Distributed Feature Composition: A Virtual Architecture for Telecommunications Services

Michael Jackson, *Member*, *IEEE Computer Society*,
and Pamela Zave, *Member*, *IEEE Computer Society*

Abstract—Distributed Feature Composition (DFC) is a new technology for feature specification and composition, based on a virtual architecture offering benefits analogous to those of a pipe-and-filter architecture. In the DFC architecture, customer calls are processed by dynamically assembled configurations of filter-like components: each component implements an applicable feature, and communicates with its neighbors by featureless internal calls that are connected by the underlying architectural substrate.

Index Terms—Feature interaction, feature specification, feature composition, architecture, pipe-and-filter.

1 Introduction

The feature-interaction problem [6], [14], [24] arises from the incremental, feature by feature, extension of telecommunications system functionality. As new features—especially call-processing features—are added, it becomes increasingly difficult to manage the behavioral complexity of the features and their interactions. Redesign of old features to fit smoothly with the new features is scarcely ever a practical option. Eventually the resulting complexity damages the quality and productivity of all phases of software development. The proceedings of three recent workshops provide a representative sample of research on the feature-interaction problem [3], [7], [9].

This paper introduces a technology, *distributed feature composition* (*DFC*), for managing the feature-interaction problem. The heart of DFC is a virtual architecture for telecommunications systems in which a feature corresponds to a component type (a few features correspond to two component types—see Section 4.7); each customer call is handled by building a configuration of instances of these components, according to the features to be applied to that particular call. The feature component instances communicate by featureless internal calls that are connected by the underlying architectural substrate. A new feature is specified by describing its corresponding component type (or, occasionally, two component types) and the rules for including the component instances into configurations.

The primary characteristic of the DFC architecture is that each feature is implemented by one or two component types, and each external call is processed by a dynamically assembled configuration of components and featureless, two-port internal calls. The resulting configuration is analogous to an assembly of pipes and filters, and has the typical advantages of the pipe-and-filter architectural style: 1) feature components are independent, they do not share state, 2) they do not know or depend on which other feature components are at the other ends of their calls (pipes), 3) they behave compositionally, and 4) the set of them is easily enhanced [12]. These characteristics contribute greatly to the power of the architecture to manage feature interactions. DFC is a virtual architecture, and offers many possibilities of convenient mapping to typical physical architectures.

Section 2 gives a brief overview of the DFC architecture, to convey an intuitive understanding of how it works and how it addresses the feature interaction problem. Section 3 gives a more detailed and formal description, and Section 4 discusses the specification of various features in a DFC setting. In Section 5 we present a summary of the conclusions we draw from our development and study of DFC, and of the contribution that we believe DFC can make to addressing the feature interaction problem.

We have made various simplifying assumptions to avoid complicating our presentation of DFC. We believe that these simplifications are immaterial to the applicability of DFC to a fully realistic context. They are briefly described in Section 5.

2 Overview of the DFC Architecture

The fundamental idea is to treat features as independent components—which we call *boxes*—through which calls are routed from caller to callee. The routing of a call reflects the features to be applied to it.

We regard the system as having a *virtual switch* that serves its customers through *line interface* (*LI*) *boxes* at its periphery; features are provided by *feature boxes* (*FB*s), appropriately interposed on the path between a calling and a callee customer. In general, boxes are not shared among calls: two concurrent collect calls will require two collect feature boxes—that is, two distinct instances of the same *feature box type*. All communication between boxes takes the form of *featureless internal calls* connected by the virtual switch at the command of its embedded *router*. The router and switch, together with the necessary voice and signaling paths, form the substrate of the architecture, routing and connecting these internal calls from box to box.

• *M. Jackson and P. Zave are with AT&T Laboratories–Research, 180 Park Ave., Florham Park, NJ 07932. M. Jackson is an independent consultant in London UK. E-mail: mj@doc.ic.ac.uk; pamela@research.att.com.*

Manuscript received 27 Sept. 1997; revised 21 Mar. 1998.
Recommended for acceptance by Y.-J. Lin.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 107043.

2.1 Usages

We use the simple term *call* chiefly to denote these internal featureless calls. Episodes of customer service, usually referred to as calls, will here be referred to as *customer calls* or *usages*. We will chiefly use the terms *caller* and *callee* to denote the boxes placing and accepting internal *calls*. Where there is no ambiguity we will sometimes use *call*, *caller*, and *callee* for a customer call and its originating and terminating customers.

Fig. 1 shows a snapshot of one usage in the DFC architecture in a state in which two customers are talking.

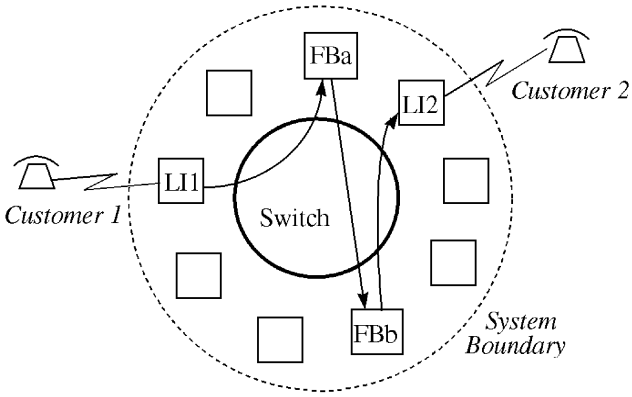


Fig. 1. A usage in the DFC architecture.

The customers' telephones are connected to the system at the line interface boxes LI1 and LI2. Two features have been applied to this usage, provided by the feature boxes Fba and FBb. The arrows in the diagram show the routing of calls and the directions in which they were placed. Customer 1's original request was routed as a call from LI1 to FBa; FBa then placed a call that was routed to FBb; FBb placed a call that was finally routed to Customer 2's line interface box LI2. The switch is, therefore, carrying three internal calls in providing the connection between Customer 1 and Customer 2.

Each call is carried by a two-way voice channel and two signaling channels, one in each direction. To complete the connection between LI1 and LI2, the feature boxes FBa and FBb join the voice and signaling channels of the calls in which they are participating. In general, a feature box has full control over the calls to which it is connected. It can coordinate their voice channels in any way desired, freely joining and separating their voice signals, playing announcements and tones, recording speech, and monitoring the in-band signals. Similarly, it can control the out-of-band signaling of its calls by suppressing or passing on messages and by introducing new messages of its own. The order of feature boxes in a usage is, therefore, significant. The behavior of each feature box is independent of the behavior of other boxes; but the effect of its behavior will depend on its position in the usage. Feature box precedence is an important aspect of feature specification in DFC.

A feature box is configured into each usage in which it may possibly be needed. For example, FBa may be an Originating Call Screening (OCS) box, and FBb a Call Forwarding on Busy (CFB) box, configured into the usage by the DFC router simply because Customer 1 subscribes to OCS and Customer 2 to CFB. If the number dialed by Customer 1 is not in the provisioned screening list, the OCS feature box FBa will behave transparently, the usage proceeding as if the box were not present; similarly, if Customer 2 is not busy, the CFB box FBb will behave transparently. This potentially transparent behavior of feature boxes is an important factor contributing to their independence and to the freedom with which they can be combined and inserted into usages.

When a feature box is actively providing its feature service by nontransparent behavior, it does so without relying on other boxes. For example, if the OCS box finds the number dialed by Customer 1 in the provisioned screening list, the OCS box will not place an outgoing call; instead it may send an *unobtainable* message back to the customer. If the CFB box detects that Customer 2 is busy, it tears down its outgoing call and places another call to the directory number (DN) specified by Customer 2 for such forwarded calls.

2.2 Call Routing

The routing of calls from box to box within the system is the responsibility of the router embedded in the DFC switch. The first internal call in a usage is typically placed by the originating customer's line interface box: the box sends the switch a *setup message* containing an empty *routing list* and four fields: 1) a *source* field (whose value is the DN of the originating customer), 2) a *dialed-string* field, 3) an empty *target* field, and 4) a *command* field with an associated *modifier* (which in this case would specify that a new routing list was to be constructed). The router may assign a value to the target field from the dialed string; it then uses the four field values, together with information about customer subscriptions and features, to construct a routing list for the usage. This list is then inserted into the setup message for dispatch to the switch and onward transmission.

To dispatch a call, the router truncates the routing list by removing its head entry, and requests the switch to connect the call to the box specified in that entry. A box that receives a call receives the truncated setup message; in many cases, once that call has been set up, the box will then place a second call whose setup message is a copy of the first. When the switch receives this second setup message, the router again truncates the routing list, and the switch connects the second call to the appropriate box. In this way a chain of calls is formed until eventually the routing list is empty and the last call is routed to the line interface box of its final target.

2.3 Configuring a Usage by Zones

Features are selected by the router in three *zones*: "Source," "Dialed," and "Target." Intuitively, features in the Source zone are applied to all calls made by the Source caller: for example, the SpeedCalling (SC) feature is applied to every call made by its subscriber. Similarly, features in the Target zone are applied to all calls directed to the Target callee: for example, the Call Forwarding on Busy (CFB) feature is applied because the callee subscribes to it. Features in the Dialed zone are applied according to the string dialed by the caller: for example, the prefix '0' causes the Collect feature to be applied. Roughly, the three zones correspond to three obvious subchains in the construction of a usage, as shown in Fig. 2.

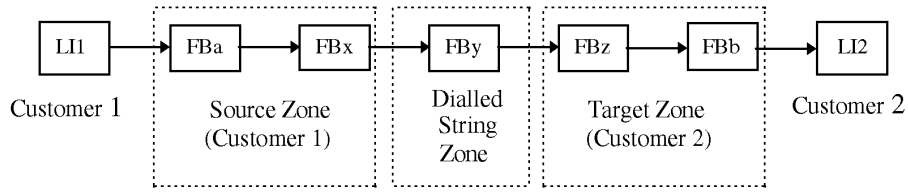


Fig. 2. Feature zones in a usage.

The applicable feature boxes are specified in the routing list field of the setup message in zone order. The routing list specifies first the appropriate source zone feature boxes, such as SC and OCS; then the appropriate dialed zone feature boxes, such as Collect; and finally the appropriate target zone callee feature boxes, such as TCS and CFB. Within each of the three zones, the feature boxes appear in the precedence order given by their feature specifications.

Although this arrangement of features in zones may at first sight seem restrictive, we know of no case where it precludes a desirable behavior or obscures or complicates the desired application of features to a usage.

2.4 Routing Data

The DFC router uses three sets of global data: *subscription*, *specification*, and *configuration* data. All of this data takes the form of relations. Subscription and specification data (with one exception) is partitioned by feature, each relational tuple being associated with just one feature.

