An Architecture for Three Challenging Features

Pamela Zave
AT&T Laboratories—Research
Florham Park, New Jersey, USA
pamela@research.att.com

Abstract— **The ECLIPSE project shows how concerns related to the complex behavior of telecommunication systems can be separated from concerns related to the IP implementation of that behavior. Within this context, the behavioral problems of Location and Identification, Switching and Spontaneous Conferencing, and Mail are addressed. Each of these features is challenging because it is not clear which functions should be included within its boundary, because it encompasses many possible behavioral variations and conflicting requirements, because it interacts with many other features, and because (like all features) it must be extensible. For each feature, the paper proposes a boundary and presents an architecture for performing all the functions within the boundary, with all their behavioral variations. The paper also shows how the composition of these feature architectures is extensible and reveals potential feature interactions. Bad feature interactions can be prevented, and good feature interactions preserved, by minor adjustments to the feature composition.**

Keywords— **multimedia telecommunication services, requirements, feature interaction, user interfaces, personal mobility, conferencing, mail**

I. Introduction

TELECOMMUNICATION services are now migrating to IP networks. The good news is that designers can offer customers whatever they might need or want. In this new context, services are unconstrained by the PSTN legacy or by inflexible technology.

The bad news is the same. Designers of telecommunication features in this new, unconstrained context will be faced with a truly bewildering range of surprisingly difficult choices. They will find that the work required to choose and justify the externally observable behavior of a new feature—in every possible situation—is a large fraction of the total work required to create it.

There are three fundamental reasons why these choices are so difficult. First, telecommunication services are developed incrementally, adding features over time (this has been true of telephony and many other complex application domains, and there is no reason why IP telecommunications should be different). Although decisions should be made with future extensibility in mind, they must be made at a time when future requirements cannot be predicted.

Second, many worthy goals conflict. Everyone expects greatly enhanced functionality, yet enhanced functionality conflicts with ease of use, because complex functionality requires a complex user interface. Enhanced functionality also conflicts with universality of communication, because some functions require the cooperation of all communicating parties. Such functions can only be used by a customer to communicate with other parties whose functions are similarly enhanced.

Third, even when telecommunication features are conceived as being independent, they necessarily interact, which means that they modify or influence one another in determining the system's overall behavior. To manage feature interactions well, designers must predict potential feature interactions, decide which are desirable and which are undesirable, and engineer feature composition so that only the desirable interactions occur. Unfortunately, there is a great deal of experience to prove that people perform these tasks poorly [15], and that they are in fact intrinsically difficult [3], [5], [6], [7], [11].

This paper addresses these problems for three important features. It shows that each of the three features—Location and Identification, Switching and Spontaneous Conferencing, and Mail—is an appropriate unit of development, in the sense that it satisfies a set of closely related requirements, best considered together. A software architecture for each feature shows how the requirements can be satisfied straightforwardly. In addition to shortening the path from requirements to implementation, the architecture alleviates each of the three problems presented above.

First, the architecture guarantees extensibility. It is a specialization or application of the Distributed Feature Composition architecture for describing telecommunication services [9], [16], [17], [19]. DFC has been proven successful at providing feature modularity and feature compositionality, so that the problems of extending a DFC system with new features are minimal.

Second, the architecture helps designers make requirements trade-offs. The architecture for each feature is somewhat general-purpose, and accommodates a range of behavioral variations. Thus designers can experiment with the details of user interfaces and detailed behaviors for

each feature, without altering the overall system organization.

Third, the architecture helps manage feature interaction. DFC is well-suited to predicting potential feature interactions and to engineering feature composition so that all of the desirable interactions can occur, while none of the undesirable ones can. Obviously, the feature architecture here inherits these capabilities from DFC.

II. Basic Service and Baseline Architecture

Figure 1 illustrates the basic multimedia service to which the three features are added. The features are intended to work with all telecommunication devices, from the large (PCs) to the small (cellphones and pagers). A call includes one two-way signaling channel and any number of two-way media channels. For concreteness, this paper focuses on just two communication media: voice and text. For simplicity, it ignores gateways to other networks.

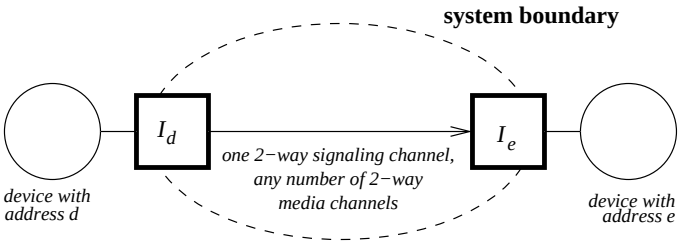

Fig. 1. Basic service.

The baseline architecture is DFC. The following overview of DFC provides just enough information to understand the architecture of the three features.

In Figure 1, I_d and I_e are *interface boxes*. These modules translate between the external protocols of the devices and the internal DFC protocol. An interface box is persistent even if the connection between the device and the network is not, so it is always available to represent the device to the network.

In DFC the term *customer call* is used informally, referring to an attempt by a user to communicate. A customer call generates and is responded to by a *usage*, which is a dynamic assembly of *boxes* and *internal calls*. A *box* is a concurrent process, and is either an interface box or a feature box. An *internal call* is a featureless connection between two ports on two different boxes. In Figure 1, there are no features, so the entire usage consists of the interface boxes and one internal call from I_d to I_e.

All the subsequent figures illustrate usages with *feature boxes* in them. In any snapshot of a DFC system, the usages can be defined formally as connected graphs of boxes and internal calls. Since usages change shape, merge, and split over time, however, the relationship between a usage and a customer call, and for that matter the concept of a customer call, cannot be formally defined.

Each internal call begins with a *setup phase* in which the initiating port sends a setup signal to a DFC router, and the DFC router chooses a box and forwards the signal to it. The receiving port completes the setup phase with a signal back to the initiating port. From that time until the *teardown phase,* the call exists and has a two-way signaling channel. The media channels of the call, which can be initiated from either port, are opened and closed explicitly by signals on the signaling channel.

When a feature box does not need to function, it can behave *transparently.* For a box with two ports, both of which are engaged in calls, transparent behavior is sending any signal received from one port out the other port, and connecting the media channels in both directions. The two calls will behave as one, and the presence of the transparent box will not be observable by any other box in the usage.

