An Open Architecture for Next-Generation Telecommunication Services

GREGORY W. BOND, ERIC CHEUNG, K. HAL PURDY, and PAMELA ZAVE
AT&T Laboratories—Research
and
J. CHRISTOPHER RAMMING
SRI International

An open (in the sense of extensible and programmable) architecture for IP telecommunications must be based on a comprehensive strategy for managing feature interaction. We describe our experience with BoxOS, an IP telecommunication platform that implements the DFC technology for feature composition. We present solutions to problems, common to all efforts in IP telecommunications, of feature distribution, interoperability, and media management. We also explain how BoxOS addresses many deficiencies in SIP, including how BoxOS can be used as a SIP application server.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Applications*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.2.11 [**Software Engineering**]: Software Architectures—*Damain-specific architectures*

General Terms: Design, Languages

Additional Key Words and Phrases: Component architectures, electronic mail, feature interaction, instant messaging, Intelligent Network architecture, multimedia systems, network addressing, network interoperation, network protocols, network optimization, service creation, Session Initiation Protocol

1. INTRODUCTION

Telecommunications is networking with an emphasis on real-time, person-to-person communication. Telecommunication services include conversation in media such as voice (telephony), video (videoconferencing), and text ("instant messaging"). Telecommunication services also include mail in media such as

J. Christopher Ramming's work was performed while he was an employee of AT&T.
Authors' addresses: G. W. Bond, E. Cheung, K. Hal Purdy, P. Zave, AT&T Laboratories, 180 Florham Park, NJ 07932; email: {bond,cheung,khp,pamela}@research.att.com; J. C. Ramming, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025; email: jcr@aya.yale.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2004 ACM 1533-5399/04/0200-0083 $5.00

voice (voice mail) and text (electronic mail), both because mail is simply buffered person-to-person communication, and because mail is often used as a backup when an attempt at conversation fails.

Because telecommunications usually involves two or more people, the mode and timing of the communication must somehow be negotiated. Social customs, the dynamics of relationships, the need for privacy, the mobility of modern life, a diversity of telecommunication devices, and the complexity of peoples' schedules all create requirements for the technology. This creates a continual demand for new *features*, which are increments of functionality added to the basic service.

A *feature interaction* is some way in which a feature or features modify or influence another feature in generating the system's overall behavior. Feature interactions are an inevitable by-product of modular, incremental development. They have been a tremendous source of software complexity in the Public Switched Telephone Network (PSTN), and the problem will only get worse in Internet telecommunications (Section 2).

In 1999 we began a project with the assumption that the behavior of an IP-based telecommunication system must be viewed, first and foremost, as the *composition* of a large and malleable set of features. This viewpoint provides an open—in the sense of extensible and programmable—infrastructure. This viewpoint also appears to be the only effective way to build in mechanisms for managing feature interactions.

The project was inspired by Distributed Feature Composition (DFC) [Jackson and Zave 1998, 2001], which is an abstract component architecture for describing telecommunication services. It was designed for generality, feature modularity, structured feature composition, and management of feature interactions, and has been demonstrably successful at meeting all four goals. Section 3 gives an overview of DFC.

DFC feature components are called *boxes.* Our Building Box project in AT&T Research has developed an IP implementation of DFC. This includes BoxOS, a multimedia telecommunication network and infrastructure for running feature boxes, and Boxware, a growing suite of tools for creating and validating feature boxes. Section 4 presents an overview of BoxOS, emphasizing how the structure of DFC is reflected directly in BoxOS.[1]

Since every design decision has followed from the initial assumption of compositionality, we have learned a great deal about its implications for IP telecommunications. In particular, we have had to grapple with three major issues. Any IP telecommunication system will have to confront these issues, so our compositional solutions to common problems may be relevant to many efforts in addition to ours.

First, there is the issue of how and where to distribute features (Section 5). This issue interacts with the need for flexible control over which features are applied in which situations.

Second, there is the well-known and critical issue of interoperability, as discussed in Section 6.

[1]BoxOS was known as "ECLIPSE" in some earlier articles.

The third issue is how to manage media transmission and processing (Section 7). This includes a number of problems such as how to separate signaling and media, how to implement feature composition, how to optimize bandwidth, and how to interact with VoiceXML servers.

Other implementation experience is presented in Section 8. Section 9 describes the now-intimate relationship between BoxOS and the Session Initiation Protocol (SIP). Although SIP is being proposed for service creation in IP telecommunications, its approach is radically different from that of DFC. Nevertheless, the prominence of SIP has influenced us, not only to interoperate well with SIP, but also to package BoxOS as a SIP application server, so that the advantages of BoxOS for service creation can be exploited directly in any SIP context. Also, SIP has had a major impact on the industry's development of media-processing technology and resources, which inevitably affects what is available for use by BoxOS.

Section 10 gives comparisons with other related work.

2. THE FEATURE-INTERACTION PROBLEM

Large PSTN switches have hundreds or even thousands of features. The most familiar ones include Speed Dialing, Call Waiting, Three-Way Calling, Caller Identification, and Call Forwarding (in many versions). Already a significant number of features are used with Internet electronic mail, including features for encryption, filtering, automatic response, and forwarding [Hall 2000].

A feature interaction is simply the effect of one or more features on another's behavior. It can be good (desirable) or bad (undesirable), depending on which behavior is intended or desired. Here are a few examples of typical feature interactions and their causes.

Exceptions. Many features create particular exceptions to the general behavior of other features. For example, a user might subscribe to a feature that routes his calls to one of the places he frequents, based on time of day and a personal schedule. He might then need a feature that allows him to override the previous feature, on those occasions when he is not following his usual schedule.

Triggering. Many features trigger (and suppress) each other. For example, a user might have a Voice Mail feature triggered by a busy condition. A Parallel Ringing feature, which calls several telephones at once in the hope that one will be answered, should interact with Voice Mail. It must suppress Voice Mail when only one of the telephones is busy, because one of the other telephones may still be answered. If no telephone is answered, Parallel Ringing may generate a no-answer timeout and trigger Voice Mail. Or Voice Mail may generate a no-answer timeout that triggers its own function and cancels Parallel Ringing.

Signal overloading. In the PSTN most user commands must be encoded using the twelve DTMF tones. If two simultaneously active features respond to overlapping strings in this alphabet, there is no way for the user to signal his intentions unambiguously. For example, a person may be using a Credit Card Calling feature to access his voice mail. The DTMF tone "#" is both a command to many Voice Mail features, and a command to many Credit Card

Calling features (instructing the feature to end the current call and place another one).

Protocol overloading. All telecommunication protocols have been designed, basically, for two-party communication initiated by one of the parties. Features that introduce additional parties, or initiate communication in unusual ways, will overload these protocols.[2] For example, the standard response heard by a caller in the PSTN is one of three tones, indicating that there is a dialing error, that the callee telephone is busy, or that the callee telephone is being alerted. The alerting tone is heard by the caller until the callee answers or the caller hangs up. Yet even as simple a feature as Call Forwarding on No Answer introduces a third (forwarded-to) party. If there is no answer at the callee telephone, if the feature is triggered, and if the forwarded-to telephone is busy, then the alerting tone will be superseded by a busy tone. What appears to the caller as a single attempt to reach a callee has *two* outcomes rather than one. This can affect many other features, often adversely.