Subscription data records customer choices to subscribe to optional features, such as OCS and CFB; all subscription data is statically provisioned. Some features are compulsory (for example, the Emergency Break-In feature (EBI) which allows emergency service officials to break into an existing conversation). All customers are considered to subscribe—albeit *gratis*—to such features, and these subscriptions too are recorded in subscription data.

The *feature specification* data records the general external specifications of individual features: that is, their relationships to usages and to each other but not the behaviors of the feature boxes that provide them. These general specifications describe the applicability of each feature, identify the box types that provide it, and define a precedence order (not partitioned by feature) in which feature boxes are configured into a usage. Feature specification data is written by system engineers when features are introduced or modified.

The *configuration data* records the set of boxes that exist in the system, the DN of each line interface box, and the internal addresses used for interface and feature boxes. This data provides a part of the underlying mechanism for routing, but does not affect the selection of feature box types to be configured into usages.

2.5 Nonrouting Data

Another global set of data, not used in routing, is the *feature operational* data. Most of this data is provided by customers and accessed only for reading by the features that use it: for example, a forwarding number for CFB is provided by the customer and read by a CFB feature box. Some of this data may be written and read by different boxes cooperating to provide one feature: for example, the Automatic CallBack (ACB) feature is provided by one box that stores the caller ID for an unanswered call, and another box that later places a return call at the callee's request.

The local state of a feature box may contain data in any necessary form. For example, a Collect feature may allow the caller to record a spoken identification to be played to the callee: the feature box can hold this recorded message as internal data for subsequent playback to the callee.

2.6 More Complex Routing

In general, the behavior of the router will not be so simple as the sketch above. A detailed description of the routing scheme is given in Section 3; here we mention only some obvious deviations from the simple sketch.

- Often the target DN cannot be determined from the original dialed string. If the customer is using the Speed Calling (SC) feature, the target DN must be obtained by the SC feature box from the SC feature operational data; if the customer is using the Sequence Credit Card Calling (SCCC) feature, the target DN for follow-on calls must be collected by the SCCC feature box from digits dialed by the customer during the usage.
- Some features may use the command field of the setup message to change the routing. The Call Forwarding Always (CFA) feature changes the target before the original callee is reached; this change will require reselection of the callee features and hence reconstruction of the unused part of the routing list. A usage resulting from this kind of feature behavior can be regarded as composed of *segments*, each segment having its own "Source," "Dialed," and "Target" zones.
- Not all usages can be depicted by linear graphs. A usage may *fork* as a result of a customer service request during the usage. A customer who subscribes to Three-Way Calling (3WC) can flash and dial digits to request connection to a third party while maintaining an existing connection to another customer. *Joins*, too, can occur. For example, the Call Waiting (CW) feature joins a new caller customer into an existing connection.

2.7 Routing: Independence of Features

Although routing can be quite complex, as indicated above, it does not encroach on the functions of the feature boxes. In particular, it does not access the feature operational data. For example, when the subscription data shows that a callee customer subscribes to CFA, the router merely inserts a CFA box into the usage; it does not examine the operational data and reroute the call to the provisioned CFA destination. Similarly, for a customer subscribing to OCS the router inserts an OCS box into the usage: it does not examine the dialed string and abort the call if the called number is in the OCS customer's screening list.

2.8 Boxes and Ports

Calls are connected between *named ports* of boxes. Each internal call that is active at a line interface or feature box requires a dedicated port of that box. In Fig. 1, each of FBa and FBb has two active calls; each, therefore, must have at least two ports. A Three-Way Calling (3WC) box must have three ports. A feature box providing Large-Scale Conferencing (LSC) for up to 10 participants must have at least 11 ports: one for each participant and one for the operator.

Each port of a box is statically typed as a *callee*, *caller*, or *dual* port. A callee port can only receive calls; a caller port can only place calls; a dual port can play either role.

A line interface box communicates through the DFC substrate with other boxes, making and receiving calls on behalf of the customer it serves. It can participate in only one internal call at any time, any multiplexing being handled by boxes that provide call multiplexing features such as CW or 3WC. A line interface box, therefore, has exactly one port, of the dual type. Obviously the interface box must also interact with its own customer line, but these interactions are not constrained by the DFC architecture; rather, they obey the native protocol of the external line or trunk. Since these interactions are not internal calls they do not require a DFC port.

2.9 Connecting Internal Calls

Calls are carried by the DFC substrate, which consists of the switch, its embedded router, and numerous voice and signaling channels. Each active port in the system is communicating with one port of another box by a two-way voice channel, and with the switch by two reliable FIFO buffered signaling channels `in` and `out`. In addition, each box that has a callee or dual port (and can, therefore, receive calls) is connected to the switch by two reliable FIFO buffered signaling channels `in` and `out`, that are associated with the box itself and not with any port.

Each call is initiated by a dual or caller port of a box sending a *setup* message to the router. The router determines the callee box for the call, and the switch forwards the message to that box. The box might have no idle callee or dual port at which to receive the call, in which case it responds with a *quickbusy* message and the call attempt has failed.

Alternatively, the box might accept the call. In this case, the box reserves an idle callee or dual port for the call, and returns a *reserve* message that names the chosen port in a data field. Upon receiving the *reserve* message, the switch informs the caller and callee ports, and provides a two-way voice channel between them. If every feature box in a linear usage connects its incoming and outgoing voice channels, an unbroken voice path is formed between the originating and terminating customer lines.

Either port can end a call by sending a *teardown* message. The switch will respond by disconnecting the voice channel. The teardown phase of the call protocol ensures that both ports are disconnected from the switch and their signaling channels are flushed.

When a call has been set up between two ports, either participating port can send *status* messages, and the switch will forward them to the other port. For example, the callee port can send a status message indicating that the target destination is busy, or is alerting. The set of possible status messages can be easily extended to meet the needs of new features.

2.10 In-Band and Out-of-Band Signaling

Messages sent by a port on its outgoing signaling channel (out-of-band signals) are typically passed from box to box along the chain formed by their configuration into a usage. As a result, the precedence ordering of features and appropriate design of feature boxes can be used to resolve any conflict among boxes awaiting the same signal.

Consider, for example, a customer who subscribes to both CFB and Callee Messaging-on-Busy (CMB). Evidently, there is a conflict between the two feature boxes: if an incoming call, in which both CFB and CMB feature boxes are configured, arrives while the customer is busy, CFB must redirect the call to another DN, while CMB must invite the caller to leave a voicemail message. In the DFC architecture, busy status is signaled out-of-band. Once the usage has been completed as far as the customer's line interface box, a busy signal is sent from that LI box. Whichever of the two feature boxes CFB and CMB is closer to the LI box will receive the busy signal, act on it, and not propagate it further back in the chain; the more distant feature box will not receive the signal at all. Feature precedence determines how the conflict will be resolved.

Conflicts in respect of in-band signals sent on a voice channel (in-band signals) are not so easily resolved. If the LI box were to signal its busy status by emitting an in-band tone, both feature boxes could detect the tone simultaneously. The conflict might then be resolved by the outcome of a race, with all its attendant difficulties. For the most part, then, we prefer to rely on out-of-band status messages. They have the further advantage that they can carry additional information in the message data fields.

One might jump to the conclusion that all voice analysis and generation functions should be performed at line interfaces, so that feature boxes would handle only out-of-band signals. But this is not feasible, because a system's full repertoire of tones, announcements, and recognition vocabularies cannot be anticipated. Our scheme is, therefore, a compromise in which line interface boxes translate the most common in-band signals to and from out-of-band signals. Where feature box processing of in-band signals is inescapable, it is sometimes possible to avoid race conditions, using a technique described in Section 4.9.

2.11 Line Interface Box Behavior

The regime of out-of-band status messages has significant consequences for the specification of line interface boxes. Each LI box translates between its own particular line protocol and the protocol of the virtual network: this translation must include translation from status messages to tones familiar to the human customer. For example, on receiving a busy status message, a calling customer's LI box must place a busy tone on the outgoing voice channel of its line so that the customer can hear it.

The DFC call protocol accommodates, but does not define, a wide range of status message patterns that a LI box must translate into tones. For example, a user of SCCC can

make a sequence of customer calls within one usage, dialing '#' instead of going onhook at the end of each call except the last. The subscriber's LI box must produce the appropriate tones—alerting, busy, unobtainable—during each call, mute the alerting tone when a callee answers, and produce dialtone when '#' is dialed. These tones must be produced by the LI box in response to messages sent or passed back by the SCCC feature box. Because a LI box must accommodate sequences of calls, as in the SCCC feature, it never interprets a tone message as a disconnect.

A LI box that receives a setup message when its one port is already active returns a *quickbusy* message. The box whose call attempt has failed will typically pass back a *busy* status message that eventually reaches the calling customer's LI box, where it will be translated into a busy tone on the line. A LI box that sends a setup message and receives an immediate *quickbusy* message in response must translate this *quickbusy* message into a busy tone on the line. (In practice, this can scarcely occur, because a usage with no feature box is unlikely: a setup message sent by a LI box will therefore usually be routed first to a feature box, which will be able to accept the call.)

2.12 Feature Box Availability

Like conference bridges in a conventional physical architecture, feature boxes in the DFC architecture are regarded as permanent, named individuals. At any particular time, a feature box may be *occupied* or *available*. When a feature is to be inserted into a usage, an available box of the right type is found, and the switch sends the setup message to that box. It is assumed that the population of feature boxes is effectively unlimited: an available box of the right type can always be found when one is needed.

A box may be occupied although it is not currently participating in any call. For example, a box providing a messaging feature may try to deliver a caller's recorded voice message at regular intervals until the intended recipient customer answers the phone. Between attempts, the box is not available: it is occupied waiting for the current interval to elapse before it makes a further attempt.

2.13 Box Classes: Free, Bound, and Addressable

For many features—such as OCS—any available box of the appropriate type may be selected when one is required. Such boxes are called *free* boxes, because they may be freely selected. Some features—such as Call Waiting (CW)—do not offer this freedom. To provide CW service, a CW box must be inserted into each usage of the subscribing customer; any incoming customer call that arrives while the usage is in progress must then be directed to that particular CW box. Only in this way can the new call be joined into the existing usage.

To handle this kind of requirement conveniently, we classify the CW box as a *bound* box: it is bound to the line interface of the subscribing customer. Whenever that customer makes or receives a call, the particular CW box bound to that customer will appear in the usage. Essentially, the class of bound boxes is the class of boxes at which a join can occur.