Having full control of all the calls it places or receives, a feature box has the autonomy to carry out its function without external assistance. It can re-route or disconnect internal calls, process media streams, and absorb or generate signals.

Figure 2 shows how a well-known desirable feature interaction (see, e.g., [4]) is accomplished in DFC. The user of the device *d* is attempting to call *e*, creating a usage with three feature boxes. F_3 has placed an internal call routed to I_e, which failed because I_e's single port is already occupied with another internal call. As a result, F_3 sends the status signal *unavailable* upstream. Any upstream feature box that receives it can treat the *unavailable* condition and absorb the signal, or can ignore the *unavailable* condition and propagate the signal further upstream. Thus downstream *unavailable* treatments have priority over upstream ones.

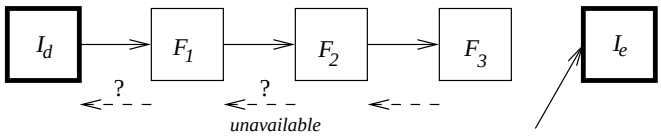

Fig. 2. How *unavailable* treatments interact.

DFC routing is an algorithm that uses provisioned configuration, subscription, and precedence information to determine the boxes in a usage. The setup signal of each internal call is sent separately to a *DFC router* for routing to another box, which may be a feature box (if more features need to be applied) or an interface box (if all relevant fea-

tures have been applied). Necessary information about the history of the usage is carried in the setup signal, so that the routing algorithm itself can be stateless.

The default routing situation is as follows. When the user of a device initiates a customer call, the device interface issues a *new call* whose setup signal contains *source* and *target* addresses. The source address determines a *source zone*, which is a sequence of feature box types subscribed to by the source address. Similarly, the target address determines a *target zone*. The usage unfolds as a linear chain of boxes and calls, eventually containing a box of each type in the source zone, followed by a box of each type in the target zone. When both zones are exhausted, the next internal call is routed to the target interface box.

Most feature boxes are *free*; when the router needs a free box, it simply routes to a new, anonymous instance of the box type. Some feature boxes, however, are *bound*; for each address subscribing to a bound box type, there is exactly one, persistent, addressable instance of the box type. Bound boxes allow joins in usages, as an internal call can be routed to a bound box that is already engaged in other internal calls. In figures, for example 4 and 5, bound boxes and interface boxes are drawn with heavier lines than free boxes.

In the default situation, each feature box makes a *continuation call*, which is an outgoing call using exactly the same setup signal that it received as part of its incoming call. This is the mechanism that causes default routing to unfold. In contrast, feature boxes can also affect routing by making various structured changes to the setup signal of an incoming call before using it to place an outgoing call. Each routing variation is used somewhere in this paper, and is explained where used.

Most features are implemented by one type of feature box. Some features, however, require more than one box type; addresses subscribe to box types depending on which role they are playing with respect to the feature. Boxes *of the same feature* can communicate through, and maintain persisent data in, global *operational data*. Since the features in this paper are complex ones, they use more feature box types than average features.

The DFC architecture is actually a domain-specific adaptation of the pipes-and-filters architecture [13]. Its modularity is exactly the same kind as that claimed for pipes and filters in general.

III. The ECLIPSE Project

The ECLIPSE project has produced an IP implementation of DFC [1].

The philosophy of the ECLIPSE project is to separate the concerns of behavior and optimization. Feature behavior and interactions are described and analyzed in terms of DFC, so that description and analysis can benefit from DFC's modularity, abstraction, and formality. Meanwhile, the ECLIPSE implementation incorporates many optimizations that allow feature boxes to function efficiently in an IP setting. These optimizations are invisible to feature designers, and apply equally to all features developed within the DFC framework.

The current version of ECLIPSE is fully distributed, so that feature boxes can be located anywhere in a network. It incorporates an optimization that allows media streams to travel end-to-end rather than following the signaling path, and another optimization that supports efficient distribution of all routing data and most operational data. Future optimizations will address signaling latency, efficient maintenance of data consistency, and signaling/media synchronization. Separation of concerns allows us to improve these optimizations incrementally, without disturbing feature development or the execution of existing features.

Because of this separation of concerns, it is meaningful to discuss the behavior of a telecommunication system without constant reference to its implementation. This paper takes advantage of this freedom, deferring the implementation issues to other venues.

IV. The Features

A. Location and Identification

A.1 Requirements

It is attractive for a telecommunication system to offer, in addition to addresses tied to devices, *personal addresses* associated uniquely with people. A personal address can subscribe to feature boxes, own private data, and be the source or target of calls.

The concept of personal addresses leads directly to two requirements:

1. When the target of a call is a personal address, it is necessary to find a device where the person is located, and direct the call to that device [location].
2. When a user is connected to the system through a device, it is necessary to identify that user, and to ensure that the user only has access to his own subscribed feature boxes and personal data [identification].

These two requirements are intimately related and best addressed together. Briefly, the location requirement demands an answer to the question, "Which device is this person using?" The identification requirement demands an answer to the dual question, "Which person is using this device?"

The relationship between location and identification can best be explained in terms of shared customer

data. Location relies primarily on the relation *located_near(person,device)*. Identification relies primarily on the relations *has_access_to(person,device)*, representing long-term possession or authorization rather than current location, and *authenticates(password,person)*.[1] Nevertheless, identification can also use *located_near* when several people have access to a device and it is too inconvenient to demand a password. Also, a location feature might bear the responsibility of guaranteeing that the correct person has been reached, which it can only do by use of *authenticates*. Also, a location feature might use *has_access_to* to create a menu from which a user can choose in updating his location.

The concept of Location and Identification encompasses the IP-oriented definition of personal mobility [12], in which a user becomes available to the network through registration or login. It also encompasses the communication style of traditional (wireline) telephony, in which some devices are connected to the network at all times. A good feature is actually much harder to design in the telephony context, because users do not expect to have to register before they can communicate, and because some devices are shared simultaneously by multiple people.

A.2 Core Functions and Software Architecture

DFC has *mobile addresses*, addresses not permanently associated with any device. DFC mobile addresses can be used (among other things) as personal addresses.