Role confusion. If x is forwarding to y and y is forwarding to z, should a call addressed to x be forwarded to z? In other words, should the forwarding of x and the forwarding of y interact? The question is impossible to answer definitively because we do not know what role address y is playing for x. If y is playing the role of a location, such as a guest office, where the owner of address x expects to be found, then the call should not be forwarded to z. If y is playing the role of a delegate who can represent the owner of x when he is not available, then the call should be forwarded to z, because any arrangements made on behalf of y are applicable.

Conflicting goals. The parties involved in a communication attempt may have conflicting goals. If each party uses features to pursue his own goals, the features will interact dramatically. For example, Caller Identification displays the source address of a call while the callee is being alerted. Since some callers prefer to remain anonymous, Caller Identification Blocking cancels Caller Identification for some source addresses. Continuing the "arms race," a subscriber to Caller Identification may respond to the existence of Caller Identification Blocking by subscribing to Anonymous Call Blocking, which prevents him from receiving calls whose identification is blocked.

Feature interactions cause numerous difficulties. It is often amazingly hard to understand the various ways in which features *could* interact, and to decide how they *should* interact. Even when a feature's interactions are well-understood before it is implemented, the interactions are a significant source of software complexity, development expense, and ongoing maintenance work. When a feature's interactions are not well-understood before it is implemented, the interactions are also a significant source of software faults.

[2]This raises the question of whether a completely different, intrinsically multi-party protocol might be better. The question is open; at best, the user interface of a new protocol would be unfamiliar to people.

Ever since the advent of software control made features possible, the PSTN has been plagued by the feature-interaction problem. Feature interactions have increased software complexity combinatorially. They have prevented PSTN switch vendors from offering any significant degree of customer programmabililty, despite insistent demands and many years of effort. The problem is exacerbated by boundaries between administrative domains [Cameron et al. 1993], despite extensive cooperation and standardization among service providers. This situation has motivated a great deal of research on feature interaction [Bouma and Velthuijsen 1994; Calder and Magill 2000; Cheng and Ohta 1995; Dini et al. 1997; Kimbler and Bouma 1998], but its results appear to be too little, too late for the PSTN.

Some people believe that there is no significant feature-interaction problem in IP telecommunications. For one reason, IP devices and signaling are richer than PSTN devices and signaling, so that feature interactions caused by signal overloading are alleviated. On the other hand, except for signal overloading, not one of the known causes of feature interaction is missing in IP telecommunications. On the contrary, the broader range of capabilities and requirements will make feature interaction much worse.

Another reason is that people focus on familiar consumer features. If everyone has the same ten features in his telephone, how hard can they be to get right? This viewpoint misses the amazing complexity of personal mobility, in which a person uses numerous devices, in different places, each with different capabilities. It ignores business-oriented features for call centers, voice-enabled commercial services, and so on. And it ignores the widespread desire for multimedia services, converged services, location-sensitive services, and integration with Web services.

Since it is not too late to manage feature interactions well in IP telecommunications, we are looking for a way to achieve that goal.

3. AN OVERVIEW OF DFC

3.1 Signaling and Routing

In DFC a request for telecommunication service is satisfied by a *usage*, which is a dynamically assembled graph of *boxes* and *internal calls*. A *box* is a concurrent process providing either interface functions (an *interface box*) or feature functions (a *feature box*). An *internal call* is a featureless, point-to-point connection with a two-way signaling channel and any number of media channels. In Figures 1 through 3, which illustrate usages, the internal calls are represented as arrows between boxes.

The fundamental idea of DFC is pipe-and-filter modularity [Shaw and Garlan 1996]. Each feature box behaves transparently when its functions are not needed. Each feature box has the autonomy to carry out its functions when they are needed; it can place, receive, or tear down internal calls; generate, absorb, or propagate signals traveling on the signaling channels of the internal calls; and process or transmit media streams traveling on the media channels of the internal calls. A feature box interacts with other feature boxes mainly

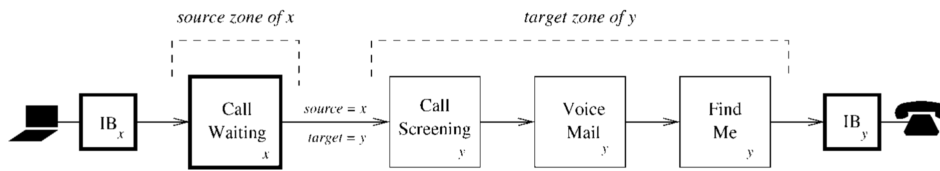

Fig. 1. A DFC usage that has grown from its source interface box (IB) to its target interface box (IB), incorporating all the feature boxes subscribed to by its source address in a source role, followed by all the feature boxes subscribed to by its target address in a target role.

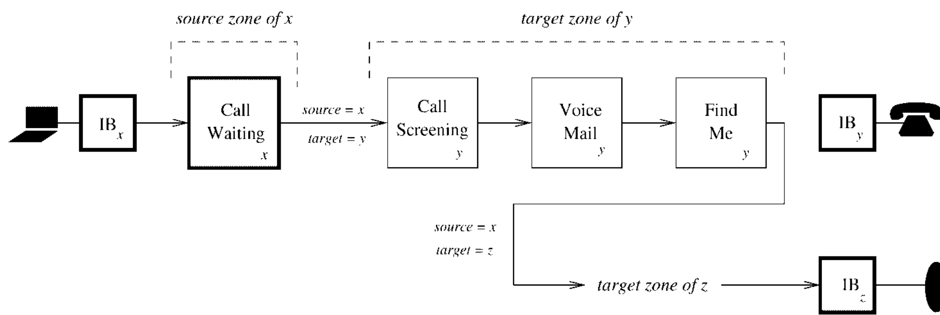

Fig. 2. Because the attempt to reach the original target failed, Find Me has reconfigured the usage with a new target and additional target-zone feature boxes.

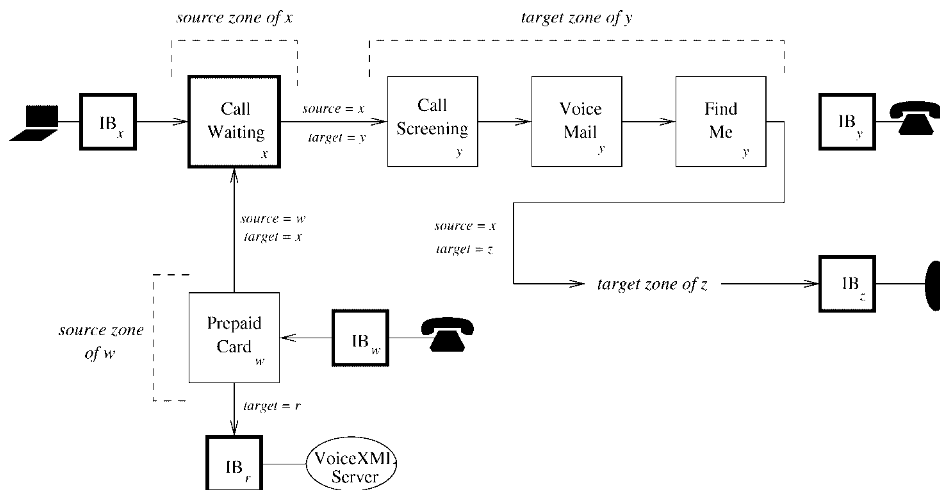

Fig. 3. Through the functions of Call Waiting, a new usage initiated from a public telephone has joined the previous usage.

through its internal calls, yet does not know what is at the far ends of its internal calls. Thus each feature box is largely context-independent; feature boxes can easily be added, deleted, and changed.