In addition to free and bound boxes there is a third class. Some features are directly addressable by customers through the standard dialing plan. For example, in a Large-Scale Conferencing (LSC) feature, intended participants in a conference may be informed of a special DN to call; the feature box at that DN connects them into the conference after checking a password. Boxes implementing such features are *addressable* boxes.

3 Description of the DFC Architecture

Later, in Section 4, we consider a number of features and show how they may be specified and combined in the DFC environment. First, in this section, we give a more detailed and formal description of the DFC architecture. We begin by describing the configuration: that is, the boxes, their types and classes, and how they are addressed. Then we describe the subscription, specification, and operational data. Then we describe the features, their relationship to the configuration and to the data, and the router's behavior. Next we describe the protocol of internal calls observed by boxes and ports. Finally, we describe the possible behaviors of feature boxes in managing their calls, including the manipulation of voice and signal paths.

In each part, we use a notation convenient for the purpose in hand, connecting our formal descriptions by informal narrative. We write relations indicating their types, in the Z style [23], by distinctive arrow symbols (such as $\twoheadrightarrow$ for a total surjection). We also use relations as sets of pairs, and *vice versa*, wherever it is convenient to do so. Appendix A lists the symbols for relations.

3.1 Configuration Data: Boxes, Addresses, and DNs

The basic sets of the configuration are these disjoint sets:

$[LIBox, FFBox, BFBox, AFBox, FFBType, BFBType, AFBType, Port, DN]$.

They are, respectively: the set of LI boxes; the sets of free, bound and addressable feature boxes; the sets of free, bound and addressable feature box types; the set of ports; and the set of well-formed directory numbers.

Two further sets, the set of all feature boxes and the set of all boxes, are defined from these basic sets:

$$FBox \mathrel{\widehat{=}} FFBox \cup BFBox \cup AFBox; \qquad Box \mathrel{\widehat{=}} FBox \cup LIBox;$$

Each feature box has an appropriate (free, bound, or addressable) type, and each type has boxes:

$$freeType : FFBox \twoheadrightarrow FFBType; \qquad boundType : BFBox \twoheadrightarrow BFBType;$$

$$addrType : AFBox \twoheadrightarrow AFBType;$$

Each port belongs to one box and each box has at least one port:

$$portBox : Port \twoheadrightarrow Box;$$

Each LI box and each addressable feature box has a unique DN:

$$LIbDnum : LIBox \rightarrowtail DN; \qquad AFbDnum : AFBox \rightarrowtail DN;$$

$$\mathbf{ran}\ LIbDnum \cap \mathbf{ran}\ AFbDnum = \emptyset$$

Each bound feature box is bound to exactly one LI box:

$$boundLI : BFBox \rightarrow LIBox;$$

3.2 Specification Data: Features, Box Types, and Zones

The additional basic sets for specification data are [*Featr, Zone*]. They are the set of features and the set of zones {"Source," "Dialed," "Target"}; they are disjoint from each other and from all other basic sets.

The set *FBoxType* of all feature box types of any class is defined; each box type provides all or part of exactly one feature, and each feature is provided by at least one box type (a feature may be provided by cooperating boxes of different types):

$$FBoxType \mathrel{\widehat{=}} FFBType \cup BFBType \cup AFBType;$$
$$provFeatr : FBoxType \twoheadrightarrow Featr;$$

A feature may be associated with more than one zone: For example, 3WC and CW are used in both the outgoing and the incoming calls of their subscribers, and therefore belong to both source and target zones. The association of features with zones is, therefore, many-to-many. However, this association plays no direct part in the formal description of a DFC system: instead, the important association is of feature box types with zones.

For each box type we specify whether it is a source, target or dialed feature box, or more than one of these. (An example of a feature box type associated with more than one zone is a CW feature box: the same type is used for CW on an outgoing customer call as on an incoming call.) Each box type is associated with at least one zone:

$$boxZone : FBoxType \leftrightarrow Zone;$$
$$\forall t : FBoxType \bullet t \in \mathbf{dom}\ boxZone$$

No bound box type is in the dialed zone:

$$BFBType \lhd boxZone \rhd \{\text{"Dialed"}\} = \emptyset$$

But every addressable box type is only in the dialed zone:

$$boxZone\ (|\ AFBType\ |) = \{\text{"Dialed"}\};$$

3.3 Further Specification Data: Box Applicability and Precedence

An element (*t, "Source"*) or (*t, "Target"*) of the relation *boxZone* given in Section 3.2 indicates that a feature box of type *t* may be applied to a customer call in the source or target zone. Whether it will be so applied to calls made by particular customers is determined by *subscription data*, as described in Section 3.4. For dialed features, the determination depends on a relation in *specification data*. This is a relation between sets of strings—for example, all strings beginning with "0"—and feature box types:

$$dialSetFBType : \mathbb{P}\ String \nrightarrow FBoxType;$$
$$\mathbf{ran}\ dialSetFBType = boxZone\sim\ (|\{\text{"Dialed"}\}|);$$

The basic set *String* is the set of strings over the characters (0..9, '*', '#') that can be dialed. The relation *dialSetFBType* is given, tuple by tuple, in the specification of the features provided by the boxes (the sets of strings being specified by predicates over strings). One string may cause selection of more than one feature box type. For purposes of explanation it is more convenient to relate strings to feature box types directly:

$$stringFBType : String \leftrightarrow FBoxType;$$
$$\forall s : String, t : FBoxType \bullet (s, t) \in stringFBType$$
$$\Leftrightarrow \exists ss \bullet s \in ss \wedge (ss, t) \in dialSetFBType;$$

Selection of a box to provide a source or target feature is governed by *boxZone*, by the source or target field in the setup message, and by the relation *subscrip* in the subscription data (see Section 3.4). Selection of a box to provide a dialed feature is governed by *boxZone*, by the dialed-string field in the setup message, and by the relation *stringFBType* in the specification data.

Ordering of feature boxes in a routing list is constrained by a specified precedence relation:

$$boxPrec : boxZone \leftrightarrow boxZone;$$

The pair (*(fb1, z1), (fb2, z2)*) is an element of *boxPrec* iff in any routing list containing both a box of type *fb1* applied as a *z1* feature and a box of type *fb2* applied as a *z2* feature the *fb1* box must precede the *fb2* box. *boxPrec* must be a partial order.

The specified precedence applies to *routing lists* as constructed by the router and held in *setup* messages, not to *usages*. Because usages can be forked and joined, and a new routing list can be constructed for a partially complete usage, the order of boxes in usages is less constrained than their order in routing lists. The informal concept of a *segment* corresponds to the part of a usage in which the feature box order must conform to the precedence specified by *boxPrec*.

Since feature boxes are placed in the routing list in the order source, dialed, target, the partial order *boxPrec* must satisfy:

$$\forall fb1\ fb2 : FBoxType, z1, z2 : Zone \bullet$$
$$((fb1, z1), (fb2, z2)) \in boxPrec \Rightarrow (z1 = \text{"Source"})$$
$$\vee (z2 = \text{"Target"}) \vee (z1 = z2);$$

3.4 Subscription Data: Features, Feature Boxes, and DNs

A subscriber subscribes to features, ensuring that the appropriate feature boxes are applied to the subscriber's calls in the appropriate zones. However, the effect of a subscription may be different for different subscribers. For example, the Emergency Break-In (EBI) feature is used by privileged emergency subscribers to place calls that will reach even a busy telephone; normal customers are compelled to receive such break-in calls when they are engaged in an outgoing or incoming call of their own. The emergency service subscribes to EBI in the source zone; the normal customer subscribes, compulsorily, in both source and target zones. EBI is provided by two feature box types: a source zone free box for the emergency service, allowing break-in calls ro be made; and a source and target zone bound box for the normal customer, ensuring that incoming break-in calls are always received. Notice that the normal customer's bound EBI box type is in both source and target zones (as is the Call Waiting (CW) box discussed in Sections 4.4 and 4.6).

In this paper, we ignore the process of setting up subscriptions to features, and regard subscriptions simply as subscriptions to feature boxes in zones. The basic sets [*FBoxType, Zone*] were introduced above. The subscription data is the relation *subscrip*, between DNs and (*FBoxType, Zone*) pairs. Since the relation *boxZone* is precisely the set of

such pairs (t, z) in which box type t is associated with zone z, we have:

$subscrip : DN \leftrightarrow (boxZone \rhd \{\text{“Source,” “Target”}\})$;

The relation *subscrip* governs only features in the source and target zones; applicability of features in the dialed zone is governed by the relation *dialSetFBType* described in Section 3.3.

The configuration data relation *boundLI* given in Section 3.1 is constrained by *subscrip*. If the customer at DN d subscribes to a bound feature box type t, then there must be exactly one box of type t bound to the LI at d. The constraint is:

$\forall\, d : DN, i : LIBox, t : BFBType \bullet$
$(i\ LIbDnum\ d \land t \in \mathbf{dom}\ subscrip\ (\!|\, d \,|\!)) \Rightarrow$
$(\exists!\ b \bullet b\ boundLI\ i \land\ b\ boundType\ t)$;

3.5 Operational Data: Accessibility

Feature operational data consists of a set of relations [*OpernRel*]. Each of these relations supports exactly one feature; some features may be supported by more than one relation, some by none:

$suppFeatr : OpernRel \rightarrow Featr$;

Each relation is accessible only to feature boxes of the types that provide the feature supported by the data. This constraint contributes importantly to feature independence:

$boxRel : FBoxType \leftrightarrow OpernRel;\quad boxRel = provFeatr \fatsemi suppFeatr\sim$;

3.6 Routing: The Setup Message

A setup message is sent from a calling box to the switch. The router embedded in the switch modifies the message contents before the switch transmits the message to a callee box; typically, that box will create another setup message by copying all or part of the first message, and send it in turn to the switch.

The full structure of the setup message is:

$setup = (source, target : DN \cup \mathbf{nul};\ dialed : String \cup \mathbf{nul};$
$route : (\mathbf{seq}\ boxZone) \cup \mathbf{nul};$
$command : \{\text{“new,” “update,” “continue,” “direct”}\};$
$modifier : \mathbb{P}\ Zone \cup FBoxType \cup \mathbf{nul})$;

the *command* field and its *modifier* are set by the sending box to control the action of the router.

In a setup message sent by an LI box: *source* = DN of the LI; *target* = **nul**; *dialed* = the dialed string; *route* = **nul**; *command* = “new” and *modifier* = **nul**.