Loc is a free box type of this feature, subscribed to in the target zone by each personal address *p* (Figure 3). Upon receiving an incoming call, *Loc* uses the operational data to perform the core location function of determining the device *d* where *p* is presumably located, and placing a continuation call with *target = d*.

When a feature box in the target zone changes the target in a continuation call, routing is automatically affected. The DFC router discards the remainder of the old target zone (feature box types that have not yet been routed to) and begins routing to the target zone of the new target.

Once a user has been reached through device *d*, *Loc* behaves transparently. By doing this, it performs the core function of giving the user access to the feature boxes subscribed to by *p*, and through them to the private data of *p* (only feature boxes routed to on behalf of *p* can access *p*'s slice of the feature operational data).

Note that *Loc* must be the *last* feature box in the target zone of *p*, so that all personal feature boxes are guarded by it, and none are skipped when the usage is retargeted to *d*.

[1] The type *password* is intended to include fingerprints, voice prints, and other means of bio-identification.

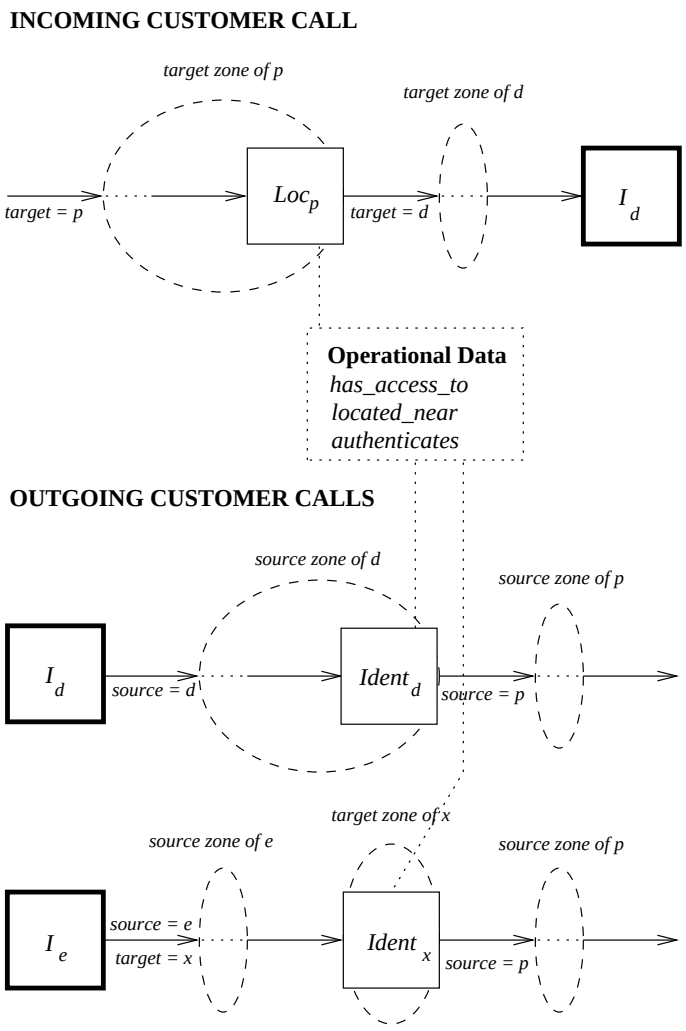

Fig. 3. Architecture of the Location and Identification feature.

Ident is a free box type of this feature, subscribed to in the source zone by each participating device *d* (Figure 3). Upon receiving an incoming call, *Ident* uses the operational data to perform the core identification function of determining which personal address the user owns, if any. If there is a relevant personal address *p*, it places a continuation call with *source = p*. Just as in the target zone, this will cause the router to throw away the rest of the current source zone, and to route the continuation call to the first feature box of the source zone of *p*, giving the user access to the feature boxes and data of *p*. If there is no relevant personal address, the box places a continuation call with no changes, thus continuing with the source-zone feature boxes of *d*.

A *participating device* is intended for use by people with personal addresses, and thus subscribes to *Ident*. But the owner of a personal address might be using a public device with no such subscription, such as *e* in Figure 3. In

such a case the only way to invoke *Ident* is to dial a special address *x* that subscribes to *Ident* in the target zone. In this context, if identification succeeds *Ident* collects a real target address from the caller and places a *new* outgoing call, so that routing to the source zone can begin again.

The relation *located_near* is highly dynamic. New location information can be entered manually through *Loc* or *Ident*, once the feature box has identified its user.[2]

A.3 Behavioral Variations

A wide variety of identification policies can be built into different versions of this feature. At one end of the spectrum, *located_near* and *has_access_to* might be unambiguous enough, and trusted enough, to identify the user in all cases. For example, if there is a device such as a cellphone to which only one person has access, the feature might assume that any user of the cellphone is that person. At the other end of the spectrum, every new use of a device requires a password. Note that if *Loc* demands a password, it does so after a user has answered the call to the device interface, but before allowing the user to exchange signals with the personal feature boxes of *p*.

Between the two extremes lies the murky territory of devices shared among several people. For these devices, it might take quite a bit of experimentation to find a policy that balances security and convenience satisfactorily.

Another range of variation concerns the failure behavior of *Loc*. First, is it a failure if no user answers the outgoing call, if the user cannot authenticate himself, or only if *Loc* has no current location for *p*? Second, does *Loc* attempt to handle a failure itself (perhaps by placing an outgoing call to a different address), or does it send an *unavailable* signal upstream to be treated by another feature?

There is also a range of variation in the user interface of this feature. How is *located_near* modified? How is its current value displayed? Do *Loc* or *Ident* inform the user when they have identified him as owning a particular personal address?

Finally, if a user connected to a *Loc* or *Ident* box through device d uses the box to change his current location from d to e, the box can offer to transfer the ongoing customer call from d to e. For example, d might be a home PC and e a cellphone. If the user is talking through d and needs to leave home, he can transfer his location and the conversation to e, pick up the cellphone, and walk out the door with it. This function is discussed further in Section V-B.

[2]It also makes perfect sense to collect location information automatically. However, the design decisions entailed therein are so different that it is a different feature entirely.