Figure 1 shows a usage formed because the device with address *x* is requesting communication. Each internal call is set up with the help of a *DFC router*, which decides which box to route it to. Each DFC router implements an algorithm that does not rely on stored information about the state of any usage,

but rather relies on static data such as which addresses subscribe to which features. All the necessary usage state is carried along in the setup signals of the internal calls.

In the default situation, illustrated by Figure 1, a usage with source address *x* and target address *y* will grow from the interface box of *x* to the interface box of *y*. The action of the DFC routing algorithm will ensure that it includes all the features subscribed to by *x* in the source zone, followed by all the features subscribed to by *y* in the target zone.

When the snapshot in Figure 1 was taken, the Find Me feature box was behaving transparently. An internal no-answer timeout subsequently triggered its functions. It tore down its internal call to *y*, and placed a new internal call to *z*. The change of target from *y* to *z* modified the routing, causing the DFC router to route to the target-zone features of *z*, as shown in Figure 2.

The order of feature boxes within a zone is constrained by a *precedence* order on them. If *y* had subscribed in the target zone to another feature box with later precedence than Find Me, then it would have appeared in Figure 1 between Find Me and the interface box. It would not appear in Figure 2 because the original instance of it would have disappeared when the usage was torn down between Find Me and the interface box.

Precedence is often used to coordinate alternative treatments of a condition such as failure to reach a desired target. For example, in the usage of Figure 2, Find Me was activated because *y* could not be reached. If the attempt to reach *z* also fails, Find Me might not have any additional mechanisms for treating the failure. In this case it would simply propagate the failure signal upstream. The signal would next reach the Voice Mail feature, which would be activated to take a voice message. Thus the fact that Find Me is closer to the failure than Voice Mail gives it priority over Voice Mail in treating failure.

Precedence has other effects on feature composition. For example, the Call Screening feature rejects incoming calls from certain sources that the owner of address *y* does not like. When it rejects an incoming call, it generates a failure signal and sends it upstream. If Call Screening were downstream of Voice Mail, then the generated failure signal would activate Voice Mail, and the undesired caller would be able to leave a message. As it is, Voice Mail never receives the generated failure signal.

In Figure 3, the usage has continued to grow. The device with address *w* is a public telephone designed for use with prepaid cards. So *w* subscribes to a Prepaid Card feature in its source zone. The Prepaid Card feature must engage in an interactive voice-response dialogue with the caller, to collect the authentication code, inform the caller of how much time he has left, and even disconnect the communication when credit is exhausted. To do this, the feature box calls a VoiceXML server with an appropriate script for the dialogue.

Most feature boxes are transient, anonymous instances of their type. They are created by a DFC router when needed. A few feature boxes, however, are persistent and identifiable. They are drawn in the figures with heavier lines, like interface boxes, which are also persistent and identifiable. When a router routes a call to a persistent feature box, it may be joining two existing usages into one.

This happened as the usage developed to the point shown in Figure 3. To understand Figure 3, it is necessary to know that Call Waiting boxes are persistent, and that address x subscribes to Call Waiting *in both the source and target zones*.

Since the caller at w actually dialed x, its branch of the usage grew through the source zone of w to the target zone of x. The target zone of x consists only of the Call Waiting box of x. Since it is persistent, the router routed to that specific one. Since that specific one was already in use, the effect was to join two usages.

All of the addresses in the three figures map to interface boxes and telecommunication devices. DFC also has *mobile addresses*, which do not have any permanent associations with endpoints. Mobile addresses subscribe to features, and these features usually translate mobile addresses to device addresses based on current information and conditions.

Although it is not shown in this example, DFC also provides a *network zone* for feature boxes required by the network rather than subscribed to by individual addresses. Boxes of the network zone are routed to between the source and target zones. Billing is a typical network feature box.

3.2 Operational Data

Some feature boxes require persistent global data. For example, each instance of Find Me must know the alternative target address for its subscriber's address. This is provided in DFC by *operational data*. Operational data is usually partitioned by features, so that only the boxes of a feature can access the partition of that feature. This prevents the accidental use of operational data as a covert channel for feature interaction.

In DFC a feature can actually consist of several different feature box types and a declaration of operational data. The operational data can be a valuable channel of communication between boxes of the same feature. For example, the Voice Mail feature box in the example records messages and stores them in operational data. The feature includes a different type of feature box that enables a subscriber to retrieve his messages from operational data and listen to them (see Section 6.1).

Operational data is accessed in two ways: it is manipulated by feature boxes, and it is updated through a provisioning interface that is completely separate from call processing. Currently DFC constrains feature-box access based only on the feature to which the box belongs and the subscriber address for which the box was invoked. DFC does not constrain the provisioning interface at all.

This simple access control is clearly inadequate. For example, many security-related feature boxes rely on operational data such as passwords, and the features only provide security if the operational data is secure. Thus access control in the provisioning interface is not just a matter of general hygiene—it is a vital part of feature functionality.

At the same time, features must be able to use external databases such as corporate directories. So the approach to data organization in DFC, as it evolves, must be extremely flexible at the same time that it preserves feature modularity and supports analysis of feature interactions.

3.3 Media Channels and Media Processing

The DFC protocol has two distinct levels so that, along a path established by DFC routing and call-level signaling, two-way media channels can be opened at any time, from any direction. This handles many multimedia feature interactions. For example, consider the situation in which *t* places a voice call to *u*, *u* is unavailable, and the call is forwarded by the features of *u* to *v*. Now *t* adds a video channel. Outside DFC, there might be a bad feature interaction in which the video request is directed to *u* and simply rejected [Tsang et al. 1997]. Inside DFC this does not happen, because the video request follows the signaling path established by the usage, all the way to *v*.

For each one-way media stream coming into a feature box from a media channel of a call, the feature-box program controls which media channel (if any) it goes out on. This gives the program the ability to connect two media channels bidirectionally, to switch channels, to mute a channel in one direction, and so on. A stream can be replicated by directing it to two destinations simultaneously. Two streams can be mixed by directing them both to the same destination. In the example, the Prepaid Card feature box will be switching the voice channel to *w* back and forth between the server and *x*. Call Waiting will enable the user at *x* to switch his voice channel back and forth between *z* and *w*.

All other media processing, such as recording, playback, and monitoring, is performed by resources that stand behind interface boxes, for example the VoiceXML server in Figure 3. Feature boxes communicate with these resources through internal calls; they control the resources through the signaling channels of these calls.

No one seems to have much experience yet with multimedia telecommunications and the features that users are going to want. As a result, it is difficult to have confidence in any approach to specifying multimedia service. On the other hand, the DFC media protocol seems to impose minimal constraints. For example, it does not insist that all the media channels in a communication path be opened or closed at the same time, or even in the same direction.

3.4 Management of Feature Interactions

DFC has two major architectural mechanisms for feature composition. These mechanisms are designed to minimize bad or unnecessary feature interactions, and to enable good ones. They are controlled by designers to get exactly the intended feature interactions.