In setup messages sent by feature boxes many combinations of values are possible; they must conform to the constraints described in the next section. Further, a box may not access the *route* component except to set it to **nul** or to the value in an incoming setup message previously received by the box.

3.7 Routing: Setting the Target and Routing List

Initially the router examines the values of *target* and *dialed*. If *dialed* ≠ **nul** and *target* = **nul**, the router sets a value in *target* derived from *dialed* according to the dialing plan; this value may be **nul** when the dialed string does not indicate a target DN (for example, if the dialed string is a speed-calling code, or an 800-number).

Then, after acting on the value of *command*, as described in this section, the router determines the destination box to which the message should be sent by the switch, as described in Section 3.8.

- If *command* = “new,” the router computes a new routing list, as explained below, for all three zones, and inserts it into *route*.
- If *command* = “update,” then *modifier* = a subset of {“Source,” “Target,” “Dialed”} and route ≠ **nul**. The router computes a new routing list for each zone specified in *modifier*, and uses it to replace the existing value of *route* for each of those zones.
- If *command* = “continue,” then route ≠ **nul** (although it may be empty); *modifier* is ignored. The router leaves the existing value of *route* unchanged.
- If *command* = “direct,” then *modifier* = *bt*, where *bt* is a box type providing the same feature *f* as the box that sent the setup message, and *target* = *dn*, where *dn* is a subscriber to *bt* in zone “Target.” The router replaces the existing value of *route* with the singleton sequence $\langle (bt, \text{“Target”}) \rangle$.

Each zone of the routing list is computed as follows:

- For zone “Source” or “Target” the list is empty if *source* or *target*, respectively, is **nul**. Otherwise the list contains those box types of the zone that are subscribed to by the *source* or *target*, respectively. For “Source” and “Target” these are the following subsets of *boxZone*:

  $(subscrip\ (\!|\ source\ |\!)) \rhd \{\text{“Source”}\}$

  and

  $(subscrip\ (\!|\ target\ |\!)) \rhd \{\text{“Target”}\}$

- For zone “Dialed” the list is empty if *dialed* = **nul**. Otherwise the list contains the following subset of *boxZone*:

  $(stringFBType\ (\!|\, dialed \,|\!)) \lhd boxZone$

  The set $stringFBType\ (\!|\, dialed \,|\!)$ is the set of dialed zone feature boxes to be applied to the string *dialed*.

The router ensures that the complete routing list value set in *route* satisfies the precedence relation *boxPrec*.

3.8 Routing: The Destination Box

After resetting *target* and *route* as described in Section 3.7, the router chooses the destination box *b* as follows:

- If *route* is empty, *b* is chosen according to the value of *target* as follows:
  - If *target* is not the DN of any LI box or addressable feature box, then *b* is any available Dialing Error Response (DER) feature box (a free box).
  - If $target \in LIbDnum$ then $b = LibDnum\sim(target)$.
  - If $target \in AFbDnum$ then $b = AfbDnum\sim(target)$.
- If *route* is not empty, the head element is (t, z) of type $FBoxType \times Zone$. This element is removed from the *route* list, and used to choose *b* as follows:
  - If $t \in FFBType$ then *b* is any available box of type *t*.
  - If $t \in BFBType$ then *b* is the box of type *t* bound to $LibDnum\sim(source)$ if z = “Source” and to $LibDnum\sim$

TABLE 1
MESSAGE TYPES IN THE DFC CALL PROTOCOL

Phase	Message Type	Data Fields	From/To
setup and teardown phase messages	setup	(see Section 3.6)	caller to box
	quickbusy	*initiator: DN*	box to caller
	reserve	*reserved: port*	box to switch
	upack	none	switch to caller
	init	none	switch to callee
	teardown	none	caller to callee and callee to caller
	downack	none	caller to callee and callee to caller
status messages	busy	*initiator: DN*	callee to caller
	alerting	*initiator: DN*	callee to caller
	answered	*initiator: DN*	callee to caller
	unobtainable	*unallocated*: *String*	DER box to caller
	dialtone	none	caller to callee and callee to caller
	quiet	none	caller to callee and callee to caller
	flash	none	caller to callee and callee to caller
	DTMF	*dialed-char: Char*	caller to callee and callee to caller
	…	…	callee to caller and /or caller to callee

(*target*) if *z* = "Target." Such a box necessarily exists, by the constraints given in Sections 3.1 and 3.4, which guarantee the existence of a bound box for each subscriber to the feature implemented by the box.

After the router has modified the setup message and determined *b*, the switch transmits the modified setup message to the chosen box.

3.9 Calls: Message Types

The message types for internal calls are shown in Table 1, along with their fields, senders, and recipients.

The setup and teardown phase messages are concerned solely with the connection and disconnection of calls; the protocols for their use are given in Section 3.10.

The other messages are status messages, used to communicate between call participants after the setup phase and before the teardown phase. Some messages travel only from a callee to a caller, while others can travel in either direction. The types of status message shown are of basic utility in most systems; additional message types and data fields can be freely defined for the purposes of particular features.

Among the types shown, *busy*, *alerting*, *answered*, *unobtainable*, *dialtone*, and *quiet* are of particular importance: they are the status messages that a receiving LI box must translate into audio tones (or silence) on the external line to the customer's telephone, as described in Section 2.11. *DTMF* and *flash* status messages originate at a LI box when the customer dials a digit or flashes the switchhook while a call is in progress.

3.10 Call Protocols

The protocol for calls to be executed by feature boxes is in three parts: a protocol for the caller port; a protocol for the callee box; and a protocol for the callee port. The protocol for the switch is given in Section 3.11.

Each protocol is described below in Promela [16]. A message of type **msg** is sent on channel **out** by the statement **out!msg**. A message of type **msg** is received on channel `in` by the statement **in?msg**. An **if** statement is executed by executing exactly one of its executable alternatives; in these protocols an alternative is executable if it is unguarded or guarded by a send statement or by a receive statement for a message that is available as the next message on the channel. A **do** statement is like an **if** statement except that it continues to execute alternatives until there is an explicit exit from the loop by means of a `break` or **goto** statement.

A callee port on a box is capable of receiving a sequence of calls. Each of its calls must satisfy the following protocol:

```
begin:      in?init; goto linked;
linked:     do
            :: out!status
            :: in?status
            :: out!teardown; goto unlinking
            :: in?teardown; out!downack; goto end
            od;
unlinking:  do
            :: in?status
            :: in?teardown; out!downack
            :: in?downack; goto end
            od;
end:        skip
```

A caller port on a box is capable of making a sequence of calls. Each of its calls must satisfy the following protocol:

```
begin:      out!setup; goto requesting;
requesting:if
            :: in?upack; goto linked
            :: in?quickbusy; goto end
            fi;
linked:     do
            :: out!status
            :: in?status
            :: out!teardown; goto unlinking
            :: in?teardown; out!downack; goto end
            od;
unlinking:  do
            :: in?status
            :: in?teardown; out!downack
            :: in?downack; goto end
            od;
end:        skip
```

A dual port can participate in a sequence of calls. During each call it plays the role of either a caller port or a callee port. Any port is busy when it is in a requesting, linked, or unlinking state, and idle otherwise. Once a call is successfully established the protocols for caller and callee are identical. Because communication is asynchronous a port can receive incoming messages even after it has written a teardown message.

The life history of a box and its own signaling channels (those not associated with particular ports) must conform to the following callee box protocol:

```
begin: do
       :: in?setup; if
                    :: out!reserve
                    :: out!quickbusy
                    fi
       od
```

If the box sends a **reserve** message, it must be accompanied by the identifier of an idle, unreserved callee or dual port. That port must remain idle and reserved until it receives its **init** message from the switch, at which time it becomes busy and unreserved.

3.11 Switch Call Protocol

The corresponding protocol to be observed by the switch is more complex because it must take account of the interleaving of many calls and of the connections between them. In the protocol given below, it is assumed that:

- There are four arrays of signaling channels *p_s*, *s_p*, *b_s*, and *s_b* used, respectively, for sending out-of-band messages from a port to the switch, from the switch to a port, from a box to the switch and from the switch to a box. They are indexed by port and box identifiers (*[p]* and *[b]*). Each channel is an unbounded reliable FIFO queue.
- The switch procedures *CONNECT[p,q]* and *DISCONNECT[p,q]*, respectively, create and destroy a voice path between ports *p* and *q*. The order of operands is immaterial.
- The function *ROUTE: b* returns the box identifier *b* of the destination box for the most recently read *setup* message, determined by the router as described in Section 3.8.
- The switch reads its input channels fairly (in some sense), and is fast enough to ensure that no call suffers any significant degree of starvation.

The switch maintains internal variables whose values represent the connection state:

- Current out-of-band signaling connections between ports are represented by elements of an array *WTF* ("where to forward"). If ports *p* and *q* are currently connected in a call then *WTF[p]==q* and *WTF[q]==p*. If port *p* is not currently connected to any port, then the value of *WTF[p]* is not defined.
- The array *TDF* has an element for each port; this array is used in the processing of *teardown* and *downack* messages, as explained below. Initially the value of *TDF[p]* is *0* for all *p*.
- There is an array of queues containing a queue for each box *b*. The queue for a box holds the port identifiers of senders of *setup* messages that have already been sent to the box but for which the switch has not yet received a *reserve* or *quickbusy* message from the box. The operation *ENQ[x,b]* and the function *DEQ[b] : p* have the obvious meanings.

In the protocol we depart from strict Promela in three minor ways. First, the use of parentheses rather than square brackets in the notation

```
:: p_s(p)?msg;...p...
```

occurring in a looping or conditional statement is a shorthand for

```
:: ps[p1]?msg; p=p1;...p...
:: ps[p2]?msg; p=p2;...p...
:: ps[p3]?msg; p=p3;...p...
```

etc. The syntax is reminiscent of

```
:: b_s[b]?reserve(y)
```

in Promela, which assigns to y the value of the data field of the reserve message. The same effect can be achieved in strict Promela with a slight loss of readability. Second, we index arrays by arbitrary box and port identifiers; in strict Promela array index values must be integers. Third, we assume unbounded queues.