B. Switching and Spontaneous Conferencing

B.1 Requirements

A person can only deal with one voice channel at a time. Even passive listening to two voice channels simultaneously is unlikely to be comfortable or effective.

The concept of a single voice channel leads directly to two requirements:

1. If a user has several customer calls in progress, it is necessary to switch the user's single voice channel among the various calls [switching].
2. A user with several customer calls in progress should be able to conference together the voice channels of some or all of those calls [conferencing].[3]

These two requirements are intimately related and best addressed together. The reason is *completeness*, which means in this case that at any time, a user should be able to group his calls into conferences in any way that he chooses, and to be speaking with whichever conference (counting an unconferenced call as a singleton conference) that he chooses.

If switching and conferencing are separated, then completeness becomes extremely difficult to achieve. To see why, consider Figure 4, which shows a configuration achievable through traditional PSTN features. Subscriber d has used 3-Way Calling to make a conference with e and f, and has used another instance of 3-Way Calling to make a conference with g and h. The subscriber's Call Waiting feature enables him to switch between talking to these two conferences. However, there is no way that he can form an *f/g* conference. The historical grouping of customer calls into conferences is embedded in the configuration, and cannot be altered without tearing calls down and setting them up again.

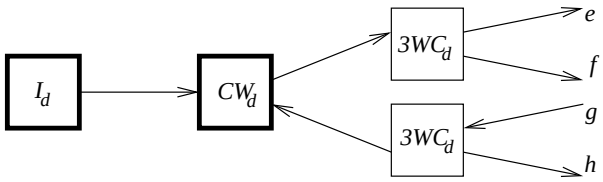

Fig. 4. A completeness problem.

B.2 Core Functions and Software Architecture

This feature is implemented by the bound box type *SSC*, as shown in Figure 5. The instance of *SSC* has one internal

[3]This function is referred to as *spontaneous conferencing* because it operates locally on existing customer calls. Pre-arranged conferences are much more elaborate, requiring configuration functions, participation management, floor control, and security [10].

call on its left, connecting *SSC* to its associated device. This internal call has one voice channel, which is managed by *SSC*.

On its right *SSC* has many internal calls, each corresponding, from the perspective of device *d*, to a customer call. The design of *SSC* gives each of the internal calls on its right a special, formal significance here designated a *switched call*, because it is a call relative to, and preserved by, *SSC*.

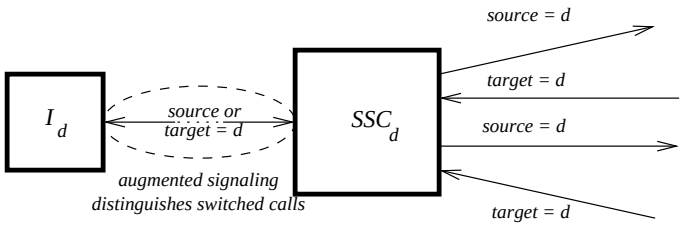

Fig. 5. Architecture of the Switching and Spontaneous Conferencing feature.

Except for its switching and conferencing functions, the behavior of *SSC* with respect to switched calls is intended to be transparent. However, in this case the effect of transparency takes some work to achieve:

1. Each switched call has a locally unique identifier used by *SSC* and *I*. If the switched call is an outgoing call from the device, *I* assigns its identifier. If the switched call is an incoming call to the device, *SSC* assigns its identifier.
2. Except for signals specifically related to switching and conferencing functions, all signals of all switched calls pass transparently through *SSC*. Between *I* and *SSC* signals are labeled with their switched-call identifiers, so both boxes can distinguish which switched call a signal belongs to.
3. From the perspective of *I* and *SSC*, there is a big difference between a switched call that appears when the boxes are idle, and a switched call that appears when the boxes are busy; the former results in setting up a chain of boxes and internal calls between *I* and *SSC*, while the latter piggybacks signaling and media on the existing chain. Nevertheless, the user interface provided by *I* makes both look the same to the user.

Note that the architecture of this feature requires substantial cooperation from the interface box.

Device address *d* subscribes to *SSC* in both the source and target zones. *SSC* is a bound box type. Therefore each switched call to or from *d* is routed through the unique box SSC_d. In Figure 5, the chain of boxes and calls between I_d and SSC_d is double-arrowed because it might have been placed in either direction. It persists as long as the busy episode of the device persists.

There are many ways of providing voice conferencing and switching functions. A minimal, but complete, scheme requires that the user be able to choose individual switched calls and conferences. A voice conference is a set of switched calls. Each switched call belongs to exactly one conference; when it is created (placed or received) it is put in a new singleton conference. Then only two user operations are needed:

1. *select(f: conf)*, which connects the user's voice channel to conference *f*, disconnecting it from any other voice source/sink.
2. *move(c: call, f: conf)*, which moves call *c* (within *SSC*) from whichever conference it is in to conference *f*.

We have to choose between making *SSC* a device feature box, subscribed to by addresses of participating devices, or a personal feature box, subscribed to by personal addresses. This is not an easy choice.

The main advantage of making it a personal feature box is that it will always be available to the person, from any participating or non-participating device.

If subscribers to the system are expected to use a variety of different devices, then making *SSC* a device feature box is even more advantageous. First, to provide a good user interface, it needs to be designed with the device capabilities in mind. For example, on a voice-enabled PC each switched call can have its own window, and there is no real limit on how many of them the user can handle. On a cellphone, on the other hand, it might be wise to prohibit more than two simultaneous switched calls. This customization can be provided by having different versions of *SSC* subscribed to by different device types. If *SSC* were a personal feature box, the same version would be used from many devices, and it could not be customized in this way.

Second, this architecture preserves the integrity of switched calls by means of close cooperation between the interface box and the *SSC* box. If *SSC* were a personal feature box, then *every* device interface would have to be programmed for this particular kind of cooperation, which seems a lot to ask in an allegedly modular and extensible system. If only participating devices are expected to cooperate, then the difficulty of dealing with *SSC* as a personal feature is less, but so are the advantages, since Switching and Spontaneous Conferencing is available from participating devices in either case.

B.3 Behavioral Variations

The main variation in this feature concerns the user interface. Designers have a great deal of latitude in the user-controlled switching and conferencing operations, and in the details of how a switched call is displayed to the user.