The first major mechanism is pipe-and-filter composition, with feature boxes as the filters and internal calls as the pipes. This mechanism prevents many unnecessary feature interactions. For example, each feature box has its own independent state space, so there is no possibility that a feature's local data will be unexpectedly overwritten by another feature.

Another unnecessary feature interaction is *protocol overloading*, as in Section 2, where a sequence of two outcomes breaks other features that do not expect it. The DFC protocol is designed to accommodate sequential outcomes,

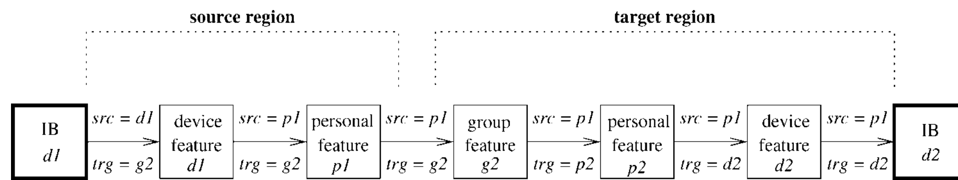

Fig. 4. DFC routing is affected by address translation.

reversed outcomes, shared signaling channels,[3] and many other artifacts of feature complexity. Any correct feature box must also accommodate them, because they are allowed by the basic protocol.

Pipe-and-filter composition is controlled by the ordering of feature boxes to produce the correct feature interactions. For example, consider the two features used to exemplify *exceptions* in Section 2. By placing the override feature box before the default feature box in the target zone, we ensure that the override box routes a call whenever it is active. When the override box is not active it behaves transparently, and the default box routes the call.

For a more complex example of pipe-and-filter composition, consider the two features used to exemplify *triggering*. A Voice Mail box precedes a Parallel Ringing box in the target zone. Since failure outcome signals travel from downstream to upstream, Parallel Ringing is in the right position to intercept failure signals when necessary, thus suppressing Voice Mail temporarily. If Parallel Ringing has a no-answer timeout, it can generate a failure signal that travels upstream and triggers Voice Mail. If Voice Mail has a no-answer timeout, it can destroy the usage downstream of itself, including Parallel Ringing, and offer recording to the caller. If both boxes have no-answer timeouts, then whichever box fires first takes its action as just described, which among other effects disarms the other timer.

Needless to say, it can be difficult for a designer to anticipate and plan for all such interactions in the pipeline. One of our ongoing research efforts concerns algorithms for analyzing feature sets to detect potential interactions so that designers can evaluate them. A preliminary version is described in [Zave 2003].

The second major mechanism for feature composition is the DFC routing algorithm, which invokes features by assembling feature boxes into usages. It is a mechanism for feature interaction because features can alter routing by their behavior, as illustrated by Figure 4.

In Figure 4, addresses *d1* and *d2* represent devices, addresses *p1* and *p2* represent people, and address *g2* represents a group such as a sales group. The device *d1* is used to place a call to *g2*, and the DFC routing algorithm routes the resulting internal call to a source feature box of *d1*.

This feature box places a continuing internal call, but changes the source address to *p1*, indicating that *p1* is the true source of the call. This is better caller identification, and it also causes the router to route this internal call to

[3]A *reversed outcome* occurs when a called party receives outcome signals (see Section 9.1.5). The signaling channel of an internal call can be *shared* by connections to several parties that are being conferenced (Three-Way Calling) or switched (Call Waiting).

the first source feature box of *p1*, whether or not *d1* subscribes to additional source feature boxes. Now the caller can use his personal features and data as part of this usage.

Because *p1* subscribes to only one feature box, and because this feature box does not change the source address as it continues the chain, the next internal call is routed to a target feature box of *g2*. This feature box changes the target address to *p2*, indicating that *p2* is a currently available representative of the group. This causes the router to route to the first target feature box of *p2*. The last target feature box of *p2* (which is the same as the first in this diagram) locates *p2* at device *d2*.

Besides group and personal addresses, there can be many other types of abstract address. For example, an address can represent a role such as "physician" or "youth group leader," an anonymous alias, or a scheduled meeting. The example illustrates a very important positive feature interaction: when a feature changes the target address (forwards), the target features of the new (forwarded-to) address must be invoked. Otherwise a malicious user has an easy way to circumvent privacy features of the forwarded-to address.

Clarifying what addresses identify, as we have done in Figure 4, also solves the *role confusion* problem in Section 2. Now we have personal addresses *px, py,* and *pz*. There are also device addresses *dx, dy,* and *dz*, representing the telephones in the corresponding peoples' offices. Address *py* is currently being forwarded to *pz*. If *px* is using the office of *py*, he forwards his calls to *dy*, and the forwarding of *py* does not apply. If *px* has named *py* as his delegate, he forwards his calls to *py*, and the forwarding of *py* does apply.

With an architecture to structure the problem, it is possible to analyze comprehensively all the feature interactions caused by address translation [Zave 2002]. This work relates address-translation functions to user requirements, deriving principles that balance conflicting goals and draw the line between good and bad interactions.

As an example of its efficacy, Hall [2000] has documented 26 undesirable feature interactions in electronic mail. Twelve of these have nothing to do with address translation. Of the remaining 14 feature interactions, all are predicted by the results in Zave [2002], and all would be eliminated if features adhered to the recommended conventions.

The DFC routing algorithm is a powerful tool for dynamic configuration. Personal needs, telecommunication devices, and network capabilities vary widely, and each situation can demand different software. In Zave et al. [2003] a variety of different configurations for different situations are produced automatically, from the same basic feature boxes, by the DFC routing algorithm.

4. AN OVERVIEW OF BOXOS

In the BoxOS architecture, each DFC system corresponds to an administrative domain. A BoxOS administrative domain is an Internet overlay network, united by some centralized programs and data. There can be, of course, many BoxOS administrative domains implementing many distinct DFC systems.

4.1 BoxOS Components

A BoxOS host is an Internet node that runs a BoxOS router to support the boxes that run there. A BoxOS router performs the functions of a DFC router, and in addition, provides the following box services:

- It creates boxes. For each box type there must be a URL pointing to the code for it. When a box of that type is needed, the router copies and initializes the code.
- It implements DFC internal calls. Boxes send and receive all their call signals through router method invocations.
- It provides access to operational data, subject to the access constraints defined in DFC.

The router and all its boxes run in the same process.

In addition to the distributed routers, each BoxOS domain has a few centralized components. Most prominent of these is a data manager. The data manager serves as a front end to the domain's database, which is a commercial relational database with a JDBC interface. This database holds data of three kinds:

- Routing data such as subscriptions and address maps supports the DFC routing function.
- Feature data supports the features in the system. It includes such items as a URL pointing to the code for each box type.
- Operational data is persistent data read and written by feature boxes as they run.

Provisioning is the telecommunication term for entering customer and configuration data into a system.[4] Except for operational data created by feature boxes, all the data in the database is provisioned.

Each BoxOS domain has a provisioning manager, colocated with its data manager, to support provisioning. The provisioning manager interacts with a Web site, developed with iStudio [Skarra et al. 2001], through which subscribers, feature developers, and administrators enter data into the system. It also interacts with the data manager to perform the updates.

In BoxOS, DFC media transmission is implemented quite separately from DFC signaling. For each medium, there is a set of media managers, which may be placed anywhere in the domain. The managers of a particular medium form a distributed subsystem that controls actual transmission according to the instructions of boxes. Designing the media managers was a major task, as discussed in Section 7.