The protocol is:

```
/* switch protocol; initially TDF[p]==0
     for all ports p */
do
:: p_s(x)?setup; b=ROUTE; ENQ[x,b];
     s_b[b]!setup
:: b_s(b)?quickbusy; x=DEQ[b];
   s_p[x]!quickbusy
:: b_s(b)?reserve(y); x=DEQ[b];
   if
   :: TDF[y]==0
   :: TDF[y]==1; z=WTF[y];
      do
      :: p_s[y]?status; s_p[z]!status
      :: p_s[y]?downack; s_p[z]!downack;
           TDF[y]=0; break
      od
   fi;
   CONNECT[x,y]; WTF[x]=y; WTF[y]=x;
     s_p[x]!upack; sp[y]!init
:: p_s(x)?status; y=WTF[x]; sp[y]!status
:: p_s(x)?teardown; y=WTF[x];
     s_p[y]!teardown; TDF[y]=1;
   if
   :: TDF[x]==1
   :: TDF[x]==0; DISCONNECT[x,y]
   fi
:: p_s(x)?downack; y=WTF[x];
     s_p[y]!downack; TDF[x]=0
od
```

TDF[y]==1 when (and only when) the switch has sent a *teardown* message to *y* and has not yet received the corresponding *downack* message from *y;* at all other times *TDF[y]==0*. The global initialization *TDF[p]=0* for all ports *p* is necessary to ensure that if a port *y* is the callee in its first call, then *TDF[y]==0* is correctly set when the *reserve(y)* message for that call arrives at the switch. At the comple-

tion of every call of port *y* the value of *TDF[y]* is guaranteed to be *0*: For a call in which *y* has received a *teardown* message, the assignment *TDF[y]=0* is executed when the *downack* sent by *y* is received at the switch; for a call in which *y* receives no *teardown* message, the value of *TDF[y]* is unchanged from the global initialization *TDF[y]=0* or from the completion of *y*'s preceding call.

When a *reserve(y)* message is received by the switch, port *y* must be idle, since otherwise its box would not have selected it to receive a call. Therefore, it must have received all its incoming messages and written all of its outgoing messages of its immediately preceding call *c* (if there has been such a call). However, there may be some messages of call *c* on the *p_s[y]* channel that have been sent by *y* but not yet received by the switch. If *TDF[y]==1*, then port *y* has been sent a *teardown* message in call *c*, but its acknowledgment *downack* has not yet been received by the switch. The signal channel *p_s[y]* from *y* to the switch must then be "cleaned," because it is still in use from call *c*. If *TDF[y]==0*, then the channel *p_s[y]* must be empty. Either the switch has already received a *downack* from port *y*; or else port *y* has itself already received a *downack*, in which case the final message of *p_s[y]* was the *teardown* of which that *downack* was the acknowledgment.

The first *teardown* message of a call received by the switch causes the switch to disconnect the voice path. When a *teardown* message arrives from port *x*, the guard *TDF[x]==0* correctly specifies the condition that this is the first *teardown* message of the call to pass through the switch. For if *TDF[x]==0* because port *x* has already received a *teardown* message and its acknowledging *downack* has already reached the switch, then that *downack* would have preceded the *teardown* message that has just arrived, contrary to the port protocol.

3.12 Feature Box Behavior

A feature box and its ports must observe the signaling protocols described in Section 3.10, and must send only correctly formed messages as described in Sections 3.6, 3.7, and 3.9. In this section, we describe the internal voice-processing behavior of a box.

In a legal box state the external output of each port is exactly one of these four alternatives:

1) silence,
2) the external input from another port of the box,
3) the normalized sum of the external inputs from a set of other ports of the box, or
4) a specified sound such as a tone or announcement.

In addition, the external input from a port can be recorded or analyzed for recognition of a specified kind of pattern. Recording and analyzing port states are not mutually exclusive or exclusive of any port-output state.

A language for box programming must make all legal voice-processing states achievable. Its semantics or analysis tools should also help the programmer avoid illegal states and states that waste implementation resources such as mixers.

The language commands for creating active port states must specify certain information. A *record* command must specify where the recording is to be stored. A *play* command must specify the sound to be played; the sound might be given as a tone frequency, a recording, or even as text (in the presence of text-to-speech facilities). An *analyze* command must specify the kind of pattern to be recognized and the place where the result of the recognition is to be stored. The kind of pattern is usually specified in terms of a vocabulary. In the presence of automated speech recognition this will typically be a vocabulary of single word commands.

In addition to these commands, a convenient box-programming language should provide events through which a program can be notified of the status of ports in active states. It should be possible to notify a box program that an announcement of fixed duration has been played completely. It should be possible to notify a box program that recording has been completed, where completion might be defined in terms of a maximum recording time or a minimum interval of silence. It should also be possible to notify a box program that recognition has been completed, where completion might be defined in terms of a maximum sequence length of recognized words, a minimum interval of silence, or recognition of a distinguished "interrupt" word.

4 Feature Specifications

In this section, we discuss the specification of various features in a DFC setting: the features and their combinations are chosen to illustrate various aspects of the architecture and its implications. The suggested specifications are neither complete nor in any way definitive: they are intended only to illustrate a possible approach to the features discussed in a DFC setting. The space limitations of this paper preclude the presentation of full programs for feature boxes. Some feature box programs are given in [25]. We, therefore, present only the barest informal sketches, but we have tried to address the chief points likely to cause difficulty in each case.

4.1 The Null Feature Box: *Transparent* and *End* States

We begin by describing a null feature box. It is completely transparent: its presence has no effect on any usage into which it is configured. The description below should be thought of as that of a process executing in parallel with the protocols on the active ports and sharing events with them.

The null feature box has a callee port *p1* and a caller port *p2*. It becomes *occupied* on receiving a *setup* message; it accepts the call on *p1*. It then places a call on *p2*, using a copy of the *setup* message of the *p1* call. If the *p2* call attempt fails, it sends a *busy* status message followed by a *teardown* message on *p1*, and becomes *available* on receiving the corresponding *downack*. If the call on *p2* succeeds, it enters the *transparent* state: that is, it joins the voice channels of *p1* and *p2*, and proceeds to copy all incoming out-of-band signals from each port to the other. When the teardown subprotocol on both ports is complete the box enters an *end* state and becomes *available* again.

4.2 Target Identity: 800-Service, Speed Calling, and Call Forwarding Always

Three features that change the target of a subscriber call are Speed Calling (SC), 800-service (800), and Call Forwarding

Always (CFA). Each is provided by a feature box which is like the transparent box with one exception. To place the *p2* call, it uses an altered copy, not a true copy, of the *setup* message of the incoming call on port *p1*. In each case the *target* field is altered, and the value of *command* is "update" with a *modifier* {"Target"}, ensuring that the target zone of the route will be recomputed by the switch. Notice that SC, 800 and CFA are, respectively, Source, Dialed, and Target zone features, although each of them changes the target of a call.

The 800-service feature allows a commercial subscriber to pay for all incoming calls; the calls are placed by dialing an easily remembered string such as 1-800-123-4567. The feature is a dialed zone feature: the string is not a DN, and the 800 feature box determines the required DN from its operational data relationt

$$eightDN : String800 \pfun DN$$

The altered value of *target* is *eightDN* (*dialed*), or **nul** if *dialed* $\notin$ **dom** *eightDN*.

The SC (Speed Calling) feature is a source zone feature. It uses an operational data relation

$$sourceSpeedDN : DN \pfun (SpeedCode \pfun DN);$$

containing each *source* subscriber's private mapping from speed-calling codes to frequently called DNs. The altered value of *target* is **nul** if *source* $\notin$ **dom** *sourceSpeedDN* or if *dialed* $\notin$ **dom** *sourceSpeedDN (source)*; otherwise it is *sourceSpeedDN (source)* (*dialed*).

Call Forwarding Always (CFA) is a target zone feature. It uses an element of an operational data relation

$$aForwardDN : DN \pfun DN$$

mapping the *target* subscriber's DN to a *forward* DN. The altered value of *target* is *aForwardDN* (*target*), or **nul** if *target* $\notin$ **dom** *aForwardDN*.

4.3 Target Identity: OCS

Originating Call Screening (OCS) is a source zone feature; its function is to prevent calls to barred numbers on a screening list maintained by a subscriber to the feature. Subscribers enter their barred numbers as strings (*DN* is a proper subset of *String*). The operational data is:

$$screenOCS : DN \rightarrow String$$

For each number *s* barred by *source*, *screenOCS* has an element (*source*, *s*). The key choice in specifying OCS is: which fields of which messages are candidate values of *s*? Three possible answers are as follows:

1) *dialed* in the *setup* message. In this case dialed speed-call codes and 800-numbers can be barred, but forward DNs reached by CFA cannot: a disobedient child whose parents have barred *DN1* simply asks a friend at *DN2* to forward *DN2*'s calls temporarily to *DN1*, and then dials *DN2*.
2) *target* in the *setup* message. In this case only DNs occurring in *setup* messages in the source zone can be barred. 800-numbers cannot be barred even if their equivalent DNs are known (because 800-numbers are not DNs and do not appear in the *target* field; they are translated to DNs by an 800 feature box in the dialed zone, too late for OCS to operate). Forward DNs cannot be barred (because CFA is in the target zone, and the forwarding, like 800 translation, operates too late for OCS). DNs accessed by speed-calling codes can be barred provided that OCS has a later precedence than SC: if the OCS feature box precedes the SC box then OCS receives only the *dialed* string, before SC has translated it to a *target* DN.
3) *initiator* in an *alerting* or *answered* message. In this case, it is the *DN* of the LI that is finally reached that is subject to barring by OCS. Forward DNs are barred, but an 800-number cannot be barred unless its equivalent DN is known to the subscriber and entered explicitly in the screening list.

By suitable design of the feature box, OCS can be specified to enforce barring of any or all of these fields. If the *setup* message received at port *p1* is barred neither by its *dialed* nor by its *target* value, the *p2* call is placed with the same *setup* message. If the *p2* call is successful, an *alerting* or *busy* message from the LI finally reached will eventually be passed back by transparent behavior on the part of other feature boxes, and can be checked for a barred *initiator* value.

4.4 Joining Usages: Call Waiting

The Call Waiting (CW) feature allows a subscriber to receive an incoming customer call while already engaged on an existing usage. The subscriber can switch to the new call by flashing the switchhook, and back again to the earlier call by flashing again. CW is both a source zone and a target zone feature, because the subscriber may be either the caller or the callee in the existing usage. It must be provided by a bound box, because clearly the second, incoming, call must be routed to the CW box already configured into the existing usage.

Before the second call is received, the CW box behaves transparently. It receives a call, either from the subscriber or from another source. If the call is from the subscriber (in which case the *source* value in the *setup* message is the subscriber's own DN), the call is accepted on port *p1* and a call placed on port *p2* with the same *setup* message field values. If the call is not from the subscriber the call is accepted on port *p2* and a call placed on port *p1*. Both *p1* and *p2* are dual ports.