Another behavioral variation concerns what happens to other media, such as text. Conferencing of switched calls can apply to all their media channels simultaneously, or to voice alone. Since a user can easily handle multiple simultaneous text conversations, it would make sense to conference voice only, and (on a PC, at least) to display the text channel of each switched call in its own window. Conferencing of text could also be an optional function, or an optional adjunct to voice conferencing.

It is easily possible to add a *transfer* function to this feature. Transfer applies to a conference; after a transfer, the switched calls are still connected to each other, but are no longer connected to the subscriber's device. The definition of a transfer is trickier if some of the media are not included in the conference.

C. Mail

C.1 Requirements

Voice mail originated with answering machines. *Text mail* or E-mail has been gaining momentum since the early days of the Internet. Although the history of *text chat* goes back to the early days of timesharing operating systems, it has had a recent surge of popularity in the form of instant messaging. In this context it is convenient to refer to telephony as *voice chat*.

Despite their disparate histories, these forms of telecommunication are closely related—in function, if not in implementation. The only difference between voice chat and text chat, and between voice mail and text mail, is the medium employed. The primary difference between chat and mail is that the former is real-time communication, while the latter is buffered communication.

Surely one of the best possible uses of IP telecommunications is to unify these functions. By doing so, we should be able to provide a smoother, richer, and more flexible user experience. The unification should satisfy these requirements:

1. Chat between two subscribers is always possible if both people want it.
2. Mail between two subscribers is always possible if either person wants it.
3. If two subscribers have capabilities for at least one common medium, then there is a way for them to communicate.

The basic service (Section II) is chat, so mail must be added as a feature.

C.2 Core Functions and Software Architecture

The Mail feature has two box types, both free: *Read/Send* and *Receive*. Both device and personal addresses can have mail by subscribing to *Read/Send* in the source zone and *Receive* in the target zone.

Mail is stored in the operational data of this feature, as shown in Figure 6. As with the operational data of Location and Identification, the mailbox is partitioned into address slices, and only a box routed to on behalf of address x can access the mail of address x.

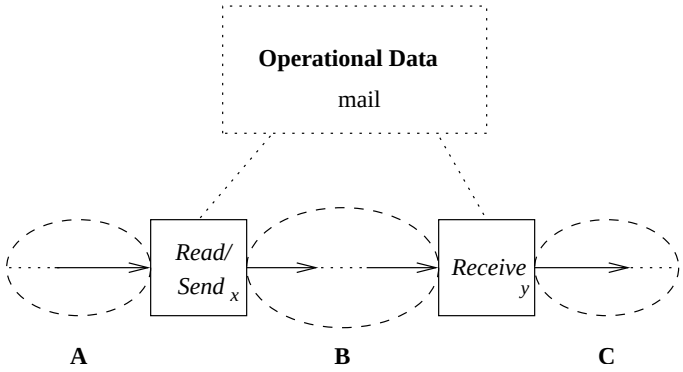

Fig. 6. Architecture of the Mail feature.

The key to understanding Figure 6 is that the three chains of boxes and internal calls marked **A, B,** and **C** may be present simultaneously or at different times. In the default situation (chat), the two feature boxes are transparent and all three chains are present simultaneously.

A *Receive* box must be provisioned (also in the operational data) to accept messages in any medium for which its subscriber has reading capability on some device.

To read mail, a subscriber places a call which will be routed through *Read/Send* in the source zone. The subscriber invokes the *read* function of this box, and opens one or more media channels to this box. The subscriber can then read his messages in all the current media. In this mode of communication, only chain **A** is present.

Alternatively, the subscriber can invoke the *send* function of the *Read/Send* box, and open one or more media channels. In this situation, with only chain **A** present, the subscriber can store in his own mailbox a message to be sent. The subscriber can disconnect **A** as soon as the message is complete.

Once the subscriber has stored a message and ordered it sent, the instance of *Read/Send* that stored the message is responsible for delivering it to the target address. It places an outgoing call to that address with a *mail* flag augmenting the setup signal. If the setup signal travels through the usage to a *Receive* box, the *Receive* box will respond to the flag. Instead of behaving transparently, it will make no outgoing call. It will accept opening of any medium in which it can accept messages, receive the stored message

from *Read/Send*, and store it in the target's mailbox. In this mode of communication only chain **B** is present.

Perhaps the target address does not subscribe to *Receive*. In this case no feature box in the usage will recognize the *mail* flag, and the flag will be ignored. Call chain **B** will be extended with call chain **C**, hopefully all the way to a device, to which the message will be delivered. If the **B** and **C** call chain does not succeed because the device is not available or no one answers it, then the instance of *Read/Send* responsible must retry periodically. Between tries it sleeps, disconnected from all internal calls. The instance cannot die until it has delivered the message for which it is responsible.

If the caller wants to chat, then both feature boxes in Figure 6 are initially transparent. However, an *unavailable* or *unanswered* condition will trigger the *Receive* function, which will accept opening of any media in which it can accept messages, and use any open media channel to offer to store mail for the target address. In this mode of communication chains **A** and **B** are present.

Finally, a caller who originally wished to chat may change his mind and want to send mail instead. He may have chatted with someone already, been disconnected by that person, and wish to send mail as an afterthought. Or the attempt to chat may have failed, he may be currently receiving an offer of mail services from the callee's *Receive* box, and wish to use his own Mail feature instead.

In either case, he can invoke the *send* function of his *Read/Send* box at this later stage. *Read/Send* tears down any parts of **B** that may remain, leaving only **A**, and then acts as it does when *send* is invoked initially.

C.3 Behavioral Variations

During chat, either box type can offer to store (record) the conversation on behalf of its subscriber.

As part of its *unanswered* treatment, *Receive* can offer a *screening* function. Instead of answering an incoming call, the callee invokes this function. The media channel is opened all the way from the caller to the callee. At the same time, *Receive* presents an *unanswered* indication to the caller and offers to store a message. If the caller has a message, *Receive* both stores it and allows it to pass through to the callee. If the callee then requests a full connection, *Receive* stops recording and connects the medium in both directions.