All the system components introduced above are directly related to aspects of DFC that they implement. In addition, BoxOS has a few other centralized components that support the integrity of the system, and extend its functions beyond those described in DFC.

There is a common mechanism for system monitoring of all kinds. Probes can be placed anywhere in system or feature code. All probes send event notices

[4]The term sometimes encompasses configuration and deployment as well.

to the monitoring manager, which distributes them to interested clients. The monitor clients found in BoxOS today do runtime visualization (see Section 4.2), performance evaluation (see Section 8), and fault detection. Some other possible monitor clients would be subsystems for billing, security enforcement, customer care, usage analysis, and fraud detection.

The registration manager keeps track of the availability of components, including hosts, routers, and remote boxes (see Section 5). Among other purposes, it is an integral part of fault management.

Failures due to hardware or system software are detected dynamically by BoxOS. There is a hierarchy of failure detectors, each one running where it can observe its subject without being compromised by the failure of its subject [Bond 2000]. Failure detectors report the failure of components such as hosts and routers to the monitoring manager. This information is published to interested monitor clients, such as the registration manager. The registration manager informs interested parties such as the peers of failed components, both when they fail and when they recover.

We have basic fault management support in place including automatic restart of failed processes, and host fail-over. This maximizes system availability, but our system testbed has not been utilized enough to provide overall reliability results. One drawback of our current approach is that it is possible for calls in-progress to be dropped during failure recovery. Our next goal is to support recovering from failures so that calls are maintained, a problem that is complicated by the fact that media connections on a failed host will require rerouting to a standby host.

Most of BoxOS is written in Java. Java has platform-independence, code mobility, and extensive libraries supporting networking and distribution. In particular, all real-time communication among the BoxOS components is implemented using TCP/IP sockets.

Several components have virtual input streams merging inputs from multiple sources. Such a stream is implemented as a queue inside the receiving component. The queue is written to by other components through sockets.

Since April 2001, an experimental Building Box implementation has been in use at the homes and AT&T offices of project participants. The purposes of this internal trial are to provide a test bed for Building Box developers and feature writers, to provide a stable system for demonstrations, and to replace the workplace PBX and PSTN for day-to-day communications. The core BoxOS components run on a Solaris machine, with several other machines providing other functionalities.

Figure 5 presents the architecture of the internal trial system. The Service Logic Execution Environment (SLEE) includes a BoxOS router, a voice manager, and feature boxes.

4.2 Programming Tools

The integrity of BoxOS relies on the conformance of feature boxes to the DFC architecture: they must obey the DFC protocol and access operational data correctly. Protecting a BoxOS domain from bad customer code is relatively easy,

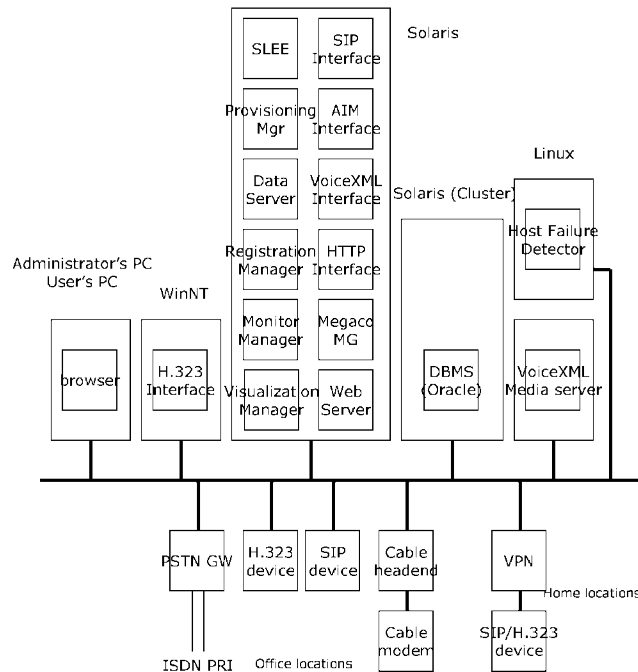

Fig. 5. This BoxOS configuration has been running as an internal trial since April 2001.

because feature boxes run under the control of BoxOS routers, which can check and enforce compliance dynamically.

Even with resource independence, programming feature boxes is a challenge, because a box can have many active ports, and each active box port is participating in an asynchronous concurrent protocol. Our approach to the well-known difficulty of concurrent programming is to provide box programmers with a high-level, domain-specific language in which programming is largely sequential, and in which most aspects of concurrency are handled by the compiler.

Development of this language is ongoing work. Our current language, named ECharts [Bond and Goguen 2002], is based on UML Statecharts, and offers many shorthands well-suited to the problem of box programming. Primary among these is the ability to reuse finite state machine fragments that implement common behavior patterns such as placing and receiving calls, setting up media channels, or simply behaving transparently. There is also an analysis tool to detect protocol errors statically, so that they can be corrected before runtime [Bond et al. 2001].

Our next language will be the even more domain specific Boxtalk, which is based on abstraction and direct manipulation of DFC calls [Zave and Jackson 2002]. It also features a declarative approach to media programming. The Boxtalk compiled code will generate the actual DFC protocol signals, so that they cannot be misused. Once Boxtalk is implemented, further tool development will focus on analysis of feature interactions.

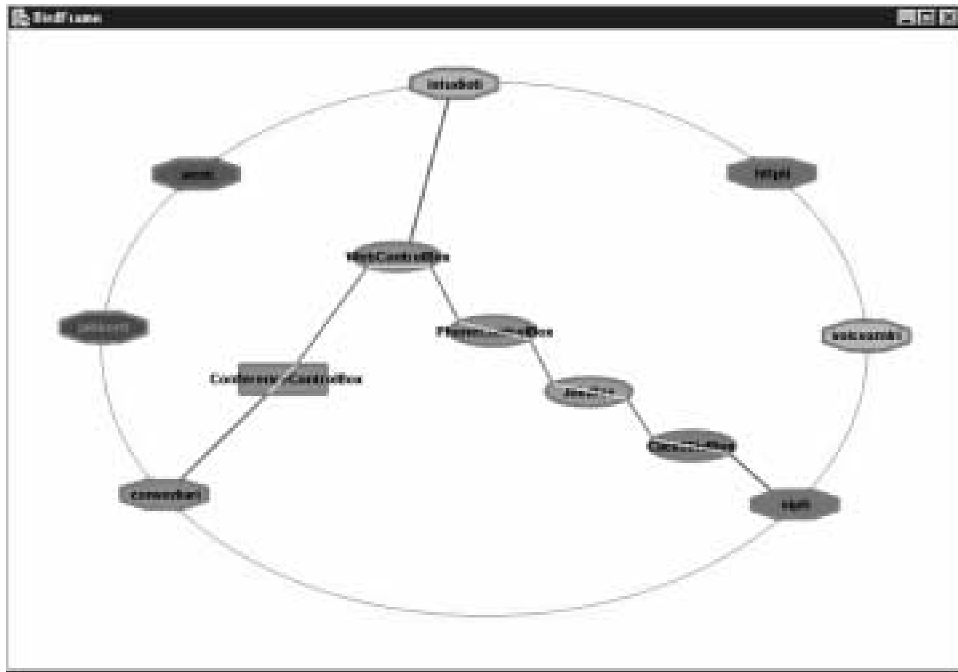

Fig. 6. The dynamic visualization of DFC usages as they execute, distinguishes between interface boxes (polygons), transient feature boxes (ovals), and persistent feature boxes (rectangles).