The CW box has a third port *p3*, which is a callee port. If a call arrives at the box while the *p1* and *p2* calls are still active, it is accepted at port *p3*, an *alerting* signal is sent from port *p3*, and the call-waiting indication (two short tones) is played on the outgoing voice channel of port *p1*, the speech signal being temporarily interrupted for the purpose. The box then waits for one of the following events:

- A *teardown* message on port *p2/p3*. The box tears down the call on that port, connects port *p1* to the remaining port *p3/p2* if it is active and was not already connected, and is now ready to receive a call on the port *p2/p3* that has become free. If the remaining port *p3/p2* was inactive, the call on *p1* is torn down.
- A *flash* message on port *p1*. The box disconnects *p1* from the port *p2/p3* to which it is connected, and connects it to the other port *p3/p2*, provided that there is an active call on that other port; if not, the *flash* message is passed on to the connected port *p2/p3* and is otherwise ignored.

- A *teardown* message on the port *p1*. If *p1* was connected to *p2/p3* and the other port *p3/p2* is inactive, the box tears down both calls and returns to its initial state. If one of the ports *p2* and *p3* is active but has not yet been connected to *p1*, the caller on that port is hearing ringback. The box tears down the call on *p1* but immediately calls the subscriber back on *p1* to connect it to the waiting caller at the active port.

In this way the CW box allows the subscriber to multiplex between two calls. Notice that CW does not require an internal three-way connector, because its two calls are multiplexed, not conferenced.

4.5 Splitting a Usage: 3-Way Calling

3-Way Calling (3WC) is a feature in both source and target zones, provided by a free box with two dual ports *p1* and *p2* and one caller port. Like CW, the box connects the subscribing customer on port *p1* when first activated. It allows the subscriber, when already engaged in a usage, to make a second, outgoing, call and to conference the two calls together. Like CW it uses *flash* messages to signal the subscriber's intentions. A *flash* while only one call is active indicates that the subscriber wishes to make a second call. The 3WC box then put the existing customer call on hold, sends a *dialtone* message to port *p1*, collects a dialed string in-band from digits dialed by the subscriber, and places the dialed call on the currently free port. When that call has been set up and the subscriber flashes again, the box conferences the two calls, joining the voice signals at all of its ports at an internal three-way connector.

Disconnection behavior of a 3WC box is as follows:

- A *teardown* message on port *p2* or *p3* occurring when all three ports are in use makes that port available for placing another outgoing dialed call when the subscriber flashes again.
- A *teardown* message on *p1* occurring when all three ports are in use causes only the call on *p1* to be torn down, leaving the other two ports connected. The 3WC box is then occupied servicing the calls on ports *p2* and *p3* until it receives a teardown message on one of those ports. (The subscriber's LI box, of course, is not involved in this residual connection between ports *p2* and *p3* of the 3WC box.) Since 3WC is not a bound box, a fresh 3WC box will be used for a subsequent call to or from the subscriber.
- A *teardown* message on any port occurring when only two ports are in use ends both calls and returns the box to the *end* state, where it becomes again available.

4.6 CW and 3WC Together

The relationship between CW and 3WC illustrates three DFC principles. The first principle is that a bound box in a usage should always come between its line interface and any free boxes. For CW and 3WC, which are subscribed to in both the source and target zones, this means that (CW, "Source") must precede (3WC, "Source"), and (3WC, "Target") must precede (CW, "Target"). Fig. 3a shows why.

If, contrary to the principle, 3WC takes precedence over CW in the source zone, then the configuration in Fig. 3a arises when the subscriber has used 3WC to make a second call while one is already in progress. The second outgoing call from 3WC does not go through a CW box, because there is only one such box bound to LI, and it is already fully occupied as far as outgoing calls are concerned. If the first party to which the LI was connected hangs up, then there will be no CW box left in the usage. The second principle then applies: externally initiated modifications of usages are strictly prohibited by the DFC architecture. The CW box, lost from the usage when the first customer call was torn down, cannot be spliced into the usage to become available in the second call. The effect is that subscribers to 3WC are sometimes deprived of CW functionality.

In contrast, Fig. 3b and 3c show configurations that arise when the precedence relation follows the first principle. 3WC is a free box, so a separate instance can appear on each fork of the usage.

It is well known that CW and 3WC interact because they both use flashes from the proximate line interface as their control command. This situation invokes a third DFC principle, which is that when two boxes compete for an out-of-band signal, the box closer to the source of the signal has priority access to it. The closer box can respond to the signal and absorb it, or it can ignore the signal and pass it on. Because of the first principle, CW has priority access to the flash, which fortunately is what we want (at least most of the time). The configuration in Fig. 3d arises when the line interface first makes an outgoing call, uses 3WC to create a conference (flashes are forwarded to 3WC1 by the transparent CW box), and then receives an incoming call through CW.

The behavior produced by always giving CW priority access to flashes may or may not be considered acceptable. Even in Fig. 3d some aspects of the user interface are awkward, and a configuration such as that shown in Fig. 3e, in which all boxes have reached their full potential, is not achievable.

If the default priority scheme for access to flash signals does not yield acceptable behavior, and if enriching the command vocabulary (usually the strategy that yields the best user interface) is impermissible, then it becomes necessary to specify explicit cooperation between two features. It is possible within the DFC architecture to use special status messages between the CW and 3WC boxes to give 3WC priority access to flashes at the appropriate times [25]. Of course, introduction of such special messages is highly undesirable because it compromises feature modularity and independence; but in the circumstances envisaged, such compromise is unavoidable whatever the underlying architecture: the desired behaviors of the two features are in conflict.

4.7 Using the Direct Routing Command

The Emergency Break-In feature (EBI) makes use of the direct routing command in the setup message. An emergency caller, such as the Fire Department, subscribes to EBI as a source feature. An EBIE box is included in the source zone of every outgoing call made by the emergency caller. This EBIE box, perhaps after checking the user's authorization, places a call with a direct routing command. The configuration that results when the callee subscriber is already engaged in a usage is shown in Fig. 4.

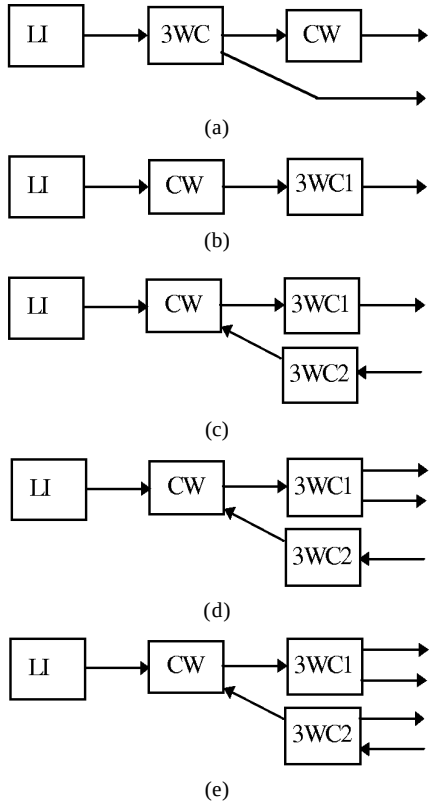


Fig. 3. Configurations of CW and 3WC.

A customer call is connected between the subscribers at LI1 and LI2. Both LI1 and LI2 have bound EBI boxes. The emergency service places an emergency call to the subscriber at LI2; this call is routed directly from the EBIE box of the emergency service to the EBI box bound to LI2, and that EBI box accepts the call and places the call from LI1 on hold. Evidently a regular subscriber's EBI box must have the highest precedence, ensuring that it is immediately adjacent to the LI box to avoid potential interference from other features.

4.8 Rerouting for a Dialed Feature

The Directory Link feature (DL) recognizes when a customer has dialed a directory assistance number. It tries to recognize the number announced by directory assistance, and offers to connect the customer to that number directly.

DL is a dialed feature, included in the usage whenever the number dialed (or speed-called) has the form of a directory assistance DN. In the North American long-distance network this form is aaa-555-1212, where aaa is the area code of the destination subscriber for which assistance is needed. A DL feature box has a callee port *p1* and a caller port *p2*. On receiving a call on p1, DL engages in a dialogue in which it offers the DL service to the calling subscriber for a small charge. If the subscriber refuses the service offer, the DL box places a call to directory assistance on port *p2* and behaves transparently from that point onwards.

If the subscriber accepts the service offer, the call to directory assistance is made on port *p2*. The DL box then monitors the incoming voice signal on *p2*, and will normally detect the number announced by the directory assistance equipment. It then tears down the call on *p2*, and places a new call with the announced number as the dialed string. If DL cannot detect the announced number, it plays a message to the caller (apologizing and saying that no charge will be made for the failed service) and tears down the calls on *p1* and *p2*.

The performance of DL depends on the intelligibility of the directory assistance announcement of the number and on the quality of the speech analysis used by the DL box. In principle DL can work on any of the following announcements:

- "The number you require is xxx-ssss." DL places a call to aaa-xxx-ssss, having stored the value aaa from the setup message on *p1*.
- "The number you require is 800-ppp-qqqq." DL places a call to the announced 800-number. Since this 800-number is the dialed string of the newly placed call, an 800 box will be included in the dialed zone of the new segment.
- "The number you require is in area code bbb." Please call directory assistance on bbb-555-1212." DL places a call to the announced directory assistance number. Since this number is the dialed string of the newly placed call, a fresh DL box will be included in the dialed zone of the new segment, and the DL service will be offered again in the new call.

4.9 Interactions Arising from In-Band Signaling

The use of voice recognition and other in-band signaling can easily give rise to awkward interactions. Whereas a feature box can absorb an out-of-band signal and prevent it from reaching other boxes simply by not passing it on, it cannot absorb an in-band signal in the same way unless it has previously disconnected the voice channels.

A technique that can be used in the DFC environment is to inhibit recognition of in-band signals by sending out-of-band messages to indicate to other boxes that they should not recognize certain in-band signals until further notice. Call connection between adjacent boxes in a usage uses a pair of voice channels, one in each direction. When a box is about to monitor the signal arriving on an incoming channel from an upstream box, it sends an *inhibit* message to its neighbor or neighbors in the downstream direction, and vice versa; when it ceases to monitor the signal it sends a complementary *uninhibit* message in the same direction. If all boxes observe the obvious discipline, each allowing a

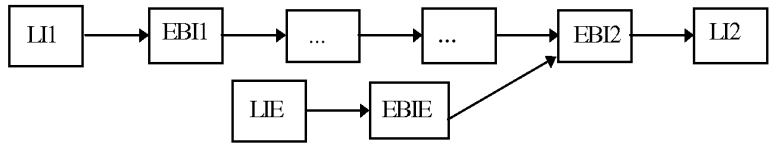


Fig. 4. Direct routing.

box from which an *inhibit* is received to take precedence in monitoring voice signals from the corresponding direction, the in-band signaling difficulty can be overcome.