In a multimedia system, more things can happen during a customer call, and old concepts such as *unanswered* must be defined anew. If a caller attempts to open channels for two distinct media, and the callee only accepts one of them, does *Receive* offer to store a message in the rejected medium?

Singh and Schulzrinne suggest some additional variations [14]. A subscriber's instance of a *Receive* box might call him to *notify* him of the arrival of an important message. A call might be *reclaimed* by a user who answers the phone while a message is being recorded (this is a slight variation on screening). It should be possible to record *multimedia messages* with components in several media.

There is a wide range of variation in the user interface for the Mail feature, particularly the user interface for retrieving stored mail. Although DFC includes all the facilities necessary for describing how mail is stored and retrieved, it would also be possible to interface *Read/Send* and *Receive* with an off-the-shelf mail server.

V. Feature Interactions

Since feature interactions are a by-product of feature modularity and feature compositionality, their precise nature depends on the feature-specification language and feature-composition operator. Thus all interactions of our three features are discussed in DFC terms.

DFC was designed specifically to minimize feature interactions that are known to be undesirable, such as logical inconsistencies and implementation conflicts. Composition of features in a less structured framework would result in many more feature interactions than these, most of them bad.

The following discussion is informal, focusing on interactions that are known to occur, and on how they can be managed within DFC. The formal basis for this work, emphasizing detection of potential feature interactions and its relation to verification, is introduced elsewhere [2], [17], [18].

A. Location and Identification

This feature has many interactions. Three of them are straightforward and have already been handled by the feature architecture:

1. Since all outgoing calls placed by *Loc* have device addresses as targets, *Loc* would unconditionally cancel any feature box placed after itself in the target zone of a personal address. This would be bad, and is prevented by placing *Loc* last.
2. *Loc* generates *unavailable* signals upstream, which other personal feature boxes might handle. For example, a *Delegate* box might delegate (forward) the customer call to another person. Or *Receive* might offer to store a message. This beneficial feature interaction is supported by placing *Loc* last in the target zone of a personal address, so signals traveling upstream will pass through them.
3. *Ident* and *Loc* are intended to guard access to personal feature boxes such as *Read/Send* and *Receive*. Their place-

ment in usages puts them between personal feature boxes and users, which is necessary for this desirable guarding interaction to occur.

There is a particularly interesting feature interaction in systems in which some users own personal addresses, while others use device addresses as if they were personal addresses. If *e* is a device address used in this old-fashioned way, it may subscribe to personal feature boxes such as *Read/Send, Receive,* and *Delegate*. What happens when a personal address is temporarily located at this device?

Consider, for example, a personal address *q* subscribing to *Receive* and *Loc* in the target zone. Each incoming call to *q* is retargeted by *Loc* to device *e*, which also subscribes to *Receive* in the target zone. If no one at e answers the call, then the *unanswered* condition will be handled by the nearest *unanswered* treatment, which in this case will be $Receive_e$. This is definitely not the most desirable behavior, as the mail belongs to *q*.

The general solution to this interaction problem is shown in Figure 7. Figure 7 depicts a complex usage involving device addresses d and e, and personal addresses p and q. Device d has placed two switched calls, one to q and one to e. The switched call to q is retargeted to e, as q is currently located at e. The switched call to e is identified as being placed by p, who is currently located at d. This usage contains all boxes of all three features. All four addresses use Mail, and therefore subscribe to *Read/Send* in their source zones and *Receive* in their target zones.

On the source side, the feature boxes subscribed to by device address *d* must be partitioned into device-oriented boxes and personal boxes. The order of boxes in *d*'s source zone must place device-oriented boxes first, followed by *Ident*, followed by personal boxes.

The upper instance of $Ident_d$ in Figure 7 does not identify the user (presumably because the user did not ask to be identified), and does not change the source address it received. It is followed in the usage by $Read/Send_d$. The lower instance of $Ident_d$ identifies the user as p and changes the source address it received. It is followed in the usage by $Read/Send_p$. On either path, there is exactly one instance of the personal box *Read/Send*.

On the target side, the feature boxes subscribed to by device address *e* must also be partitioned into device-oriented boxes and personal boxes. The order of boxes in *e*'s target zone must place personal boxes first, followed by *L&I Marker*, followed by device-oriented boxes. *L&I Marker* is another free box of the Location and Identification feature. It behaves transparently in all cases, and is present specifically to solve this feature-interaction problem.

In the lower path of Figure 7, there is no retargeting, and there are instances of all the boxes subscribed to by e in the target zone. In the upper path Loc_q retargets to e. Loc_q places its outgoing call as a special *direct call* to address *e*. Because *Loc* and *L&I Marker* are both boxes of the same feature, and therefore have the privilege of cooperating in this way, a DFC router routes the call directly to $L\&I\ Marker_e$, bypassing the personal feature boxes subscribed to by e. On either path, there is exactly one instance of the personal box *Receive*.

In Figure 7, most of the arrows are dotted, indicating that other feature boxes might be present there if the appropriate addresses subscribe to them. The exception is the direct call, which is intended specifically to avoid the inclusion of other feature boxes.[4]

Finally, Location and Identification alters addresses, and therefore interacts with many address-sensitive features. For example, the Blocking feature consists of a free target-zone feature box *Blocking* that rejects customer calls from certain callers. The Callback Last feature has a free target-zone box *Log* that logs the source of an incoming call in operational data, and a free source-zone box *Callback* that uses the last source as a target, if requested. Both of these features can be affected by the fact that *Ident* can change the source of a customer call from a device address to a personal address. At least in these cases, the feature interaction is a good one, as a personal address is preferable for both purposes.

B. Switching and Spontaneous Conferencing

The most important feature interactions have already been handled by the feature architecture. Although Switching and Spontaneous Conferencing is a powerful feature, other features should not need to know about it to work properly. The design of *SSC*, in preserving the integrity of switched calls, ensures that feature boxes on its right (in the orientation of Figure 5) will work as expected regardless of the presence of *SSC*.

There is still a question, however, about routing. With respect to device *d* in Figure 5, some switched calls are incoming and some are outgoing. Yet all share the linkage between I_d and SSC_d and whatever feature boxes might be positioned there by routing. So, to preserve completely the integrity of switched calls, any sequence of feature boxes subscribed to by *d* in the target zone after *SSC* must be the reverse of the sequence of feature boxes subscribed to by *d* in the source zone before *SSC*.