Currently debugging of feature sets is greatly assisted by the BoxOS visualization subsystem. This is a monitor client that uses incremental graph-layout technology [Gansner et al. 2003; Gansner and North 2000; North and Woodhull 2002] to display an animated picture of DFC usages as they evolve during execution. Figure 6 is a snapshot created by the visualization subsystem. Visualization is also useful for giving demonstrations of BoxOS, because it shows the internal modularity that differentiates BoxOS from other telecommunication architectures.

5. DISTRIBUTION OF FEATURES

A telecommunication system must provide flexible control over which features apply to which customer calls. The uses of this control will reflect the preferences of individuals, the policies relevant to groups such as the employees of a corporation, business arrangements for billing, and applicable laws for safety, tracing, and wiretapping. Section 3.4 explains how the DFC routing algorithm provides a mechanism for exerting this control, based on the feature subscriptions of addresses.

Once a feature box is assembled into a usage, it remains there, in the signaling path, until that part of the usage is torn down by the actions of the boxes in it. This is necessary for two reasons:

- A feature box might need to observe any signal on the path; its functions might be triggered at any time during the life of the usage.

- Even when a box has become permanently transparent, it cannot be removed from the usage. The logic of feature boxes would become hopelessly complicated if they had to be programmed to accommodate such changes in their environments.

If a feature box is a permanent part of a signaling path, its location has many ramifications for network design. Most of the same issues also apply to interface boxes, so they will be considered together.

Putting the interface box and feature boxes of an endpoint in the same node as the endpoint reduces signaling latency and contributes to scalability. It also has the reliability advantage that when the features fail, the endpoint is probably also in a failure mode, and does not need the features.

On the other hand, features and data in the network are available from many devices, even when a particular device is not available. If a node is only sporadically connected to the Internet, its telecommunication devices should be represented by interface boxes located where they are more regularly available. Centralized feature code is easier to maintain and update. It can also take better advantage of large, shared databases such as corporate directories.

It is often important to locate features in a node under the control of a particular authority. Individuals might want the privacy of having their own features on their own nodes. An application service provider (ASP) might sell the use of proprietary text-to-speech and speech-to-text technology that must run only at the ASP site. However, billing, legal wiretapping, and anonymous forwarding require features that are not on nodes controlled by private individuals [Blumenthal and Clark 2001]; such functions are known exceptions to the primacy of end-to-end reasoning in the Internet [Saltzer et al. 1984].

Finally, a few factors are relevant to the distribution of features, without pointing consistently toward the center or the edge of the network. Some code will not run on the hardware or operating system of some nodes. Mobile addresses are not permanently associated with any particular location.

Given the large range of criteria for feature distribution that might be applied by subscribers, programmers, ASPs, ISPs, and other stakeholders, the goal of BoxOS is to place as few constraints on the distribution of features as possible. The location of a feature or interface box is usually determined by the users of BoxOS rather than the design of BoxOS.

Each address known to a BoxOS domain is assigned to a BoxOS host in that domain. The host of an address is the default location of its interface box, if any, and of all feature boxes running on behalf of the address.

When a BoxOS router receives a setup signal to route, it determines from the DFC routing algorithm which address owns the next box in the usage, in the sense that the box is either a feature box subscribed to by the address, or is the interface box of that address. If that responsible address is assigned to this host, then the BoxOS router proceeds to route the setup signal to a local box. If the responsible address is assigned to another host, then the BoxOS router passes the setup signal to the router of the other host.

Network feature boxes are not owned by any particular address. They can be located at a central host, or they can be replicated wherever needed throughout the domain.

In summary, the location of most boxes is based on the address responsible for that box. In the context of the rich addressing structure mentioned above, this should be fine-grained enough for most purposes. Each domain can define its own policies for the location of network feature boxes.

Note that it is relatively easy for any Internet node to acquire a BoxOS router and run as a BoxOS host. So it is often practical for individuals to locate their own feature boxes on their own private nodes.

Unfortunately, platform incompatibilities sometimes prevent running an interface box on a BoxOS host. In this case the interface box can run remotely, that is, on a node without a BoxOS router. A remote interface box must be provisioned as such; the provisioner identifies the BoxOS router to which the interface box and its address are assigned. When the remote box is initialized, it reports its presence to the registration manager, which reports its presence to the selected router. The remote box then runs as if it were local to the router. The use of sockets ensures that the same Java box code will run both locally and remotely.

The principal deficiency of feature distribution in BoxOS is that it does not allow exceptional source or target feature boxes to run outside the host to which the subscribing address is assigned. For example, it is not possible for ASPs to sell to subscribers the use of proprietary feature boxes, yet run them only at the ASP site. In the future we hope to extend the concept of a remote interface box to encompass remote feature boxes as well.

6. INTEROPERABILITY

The ultimate goal of IP telecommunications is to make every type of telecommunication device accessible to every other type. The more devices that can be connected by a telecommunication system, the more useful it is. Interoperability is an important component of openness, because there are already so many devices, protocols, and networks in the world.

From the network perspective, the type of a device is determined by the protocol it understands. Many protocols define subnetworks of devices that are primarily intended to connect only with each other. Such protocols include the PSTN family of protocols, ITU-T's voice-and-video protocol H.323, IETF's voice-and-video protocol SIP, the electronic-mail family including SMTP, and various proprietary protocols for "instant messaging."

There are two major aspects of interoperation, namely protocols and addressing. Protocols affect addressing. Our approach to both is explained in the following sections.

6.1 Protocols

Protocol interworking is a familiar concept. A gateway between two similar protocols translates the messages of each protocol, so that devices with those two protocols can interoperate.

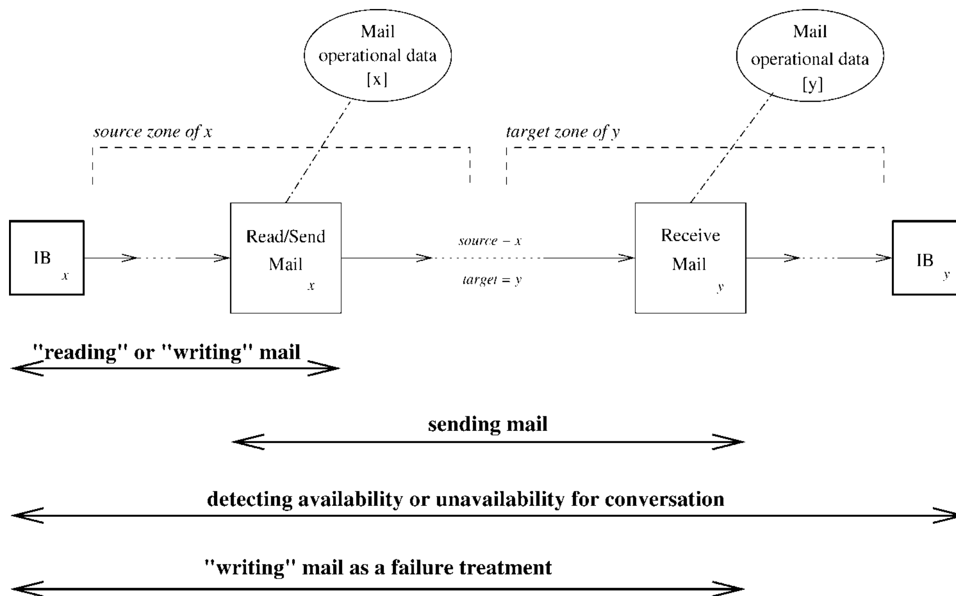

Fig. 7. Both real-time conversation and buffered communication (mail) can be provided by the same protocol. This has been a familiar part of telephony for years, and can be extended to other media besides voice.