This discipline allows the same precedence to be enforced for in-band signals as for out-of-band signals. It is also possible to enforce the opposite precedence or even a mixture, at the cost of some complexity and reduction in feature independence. To establish any precedence by such a discipline requires a taxonomy of in-band signals, and the definition of field values in the *inhibit* and *uninhibit* messages to represent the signal classes.

4.10 Addressable Boxes and Dialed Features

Addressable boxes are used only where all customers who dial a given DN must be connected to the same box. By contrast, a dialed feature is merely one that is applied according to the dialed string. Two contrasting examples will make the point clearer.

800 Service (800) is a dialed feature, configured into any usage in which the dialed string begins with the characters '1-800'. But it is not an addressable box: each call dialing an 800 number is routed to a temporarily dedicated free 800 box, which determines a target DN from the operational data of the feature—namely, the relation recording the mapping from 800 numbers to DNs. To make 800 an addressable box would mean having one feature box for each 800 number, and routing all calls to the number through that particular box. While this scheme would have the advantage that the box could contain the maplet for the number, and so would not need to access operational data, it would have a far weightier disadvantage: it would require the box to have enough ports to serve as many customers as could be calling the 800 number at any one time and a corresponding box program able to handle the interleaving of their call protocols.

By contrast, Large-Scale Conferencing (LSC) requires the use of a number of addressable boxes. When a conference is booked, a box of sufficient size to accommodate all participants is assigned; participants who will take the initiative in joining the conference are given this number to call, while other participants may wait for the box to call them and so will not need to know the number.

4.11 Disconnected Boxes

Some features are provided by boxes that are temporarily disconnected from all other boxes while engaged in their tasks. Consider, for example, a Wake-Up Call service (WUC). WUC is a dialed feature provided by a free box. On receiving an incoming call the box collects the caller's DN and the desired wake-up time from the caller in local variables. It then tears down the call, and waits until the wake-up time is indicated on the real-time clock to which it has access. During this period of waiting, the box is not connected to any other box; but it is occupied, and not available to service another wake-up call. At the expiry of the period it places the wake-up call to the subscriber whose DN is stored in its local variable.

5 Summary

5.1 Simplifications Adopted

In designing and presenting the DFC technology we have adopted several simplifying assumptions in order to clarify and focus our work. Although in themselves these simplifications are unrealistic, we believe that abandoning them would not invalidate the architecture, but merely force the introduction of additional detail that would obscure—but not vitiate—the main ideas. The chief of these simplifications are the following.

- We restrict our consideration to analog phones. The use of ISDN phones would introduce a richer vocabulary of messages that could flow to and from LI boxes, and a richer feature set. But we see no reason to think that the additional features would not fit comfortably into the DFC architecture.
- We assume that each line interface is associated with a single DN, and that DNs are not shared.
- We assume that the interface boxes are all interfaces to telephones, not to trunks. This allows us to ignore the addressing complications that arise with trunks (the relation *trunk* $\leftrightarrow$ *DN* is many-to-many), and also the race condition ('trunk glare') that can arise when it is necessary to seize a trunk.

We have also adopted some other simplifications that will take further work to abandon. In particular:

- We do not discuss billing here. However, we have given some consideration to billing issues, and it appears to us that the DFC architecture may offer a helpful environment for specifying billing features.
- We do not consider provisioning, either of subscription data or of feature operational data. Quite apart from the additional detail involved, there are issues of coordination between provisioning and operation. We see no reason to think that these issues would be harder to handle in a DFC setting.
- We assume the absence of resource contention. Although feature boxes are freely conceived as containing dedicated devices such as conference bridges and signal processors, we assume that boxes are always available when needed. We make the same assumption about signaling and voice channels.
- We have ignored broadband and multimedia telecommunications services.

An AT&T Technical Memorandum [25] discusses some aspects of the DFC architecture under less restrictive assumptions.

5.2 Related Work

The most common research approach to the problem of feature interaction is the application of formal verification techniques to system specifications, with the goal of detecting all undesired feature interactions [21]. Velthuijsen concludes:

> None of the approaches have progressed far beyond a mere proof-of-concept result. The approaches show that with the right descriptions and techniques it is indeed possible to detect certain interactions. But the trick is to come up with the right

> descriptions: the right network and service specifications, and above all, the right properties. This appears very hard to do well and presents for the time being a more urgent challenge than the application of yet another specification formalism to the same general approach.

We agree with Velthuijsen and are trying to find the right (modular, complete, and comprehensible) service specifications. Formal verification and detection of unforeseen interactions are activities we hope to pursue later, building on this new foundation [25].

There are other architectural approaches to service specification, the most well-known being the Intelligent Network (IN) architecture [1], [10], [13], [18], [20]. The major limitation of the IN architecture is that its conceptual model is strongly oriented towards two-party calls, and is difficult to extend to multiparty services. We regard the ease with which DFC handles nonlinear usages and multiparty services as an important advantage, as explained below.

More recently other architectures for telecommunications services have been proposed, for example [15], [22], [26]. In these architectures a typical component is an agent representing a subscriber, resource, common function, or type of connection. These agents determine their call-processing behavior by negotiating with each other, sending messages to each other as needed. The most important difference between these agent-based architectures and the DFC architecture is that agent-based architectures impose little in the way of limits or structure on communication among agents. For example, in [26], communication between agents is presented only loosely and informally, making it hard to see what constraints are imposed by the architecture described. For this reason, it will be very difficult to exploit the modularity of the decomposition into agents to understand feature interactions or determine global system properties.

Another characteristic of the DFC approach is that features are distinct, first-class modules. Many other formalizations of feature behavior have partitioned features into syntactic modules, for example distinct sets of rules [4], [8], [11], [19]. But a set of rules shares unrestricted state with other rules, so the syntactic constraint actually places very little constraint on semantic interaction among modules. All interactions among DFC modules, on the other hand, are strictly limited by the architecture.

The specification technique of [2], [5] is based on layered state-transition machines; it is similar to ours in having feature modules communicating by means of a fixed protocol. Their feature modules may sometimes require knowledge of each others' states, so they are more closely coupled than DFC feature boxes. DFC feature boxes achieve a much looser coupling by means of the pipe-and-filter arrangement, in which each box has a great deal of independent control over the voice channels passing through it.

There are as many possible comparisons to related work as there are research projects on feature interaction, especially since even research that is focused narrowly on detection must start with some specification formalism. In the previous paragraphs we have given a representative sample of comparisons—representative with respect to both similarities to and differences from the DFC architecture. We know of no specification approach that is closer to the DFC architecture than those we have cited.

5.3 Advantages of DFC

The treatment of features as distinct components is not in itself new. The key novelty of DFC is the choice of an architecture in a dynamic pipe-and-filter style, with the advantages that this brings.

One important advantage already mentioned is the ease with which nonlinear usages can be handled. and represented in graphs. The dynamic configuration and reconfiguration of a usage can be complex. For example, Fig. 5 shows a progression through three configurations.

In Fig. 5a there are two clearly distinct usages: LI1 is connected to LI2, and LI3 to LI4; in Fig. 5b these usages have been joined because LI1 has placed a further call to LI4; in Fig. 5c both LI1 and LI3 have dropped out, leaving LI2 connected to LI4. Notice that the final connection of LI2 and LI4 depends on continuing use of the 3WC box originally introduced in LI1's call to LI2; this causes no difficulty because 3WC is a free box.

This complexity of configuration is easily handled in DFC because of its distributed nature. Although the notion of a *usage* is convenient for informal explanation of DFC behavior, it has no formal status in DFC. There are no usage identifiers to be allocated or manipulated; nothing in the DFC architecture depends on identifying and distinguishing individual usages over time. Only individual boxes and individual internal calls are signficant: DFC configurations arise solely from the internal featureless calls, each involving just two parties. DFC can, therefore, exploit the simplicity of POTS, while avoiding the traditional, but obscure and difficult, elaboration of the notion of a *call*, which is central in some approaches. Avoiding this notion eliminates many complications and restrictions that flow from it, and simplifies the specification of feature behavior.

We believe that the DFC scheme of feature communication by featureless internal calls offers several advantages in feature specification, analysis, and implementa-

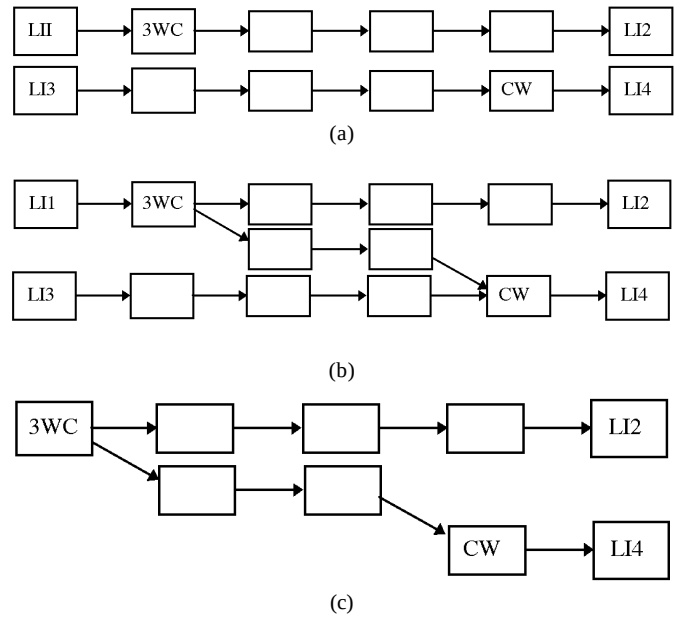


Fig. 5. Usage configurations.

tion. The primary advantage in feature specification is the high degree of separability among features. To a very great extent, at least an initial specification of a feature can be made without consideration of other features that may be present, by assuming that any other feature boxes in the usage are in a transparent state. (Of course, there are some general concerns that must always be taken into account. For example, any feature may find itself in a usage within whose lifetime several customer calls are placed and answered. The OCS feature, for example, must therefore be specified to check a sequence of customer calls, not just one.)

Another specification advantage is the clear separation between the behavior of individual features and the underlying mechanism for their composition. As was pointed out above, the DFC router does not provide any feature functionality, even for features that might perhaps be viewed as essentially routing features, such as 800 Service or CFA. Equally, the individual features play no direct part in determining the routing. Their contribution to the routing behavior is at the more abstract level of indicating that further routing is to be based on a new value of the *source* or *target* field in the setup message.

