature (i.e., three way calling, call waiting) is used to join a third party into the existing connection.

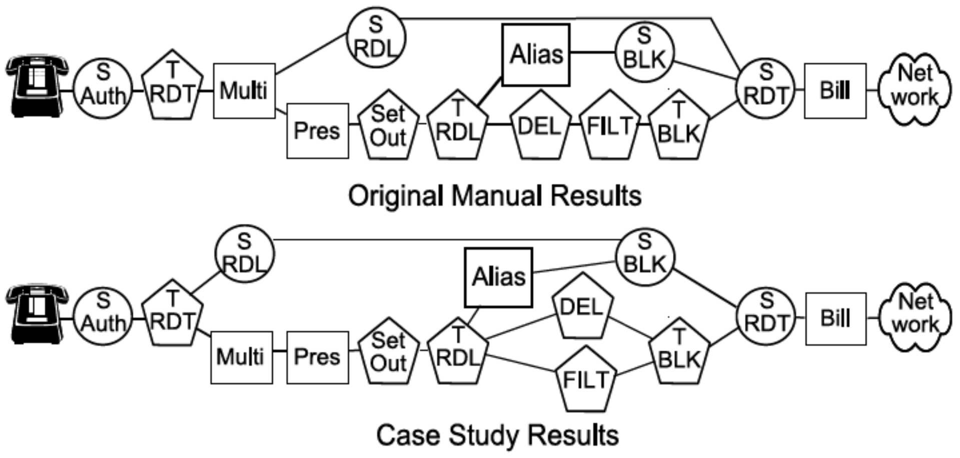
We compared the results of our case studies against the results of our manual analysis [30]. Figure 10 shows both the original manual analysis results, the updated manual analysis results and the partial ordering formed by merging the optimal orderings from our case studies. The updated manual analysis results were modified after discovering the need for extra principles while implementing the TP Model. Note that the updated manual ordering and the case study-based ordering are very similar with only the interaction between the *Delegate* and *Filter* categories not being found by the Prolog analysis. This ordering is not found because the restriction used in the manual ordering allows for the fact that calls to the subscriber should only be delegated by the *Delegated* feature, if the call is intended for the subscriber as determined by the *Filter* feature. While this is a good reason for ordering these features, no principle was recreated to satisfy this restriction (it could be added) hence the ordering was not found.

8.1. Performance Analysis

To evaluate the TP Model, we used the 14 features categories identified by categorizing the features found in the different sources identified in Table 1. The results in this section discuss the optimizations resulting from running TP Model with the following simulations:

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.

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. 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 [19] (third column of Table



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

Figure 10. The top figure shows the original partial ordering identified during our manual analysis and as presented in [30]. The middle figure shows the updated manual analysis after adjusting the ordering based on the new principles discovered during our implementation of the Prolog model. The bottom figure shows the partial ordering determined using the results output by our Prolog Model after running both case studies.

1) for comparison purposes, so our "full set of" features is of size 58 and the traditional cost is $58! \approx 2.3 * 10^{78}$.

This traditional ordering approach has some inherent limitations and is never fully explored in practice. When using this traditional approach, or close variants of this approach, the system designer will stop all exploration of possible call paths once one acceptable feature ordering is found. This means that when a new feature is added to the system, if this feature does not fit in the existing acceptable feature ordering, then the system designer must start again with the comparison of full feature orderings. This is a direct contrast to our method, where all successful partial category orderings are determined and insertion of a new feature will usually only need comparison against other features within its category; unless the new feature results in the creation of a new category. When a new category is created, the comparison needs to be explored first the pairwise level (new category against all existing categories) and then with respect to any acceptable full partial orderings not eliminated by the pairwise analysis. Another advantage of our method over the traditional approach is that it separates expected interactions (intra-category) from unexpected interactions (cross-category) and allows the system designer to focus on a small set when resolving the expected interactions, resulting in fewer comparisons of full orderings and less work for the system designer.