Unfortunately, protocol interworking is not enough for true interoperability. It must be possible for devices with dissimilar protocols—intended for different media or different modes of communication—to interoperate.

The DFC protocol accommodates both real-time and buffered modes of communication, as well as all media. So it has the capabilities to act as the universal telecommunication protocol that can subsume all others. If each other protocol is properly interworked with the DFC protocol, then they should all be able to interoperate through it.

As an example of interoperation of modes, text conversation ("instant messaging") and text (or "electronic") mail can interoperate. Interoperation of these modes is already familiar in the voice medium, where it is well-understood that voice mail is a treatment for failure to establish a real-time connection. For this reason, we discuss interoperation of modes in terms of a unified mail feature, where the difference between voice and text is not even visible.

Figure 7 sketches a unified mail feature in DFC. The difference between voice and text is determined by the media choice or choices of the users involved. Hence "reading" mail could be reading text mail or listening to voice mail; "writing" mail could be typing text or recording voice. The devices behind the interface boxes could be electronic-mail user agents or instant-messaging clients as well as telephones or voice-over-IP clients.

To send mail, a user triggers a Read/Send Mail box in his source region to store the mail in operational data. After the sender has disconnected from the usage, the Read/Send box remains to place a call to the recipient address. This extends the usage only as far as a Receive Mail box in the target region. Once

a connection is established between the two mail feature boxes, the mail can be transmitted, after which it is stored in the target's operational data, and the usage is torn down.

To read mail, a user simply triggers a Read/Send Mail box in his source region to give him access to his stored mail. So the bold arrows in Figure 7 show which part of the usage is present at the time that the labeled function is taking place.

The caller may prefer conversation, in which case the usage will be extended all the way to the target interface box, if possible. But if the attempt to reach a callee fails, this can trigger Receive Mail in the target region to offer the caller the chance to deposit mail directly in the callee's mailbox. During the latter function the part of the usage between Receive Mail and the target interface box is usually torn down, as it is of no further use.[5]

In the context of Figure 7, a user agent for electronic mail is a device that cannot receive calls, so its address must subscribe in the target zone to a Receive Mail feature box, which corresponds to a message-transfer agent. The device address also subscribes in the source zone to a Read/Send Mail box (another role of the message-transfer agent) which it can reach by placing an outgoing call.

An instant-messaging client is easier to interwork with the DFC protocol, because it is simply a device for text-only conversation.

An instant-messaging client can communicate with an electronic-mail user agent by attempting to initiate a text conversation. The target-zone Receive Mail box accepts whatever text is sent and records it as mail, to be read later by the electronic-mail user agent. An electronic-mail user agent can communicate with an instant-messaging client by sending text mail to it. Since the instant-messaging client has no Receive Mail box to intercept and record mail, the text goes directly to the instant-messaging client.

Even more surprising than interoperation of modes, the DFC protocol allows devices built for different media to communicate. A subscriber can have an address `boxos:me@home` for communication in all media, even if his actual equipment is a miscellaneous collection of single-medium devices. This address subscribes in the target zone to a Multimedia Pseudodevice feature box, which behaves toward its callers as if it were the interface box of a multimedia device. On receiving an incoming call, it determines the requested medium, and then places an outgoing call to the device that the subscriber uses for that medium. If the caller requests communication in several media at once, the feature box will place several outgoing calls in parallel to connect to all the devices needed. This is shown on the left side of Figure 8.

This arrangement also allows a single-medium device to call a device with a different medium, as shown on the right side of Figure 8. Address x subscribes in the source zone to the Multimedia Pseudodevice feature box. If the device with address x calls a text-only device, its attempt to open a voice channel will fail. The failure is intercepted by the feature box, which can attempt to open

[5]It would not be torn down if the Receive Mail box has a "screening" function. In this case the box is doing two things simultaneously with the mail: recording it, and transmitting it to the interface box so the user can get it immediately and decide if he wants to be available after all.

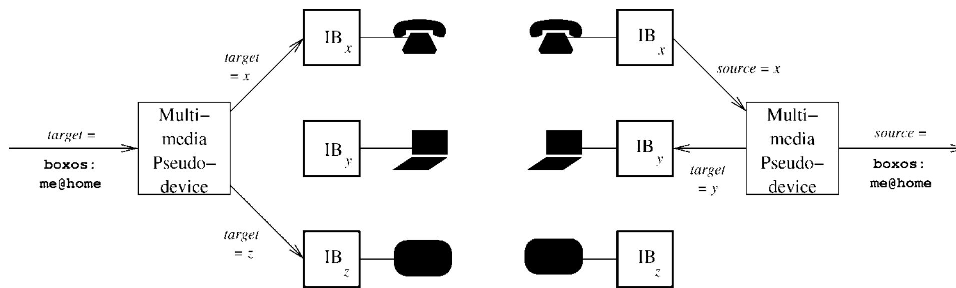

Fig. 8. A multimedia pseudodevice is a facade that makes several devices with different media capabilities function as one. It is easy to construct one using a DFC feature box, and it allows a single-medium device to place a call to any device, regardless of media capabilities.

a text channel instead. If this succeeds, the feature box can place a call to the text device at address y, which the user can answer and use to communicate.

Currently BoxOS includes voice-only interface boxes for SIP and H.323. The SIP interface box is also used for access to the PSTN, since we have a SIP/PSTN gateway. BoxOS also includes text-only interface boxes for AOL and Jabber instant messaging.

As a result of protocol interoperation, to communicate with a device through BoxOS, *it is not necessary to know the type of that device.* Its type will speak for itself through the DFC protocol, as it initiates, accepts, or declines various calls, media, and modes of communication.

Protocol interoperation has important implications for addressing. Because protocol interoperation allows BoxOS communication among a wide range of devices, including devices that are usually considered incompatible, BoxOS does not give any semantics or significance to address prefixes such as `mailto`, `h323`, `sip`, and `http`. Such addresses are interchangeable as far as BoxOS is concerned.

This is a strong contrast to the usual approach, in which these prefixes indicate separate and incompatible services, protocols, or devices. With this usual interpretation of prefixes, there are strong constraints on when and where addresses with certain prefixes can appear. For example, *all* the addresses manipulated by user agents and message-transfer agents for electronic mail must have the prefix `mailto`.

6.2 Addressing

A network maps addresses to objects. A *native* address of an administrative domain is an address for which that domain owns and maintains the mapping. A *foreign* address of an administrative domain is an address whose mapping is controlled by an external domain.

An *address space* is a set of addresses that conform to a well-known syntactic rule. For example, the address space of the PSTN is determined by the E.164 standard. The SIP address space is the set of all Universal Resource Identifiers (URIs) with prefix `sip` and some other minor constraints.