The chief advantage in analysis is that interaction between feature boxes is constrained in a well understood way closely analogous to the interaction between caller and callee in a traditional POTS call. Just as in the approach of [17], behavior of individual boxes and their interactions is susceptible to model checking techniques; properties of the interaction of the caller and callee protocols given in Section 3.10 with the DFC switch protocol and with each other have been checked in Spin [16]. The explicit treatment of voice channels in internal calls allows feature box states and behaviors to be analyzed in terms of voice processing states as well as in terms of sending and receiving out-of-band signals.

Finally, the DFC architecture appears to be relatively easy to implement on a wide variety of physical telecom munications architectures [25]. There is an obvious corre spondence between a DFC usage and a voice path through the switches and trunks of a physical network.

Furthermore, we have consciously avoided choices in the definition of the DFC architecture that are likely to be expensive to implement. For example, the constraint that source, dialed, and target zones are configured in that order simplifies the assignment of feature functionality to network nodes. Although the three zones are routed together in the virtual architecture, the routing of the three zones can easily be distributed for the purpose of efficient signaling. Another example is that each feature box has the narrowest possible scope of action: for instance, Wake-Up Call, and 800 Service features are provided by free boxes serving one usage at a time. Engineers are free to implement these and other feature boxes as individual physical components or as processes on a large platform serving many usages concurrently.

5.4 Further Work

So far we have programmed sample feature boxes in Promela. Our highest-priority task is the development of a domain-specific language that makes feature boxes easy to program and easy to analyze for unexpected feature interactions and other interesting properties.

At the same time, we are also working on:

- the extension of the DFC architecture to aspects of telecommunications that it does not currently cover;
- the systematic implementation of the DFC virtual architecture on a variety of physical architectures; and
- investigation of further techniques for abstraction, classification, and verification of features.

APPENDIX A—RELATION SYMBOLS

Appendix A explains the relation symbols used in this paper. A, B, and C are sets of any elements; R and S are relations.

Relation Symbols	Definition
$A \leftrightarrow B$	The general relation, consisting of any set of pairs (Ai, Bj).
$A \rightarrow B$	A total function. Each element Ai of A occurs in exactly one pair (Ai ,Bj).
$A \nrightarrow B$	A partial function. Each element Ai of A occurs in at most one pair (Ai, Bj).
$A \twoheadrightarrow B$	A total surjection. Each element Ai of A occurs in exactly one pair (Ai, Bj). Each element Bj of B occurs in at least one pair (Ai, Bj).
$A \rightarrowtail B$	A total injection. Each element Ai of A occurs in exactly one pair (Ai, Bj). Each element Bj of B occurs in at most one pair (Ai, Bj).
ran R	The range of R. If R consists of pairs (Ai, Bj), **ran** R is the set of Bj occurring in R.
dom R	The domain of R. If R consists of pairs (Ai, Bj), **dom** R is the set of Ai occurring in R.
R~	The inverse of R. If R consists of pairs (Ai, Bj), R~ is the set of pairs (Bj, Ai) such that (Ai, Bj) is in R.
$C \lhd R$	R domain-restricted to C. If R consists of pairs (Ai, Bj), $C \lhd R$ is the set of pairs (Ai, Bj) occurring in R such that Ai is in C.
$R \rhd C$	R range-restricted to C. If R consists of pairs (Ai, Bj), $R \rhd C$ is the set of pairs (Ai, Bj) occurring in R such that Bj is in C.
$R (\|C\|)$	The relational image of C under R. If R consists of pairs (Ai, Bj), $R (\|C\|)$ is the set of elements Bj such that at for least one Ai (Ai, Bj) occurs in R and Ai is in C.
$R \fatsemi S$	The relational composition of R and S. If R consists of pairs (Ai, Bj) and S consists of pairs (Bk, Cn), $R \fatsemi S$ consists of the pairs (Ai, Cn) such that for at least one Bj (Ai, Bj) is in R and (Bj, Cn) is in S.

ACKNOWLEDGMENTS

We thank the editor, Yow-Jian Lin, and the anonymous reviewers for many helpful suggestions.

REFERENCES

[1] I. Aggoun and P. Combes, "Observers in the SCE and SEE to Detect and Resolve Service Interactions," P. Dini et al. eds, *Feature Interactions in Telecommunication Networks,* vol. 4, pp. 198-212. IOS Press, 1997.

[2] P.K. Au and J.M. Atlee, "Evaluation of a State-Based Model of Feature Interactions," P. Dini et al., eds, *Feature Interactions in Telecommunication Networks,* vol. 4, pp. 153-167. IOS Press, 1997.

[3] *Feature Interactions in Telecommunications Systems,* L.G. Bouma and H. Velthuijsen, eds. Amsterdam: IOS Press, 1994.

[4] J. Blom, B. Jonsson, and L. Kempe, "Using Temporal Logic for Modular Specification of Telephone Services," [3], pp. 197-216, 1994.

[5] K.H. Braithwaite and J.M. Atlee, "Towards Automated Detection of Feature Interactions," [3], pp. 36-59, 1994.

[6] E.J. Cameron, N.D. Griffeth, Y.-J. Lin, M.E. Nilson, W.K. Schnure, and H. Velthuijsen, "A Feature Interaction Benchmark for IN and Beyond," [3], pp. 1-23, 1994.

[7] *Feature Interactions in Telecommunications Systems III,* K.E. Cheng and T. Ohta, eds. Amsterdam: IOS Press, 1995.

[8] P. Combes and S. Pickin, "Formalisation of a User View of Network and Services for Feature Interaction Detection," [3] pp. 120-135, 1994.

[9] *Feature Interactions in Telecommunication Networks,* vol. 4, P. Dini, R. Boutaba, and L. Logrippo, eds. Amsterdam: IOS Press, 1997.

[10] J.M. Duran and J. Visser, "International Standards for Intelligent Networks," *IEEE Comm.,* vol. 30, no. 2, pp. 34-42, Feb. 1992.

[11] A. Gammelgaard and J.E. Kristensen, "Interaction Detection, A Logical Approach. [3], pp. 178-196, 1994.

[12] D. Garlan and M. Shaw, "An Introduction to Software Architecture," V. Ambriola and G. Tortora, eds., *Advances in Software Eng. and Knowledge Eng.*, pp. 1-39. World Scientific, 1993.

[13] J.J. Garrahan, P.A. Russo, K. Kitami, and R. Kung, "Intelligent Network Overview," *IEEE Comm.,* vol. 31, no. 3, pp. 30-36, Mar. 1993.

[14] N.D. Griffeth and Y.-J. Lin, "Extending Telecommunications Systems: The Feature-Interaction Problem," *Computer,* vol. 26, no. 8, pp. 14-18, Aug. 1993.

[15] N.D. Griffeth and H. Velthuijsen, "The Negotiating Agents Approach to Runtime Feature Interaction Resolution," [3], pp. 217-235, 1994.

[16] G.J. Holzmann, "Design and Validation of Protocols: A Tutorial," *Computer Networks and ISDN Systems,* vol. 25, pp. 981-1,017, 1993.

[17] F.J. Lin and Y.-J. Lin, "A Building Block Approach to Detecting and Resolving Feature Interactions," [3], pp. 86-119, 1994.

[18] J. Kamoun, "Formal Specification and Feature Interaction Detection in the Intelligent Network," Dept. of Computer Science, Univ. of Ottawa, Ontario, 1996.

[19] T. Ohta and Y. Harada, "Classification, Detection and Resolution of Service Interactions in Telecommunication Services," [3], pp. 60-72, 1994.

[20] S. Tsang and E.H. Magill, "Behavior Based Run-Time Feature Interaction Detection and Resolution Approaches for Intelligent Networks," P. Dini et al., eds, *Feature Interactions in Telecommunication Networks,* vol. 4, pp. 254-270. IOS Press, 1997.

[21] H. Velthuijsen, "Issues of Non-Monotonicity in Feature-Interaction Detection," K.E. Cheng and T. Ohta, eds, *Feature Interactions in Telecommunications Systems,* vol. 3, pp. 31-42. IOS Press, 1995.

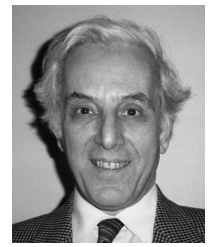
[22] M. Weiss, T. Gray, and A. Diaz, "Experiences with a Service Environment for Distributed Multimedia Applications," [9], pp. 242-253, 1997.

[23] J. Woodcock and J. Davies, *Using Z: Specification, Refinement and Proof.* Prentice Hall Int'l, 1996.

[24] P. Zave, "Feature Interactions and Formal Specifications in Telecommunications," *Computer,* vol. 26, no. 8, pp. 20-30, Aug. 1993.

[25] P. Zave and M. Jackson, "The DFC Virtual Architecture: Scenarios for Use and Plans for Future Work," AT&T Research Technical Memorandum HA6164000-971202-18TM, Murray Hill, N.J., Dec. 1997.

[26] I. Zibman, C. Woolf, P. O'Reilly, L. Strickland, D. Willis, and J. Visser, "Minimizing Feature Interactions: An Architecture and Processing Model Approach," [7], pp. 65-83, 1995.



Michael Jackson has been active in software development methodology for more than 35 years. This work is described in his books: *Principles of Program Design* (Academic Press 1975); *System Development* (Prentice Hall 1983); and *Software Requirements and Specifications* (Addison-Wesley/ACM Press 1995); and also in many papers, books, journals, and conference proceedings. Having established his own company in 1970, he now works as an independent consultant in London and as a part-time researcher at AT&T Laboratories–Research in Florham Park, New Jersey. He has held several visiting posts at various universities in England and Scotland. Jackson is a member of the IEEE Computer Society.

Pamela Zave received the AB degree in English from Cornell University, Ithaca, New York, and the PhD degree in computer sciences from the University of Wisconsin at Madison. She began her career as an assistant professor of computer science at the University of Maryland, College Park. Since 1981, she has been with AT&T Laboratories–Research, and is now a member of the Network Services Research Laboratory. Dr. Zave's research interests include formal methods for software development (particularly telecommunication software), compositional and multi-paradigm specification techniques, and requirements engineering. She was the designer of the executable specification language PAISLey, has published ~60 technical papers, and has given numerous invited lectures. She is also a member of IFIP Working Groups 2.3 (Programming Methodology) and 2.9 (Requirements Engineering). Dr. Zave is a member of the IEEE Computer Society.