In practice, the traditional method is more likely carried out as a set of pairwise feature comparisons followed by full category analysis until one acceptable full ordering is found. For our analysis, this would result in comparing 58 features at a cost of $\frac{n!}{(n-2)!}$, where n is the number of features plus the additional cost to determine one acceptable full feature ordering. If we assume the system designer is lucky and finds an acceptable ordering quickly then the cost of the full feature ordering analysis is significantly smaller than the pairwise analysis and we use $3.3 * 10^4$ as our cost for the modified traditional approach.

To compute the cost of our category approach; we determine the cost to categorize the features plus the cost of the pairwise analysis of the 12 categories, followed by the cost to analysis the remaining acceptable full CRFsets and finally the cost of analyzing the features within each category. For the intra-category feature orderings, we use the same assumptions as the modified traditional approach and the system designer stops looking after the first acceptable intra-category ordering is found. In our case study, the number of full CRF's analyzed was reduced from 12! to 399 after the pairwise analysis eliminated many potential orderings. Thus our cost to analysis is:

$$\begin{aligned}
&=\text{pairwise CRFs } + \text{full CRFs } + \text{cost to categorize features} + \text{feature orderings}\\
&=12!/(12-2)! \; + 399 \; + 58 \; + \sum_{i=1}^{12} insignificant\\
&=132 + 399 + \approx 58 + \approx insignificant\\
&= \approx 5.9 * 10^2
\end{aligned} \tag{1}$$

This is a savings of over 80%: a significant reduction in cost. For larger feature sets the savings would be significantly higher. The cost to explore the feature pairs in the modified traditional approach increases dramatically with each new feature added, but has not affect on the cost of our categorization approach, so long as not no categories are added.

9. Related Work

Previous work on the using priorities to resolve feature interactions has studied partial and full priority orderings as means to resolve feature interactions. Below, we explore various methodologies that determine the correct priority orderings by considering architectural design, run-time evaluations, design patterns, etc.

Nejati et al work with design patterns, in [18], is the most similar to our approach. Their technique similarly compares features in a DFC-based architecture to determine feature orderings that do not violate a set of "safety" properties. The approach uses design patterns to represent features, which are composed to represent feature composition and the results are tested for violation of safety properties. Nejati et al work formalizes the notation that the presence of a pairwise interaction is significant reason to reject any larger ordering containing the feature pair in the same relative ordering.[7] The notable differences between our techniques are 1) the feature evaluation is limited to a single address zone, 2) only features that are "different" can be evaluated, otherwise the analysis results are not useful 3) only a small case study of six features was performed and 4)

[7]This result is proven through the existence of transparent feature behaviour, which is also modelled by each of the feature categories in our analysis.

while the safety properties used to identify invalid orderings are similar to our set of hard-assertions, they have no method of represented soft-assertions, especially if a set of mutually-exclusive soft-assertions exists.

Tsang and Magill's technique in [25] 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 difference between Tsang and Magill's technique and our approach are 1) the concept of feature privacy (only observable actions are used to detect interactions, to allow combining features from different service providers) and 2) their technique was only tested on feature sets of at most size four, while our category-based approach is capable of ordering much larger sets.

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, such as AT&T's Distributed Feature Composition (DFC) architecture for telephony systems [12], these architectures are designed to work in combination with a priority-based resolution scheme. In other architectures, the architectural design itself imposes a priority scheme. By redesigning the call architecture into a call-processing model that broke a call into different roles and subroles inside which features are applied, Cattrall et al. [6] separated users from end devices and were able to avoid/correctly resolve interactions as roles and subroles asserting their priority to control the call and determine which features executed during the call attempt. A role 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). This concept of roles and subroles is closely related to the Ideal Address Translation principles [27] that we use in this paper 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. As another architectural example, Zibman et al. [28] 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. In Elfe et al. [10], feature priorities are determined 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. Nakamure and Tsuboi [17] 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.

Other examples of design-time solutions involve the use of Standard Transition Rule (STR) and Supervisory Control Theory (SCT) to help prioritize features. Harada et al. [11] and Kawauchi and Ohta [14] use STR in their work. Harada et al model features 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 when the implementation of the feature's behaviour results in contradicting transitions (i.e., they are both able to execute but have conflicting results). Resolution of the interaction is performed by the designer who can prioritize the conflicting transitions, or use another technique to resolve the interaction. Kawauchi and Ohta [14] 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.