[4] If e were a device address that did not subscribe to any personal boxes, then it would not subscribe to *L&I Marker*, either. In this case the direct call would be routed to the first box in the target zone of e.

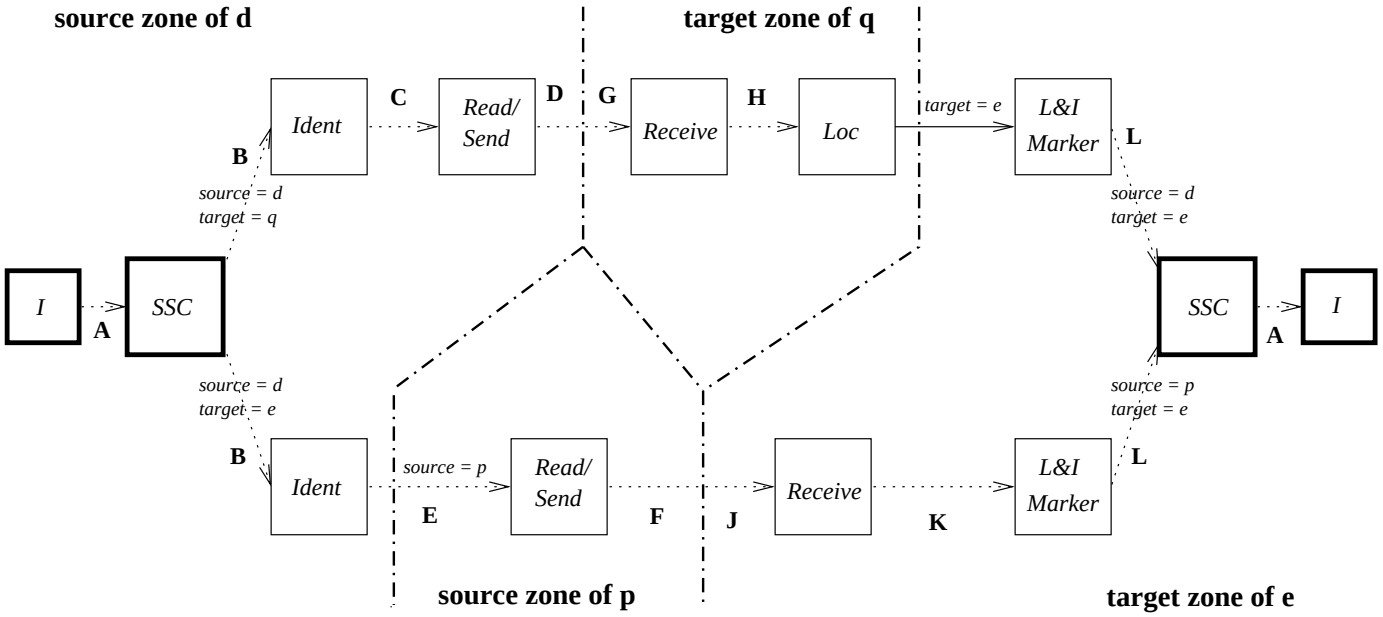

Fig. 7. The three features together.

One feature box that fits perfectly between I_d and SSC_d, and obeys the routing restriction above, is *Break-In*. *Break-In* is part of the Emergency Break-In feature, which allows agencies authorized to handle emergencies to connect immediately to a device at any time, regardless of its state and features.

Figure 8 shows how this feature works. Normally *Break-In* is completely transparent. In an emergency situation, someone makes a customer call to *d* with an *Emergency* box in its source zone. The internal call placed by the *Emergency* box is a direct call, so it goes directly to the *Break-In* box. The *Break-In* box receives it and interrupts whatever else is going on to connect the emergency caller to I_d.

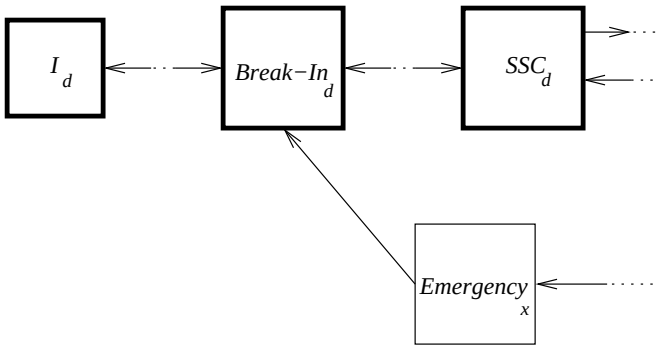

Fig. 8. The Emergency Break-In feature.

Note that the *Break-In* box needs no knowledge of the special signaling arrangement between I_d and SSC_d (see Figure 5). From its perspective, the internal calls on its right and left in Figure 8 are absolutely ordinary. Regardless of how many switched calls they are supporting, these internal calls are interrupted as wholes.

The importance of switched calls is illustrated by adding a transfer function to Location and Identification, as mentioned in Section IV-A.3 and pictured in Figure 9. Originally $Ident_d$ received the call marked **A** (and continued it to the right), and had not yet placed the call marked **B**. Later the owner of personal address *p* used $Ident_d$ to change his location to *e*, and $Ident_d$ transferred the ongoing switched call to I_e by placing **B** and tearing down **A**.

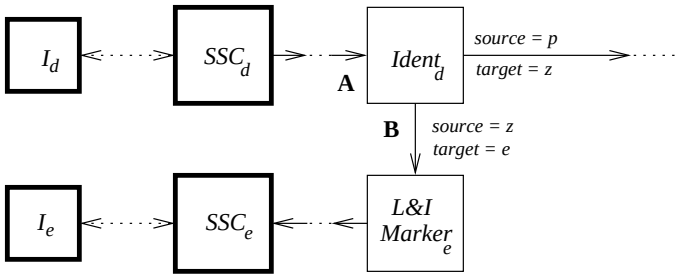

Fig. 9. The transfer function of Location and Identification.

B is a direct call to *L&I Marker*. This creates a switched call at *e* having two targets and no source. However, provided that the switched call is past the setup phase (in which source and target are asymmetric) and into a com-

munication phase (in which source and target are symmetric), the unusual configuration should cause no problems.

The point of Figure 9 is that the transfer is feasible because it is only operating on one switched call. There is no confusing involvement with the other switched calls of d or e, and there are no signaling limitations to prevent *Ident* from performing this new function. *Ident* should ensure the success of the connection to e before tearing down the connection to d, as SSC_e might already be juggling its maximum number of switched calls.

C. Mail

Many of Mail's potential interactions are exemplified by its interactions with Location and Identification, and have already been discussed in Section V-A.

The Personal Directory feature maintains a list of personalized names and their associated addresses. Its target-zone box, *Personal Caller ID*, uses the list to translate the source address of an incoming customer call to a name whenever possible. Its source-zone box, *Personal Dialing*, uses the same list to provide speed dialing.

A reasonable target-zone sequence for an address subscribing to all the personal feature boxes mentioned would be: *Personal Caller ID, Blocking, Receive, Delegate.* If the *Delegate* function is activated, it should supersede *Receive* as a failure treatment. Blocked customer calls should not reach the *Receive* box, as they should not be allowed to leave messages. All of *Blocking*, *Receive*, and *Delegate* can use personalized names rather than addresses, if the names are supplied from upstream by *Personal Caller ID*.

The function of Mail could be augmented incrementally with a source-zone *Mailing List* box that is activated when its incoming internal call has the *mail* flag set and the name of a mailing list in the place of an address. *Mailing List* takes a message from *Read/Send* as if it were a *Receive* box, buffers it, and places one outgoing internal call for each entry in the mailing list, delivering the message to that address.

A reasonable source-zone sequence for an address subscribing to all the personal feature boxes mentioned would be: *Read/Send, Mailing List, Personal Dialing. Mailing List* cannot do its job unless it comes after *Read/Send*. If *Personal Dialing* follows *Mailing List*, then mailing lists can have personalized names in them, which *Personal Dialing* will translate to addresses.

The mailing features also illustrate the limits of switched calls. When an instance of *Mailing List* is active, it receives one internal call corresponding to one switched call, and places many internal calls. Further away from a *SSC* box than a *Mailing List* box, switched calls have become meaningless.

Hall describes eight other E-mail features within in a DFC-like architecture, and analyzes their interactions [8]. Because of the close similarities between the two architectures, these results apply to DFC as well.

D. Examples of Extensions

In Figure 7, bold letters mark regions of the usage where additional feature boxes could be placed. To summarize examples used throughout Section V, here is a list of regions and feature boxes that could be placed there:

A *Break-In*
B *Callback*
D *Mailing List, Personal Dialing*
F *Mailing List, Personal Dialing*
G *Personal Caller ID, Blocking*
H *Delegate*
J *Personal Caller ID, Blocking*
K *Delegate*
L *Log*

Callback Last (*Callback, Log*) is a device feature because its semantics depends on the ordering of switched calls at a particular device.

Regions **B** and **L** might also be the home of media-choice features—features that help users negotiate on which medium they will communicate. For example, a box in region **L**, observing that a caller has attempted to open a channel of a medium that device e does not support, could signal to the caller which media e *does* support. Media-choice features must be closely associated with devices because their functions depend entirely on which media their device supports.

VI. Conclusion

This architecture also handles additional complexity that is not discussed here, for lack of space. There are additional behavioral variations and feature interactions, some of them quite interesting.

Nevertheless, there is enough detail to show the potential of the architecture. It reduces the designer's overall burden to a manageable level by minimizing his need to discover new issues and interactions, by separating concerns, and by solving some specific design problems altogether. Furthermore, the ECLIPSE implementation of DFC shows that real telecommunication systems can be built in this way.

Acknowledgments

My colleagues Greg Bond, Eric Cheung, Michael Jackson, Hal Purdy, and Chris Ramming have all contributed greatly to this work.

REFERENCES

[1] Gregory Bond, Eric Cheung, Andrew Forrest, Michael Jackson, Hal Purdy, Chris Ramming, and Pamela Zave. DFC as the basis for ECLIPSE, an IP communications software platform. In *Proceedings of the IP Telecom Services Workshop 2000.*

[2] Gregory Bond, Franjo Ivancic, Nils Klarlund, and Richard Trefler. ECLIPSE feature logic analysis. In this volume.

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

[4] Kenneth H. Braithwaite and Joanne M. Atlee. Towards automated detection of feature interactions. In [3], pages 36-59.

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

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

[7] P. Dini, R. Boutaba, and L. Logrippo, editors. *Feature Interactions in Telecommunication Networks IV.* IOS Press, Amsterdam, 1997.

[8] Robert J. Hall. Feature interactions in electronic mail. In [5], pages 67-82.

[9] Michael Jackson and Pamela Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering* XXIV(10):831-847, October 1998.

[10] Nadia Kausar and Jon Crowcroft. An architecture of conference control functions. In *Proceedings of Photonics East,* Boston, Massachusetts, September 1999.

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

[12] Henning Schulzrinne. Personal mobility for multimedia services in the Internet. In *Proceedings of the European Workshop on Interactive Distributed Multimedia Systems and Services*, Berlin, Germany, March 1996.

[13] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice-Hall, 1996.

[14] Kundan Singh and Henning Schulzrinne. Unified messaging using SIP and RTSP. In *Proceedings of the IP Telecom Services Workshop 2000*, pages 27-33. Atlanta, Georgia, September 2000.

[15] Hugo Velthuijsen. Issues of non-monotonicity in feature-interaction detection. In [6], pages 31-42.

[16] Pamela Zave. DFC website at `www.research.att.com/info/pamela/dfc.`

[17] Pamela Zave. An experiment in feature engineering. In *Essays by the Members of the IFIP Working Group on Programming Methodology*, Springer-Verlag, to appear.

[18] Pamela Zave. Feature-oriented description, formal methods, and DFC. In *Proceedings of the FIREworks Workshop on Language Constructs for Describing Features*, pages 11-26. Springer-Verlag, London, 2001.

[19] Pamela Zave and Michael Jackson. New feature interactions in mobile and multimedia telecommunication services. In [5], pages 51-66.