In the area of SCT, Thistle et al. [24] compose supervisors 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. Another example prioritization in SCT is Chen et al.'s [7], 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 last two algorithms focus on designing the priority function to give the dominant supervisor priority over the other supervisor whenever a conflict occurs.

10. Conclusion

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 [2,8,9,15,3,22]. In this paper, we explore methods to reduce the cost of prioritizing a set of features for use in priority-based techniques for resolving feature interactions.

The categorization of features is a critical step that must be carefully considered as it affects the success of our category-based approach. Once the categories are selected, the principles for proper system behaviour are created, such that the principles 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. The principles presented in Section 4.2 represent the standard set of subscriber-based features.

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. We showed that the automatically generated partial-ordering 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 and the results of our manual analysis were updated to reflect these two new principles. 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, showed the cost was reduced to less than $\frac{1}{10^{55}}\%$ of the traditional cost. Given this significant reduction in the cost and the ability of our model to accurately reproduce the manually identified partial-ordering, we can confidently argue that our category-based approach was successful.

References

[1] 3com Techincal Guide. *IP Telephony Jargon Buster and Glossary*. http://www.3com.com/voip/assets/3com_200251-003.pdf.

[2] L. G. Bouma and H. Velthuijsen, editors. *Feature Interactions in Telecommunications Systems.* IOS Press, 1994.

[3] M. Calder and E. Magill, editors. *Feature Interactions in Telecommunications and Software Systems VI.* IOS Press, 2000.

[4] E. Cameron, N. Griffeth, Y. Lin, and H. Velthuijsen. "Definitions of Services, Features, and Feature Interactions", December 1992. Bellcore Memorandum for Discussion, presented at theInternational Workshop on Feature Interactions in TelecommunicationsSoftware Systems.

[5] 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.

[6] 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.

[7] 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.

[8] K. Cheng and T. Ohta, editors. *Feature Interactions in Telecommunications Systems III.* IOS Press, 1995.

[9] P. Dini, R. Boutaba, and L. Logrippo, editors. *Feature Interactions in Telecommunications and Distributed Systems IV.* IOS Press, 1997.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] K. Kimbler and L. G. Bouma, editors. *Feature Interactions in Telecommunications and Software Systems V.* IOS Press, 1998.

[16] 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.

[17] 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.

[18] S. Nejati, M. Sabetzadeh, M. Chechik, S. Uchitel, and P. Zave. Towards compositional synthesis of evolving systems. In *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 285–296, New York, NY, USA, 2008. ACM.

[19] Nortel Networks. *Centrex Feature Library*. www.nortelnetworks.com/products/01/centrex.

[20] Nortel Networks. *Centrex Feature Library - Glossary*. www.nortelnetworks.com/products/01/centrex/library/list.html.

[21] Nortel Networks. *Centrex Feature Library - Voice Features*. www.nortelnetworks.com/products/01/centrex/library/voice/index.html.

[22] S. Reiff-Marganiec and M. Ryan, editors. *Feature Interactions in Telecommunications and Software Systems VIII.* IOS Press, 2005.

[23] R. W. Sebesta. *Concepts of Programming Languages*. Addison-Wesley, third edition, 1996.

[24] 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.

[25] S. Tsang and E. H. Magill. Behaviuor based run-time feature interaction detection and resolution approaches for intelligent networks. In *Feature Interactions in Telecommunications and Distributed Systems IV*, 1997.

[26] G. Utas. "A Pattern Language of Feature Interaction". In *International Workshop on Feature Interactions in Telecommunications Systems V*, pages 98–114, 1998.

[27] P. Zave. Address translation in telecommunication features. *ACM Trans. Softw. Eng. Methodol.*, 13(1):1–36, 2004.

[28] 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.

[29] P. A. Zimmer. *Prioritizing Features Through Categorization: An Approach to Resolving Feature Interactions*. PhD thesis, University of Waterloo, 2007.

[30] P. A. Zimmer and J. M. Atlee. Categorizing and prioritizing telephony features. In *Feature Interactions in Telecommunications and Software Systems VIII*, 2005.