For BoxOS, we needed to design three things:

- An address space.
- A subspace of the address space for native addresses.
- For every address that enters a BoxOS domain through an interface box, a translation of that address to a member of the BoxOS address space.

The goal of interoperability creates the following requirements for an addressing scheme:

- *Universality:* There must be at least one address mapping to every object relevant to telecommunications.
- *Uniqueness:* Every address must map to at most one object.
- *Scalability:* Routing must be feasible even if there is a very large number of addresses in use.
- *Reachability:* It should be possible for any telecommunication device to reach any other telecommunication device through BoxOS.
- *Consistency:* If BoxOS receives an address a through an interface box from another domain D, and if BoxOS translates a to a', then a' in BoxOS must map to the same object as a in D.

BoxOS addresses follow the URI style, so they have the form *prefix* : *user* @ *host*, where *prefix, user,* and *host* can be any strings over a specified alphabet, and where *user* @ is optional. As with URIs, additional fields can be encoded in an address.

Some foreign addresses are already in this form, with prefixes such as `mailto`, `h323`, `sip`, `jabber`, and `http`. These addresses are included in the BoxOS address space. Protocol matching makes this possible, because there is no need to treat these foreign addresses in a special or constrained way.

Other foreign addresses, such as those native to the PSTN, AOL Instant Messenger (AIM), and X10, are not in the BoxOS form. We encode them in the BoxOS address space by adding prefixes such as `pstn`, `aim`, and `x10`. The resulting addresses have the form *prefix* : *host*.

Native addresses have the prefix `boxos` or `sip`. A SIP user agent may be a native BoxOS device, but the protocol between the user agent and its interface box demands that it have a `sip` address rather than a `boxos` address.

It seems clear that this scheme achieves universality. It includes foreign address spaces not yet invented, as new prefixes can always be added.

The scheme also achieves uniqueness even if foreign address spaces overlap. For example, the string `8001234567` is a valid address in both the PSTN and AIM address spaces; it might map to one object in the PSTN and another object in AIM. In our address space it would become two distinct addresses `pstn:8001234567` and `aim:8001234567`. It should be noted that uniqueness is only achieved if foreign addresses are uniquely mapped in their native domains. If `pstn:8001234567` is ambiguous in the PSTN, there is nothing we can do about it.

The scheme is also good for scalability, because the prefixes provide a uniform, highest-level partitioning of the address space that can be exploited for hierarchical routing. Below the level of prefixes, the *host* part of an address can be distinguished and located with the help of the Internet Domain Name System.

Reachability is an inherently difficult problem, even when universality has been achieved. The challenge is to enable foreign devices to reach devices outside their own domains, because foreign domains often only allow their own addresses to be generated and transmitted within the domain. For example, only E.164 addresses can be transmitted through the PSTN.

The most general, and most unattractive, solution to this problem is “double dialing” (also known as “two-stage dialing”). Continuing the PSTN example, a gateway between the PSTN and a BoxOS domain can obtain a few E.164 addresses for access purposes. A PSTN user dials one of these numbers to reach the gateway. Then some feature in the BoxOS domain prompts the user to enter the real address through in-band signaling.

At least some addresses can be reached by the more attractive technique of *double addressing*. Again continuing the PSTN example, a block of PSTN addresses must be obtained for this purpose; each PSTN address is paired with a BoxOS address to be reached from the PSTN.

For example, native address `boxos:user1@host1` must be reached easily from the PSTN. PSTN address `8881234567` is allocated for this purpose, so a PSTN user who wishes to reach `user1` dials `8881234567`. This address is routed in the PSTN to a BoxOS gateway, which encodes it as `pstn:8881234567`. The address `pstn:8881234567` subscribes in the target zone to a feature box that translates it to `boxos:user1@host1`. No PSTN caller has to dial more than once to reach this native BoxOS address.

Finally, for every address that enters a BoxOS domain through an interface box (gateway), there must be a translation of that address to a member of the BoxOS address space. The translation should be consistent, so that the translated address in BoxOS maps to the same object as the untranslated address maps to in the domain from which it came.

Interface boxes usually perform the translation. Each interface box is programmed to suit the domain to which it interfaces. In the easy case, an address coming to BoxOS from domain D is native to domain D. The interface box knows whether native addresses of D are included or encoded in the BoxOS address space, and handles them accordingly.

The hard case occurs when domain D presents BoxOS with an address that is native to yet another domain D'. To translate such an address reliably and consistently, BoxOS must know how addresses native to D' are handled in D!

For one example, let D' be the PSTN, and let D be some domain that includes PSTN addresses in its address space, and excludes AIM addresses from its address space. A string of ten digits coming from D must be a PSTN address, and should be given a `pstn` prefix in BoxOS.

For another example, let D' be the PSTN, and let D be some domain that includes both PSTN and AIM addresses in its address space. Because a ten-digit string is ambiguous in D, and could be either a PSTN address or an AIM address, there is nothing we can do to disambiguate it.

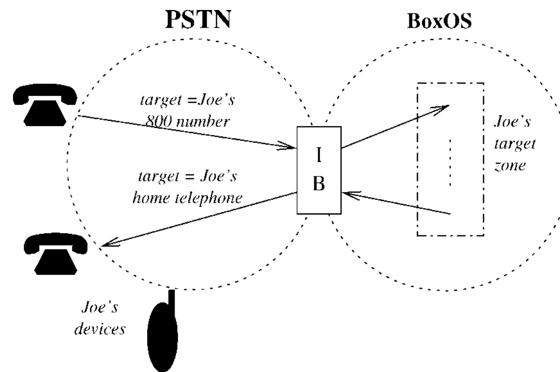

Fig. 9. Because of interoperability, PSTN service can be augmented by features in a BoxOS domain—an easy way to make the PSTN more open.

For a final example, let D' be the PSTN, and let D be a SIP domain. A PSTN address is encoded as a SIP address by giving it a `sip` prefix and a "user parameter" `user=phone`. BoxOS must recognize this encoding, translating such an address to a `pstn` address, rather than including it as a `sip` address.

Beyond reachability of devices, good interoperability with respect to addressing allows a network to integrate its features with features in other networks.

As an example of the opportunities, Figure 9 shows how BoxOS interoperation with the PSTN makes it possible to insert BoxOS features into the PSTN. Joe has a PSTN 800 number that he uses as a personal address. The PSTN is provisioned to direct its calls to a BoxOS interface box (gateway). In BoxOS, the encoded version of this address subscribes in the target zone to personal features—say the same features subscribed to by y in Figure 1. The last feature, Find Me, uses information about Joe's whereabouts to redirect an incoming call to Joe's home PSTN telephone.

7. MEDIA MANAGEMENT

7.1 Signaling-Media Separation

In DFC all media channels follow the paths of the DFC internal calls that control them (Section 3.3). Thus media channels pass through feature boxes, each of which has the capability to connect, mute, switch, replicate, and mix them.

Like all other implementations of voice over IP, the BoxOS implementation of DFC separates signaling from media, so that signaling and media can travel by different paths. We have two motivations for this separation. First is the well-known motivation of conserving bandwidth by sending media on the shortest possible path. Equally important, but less well-understood, is the need for feature composition with respect to media processing.

Just as the externally observable signaling behavior of a telecommunication system is determined by the composition of all its features, so is its media behavior. The exact content of a voice channel at a particular time may be the result of several features, acting concurrently.

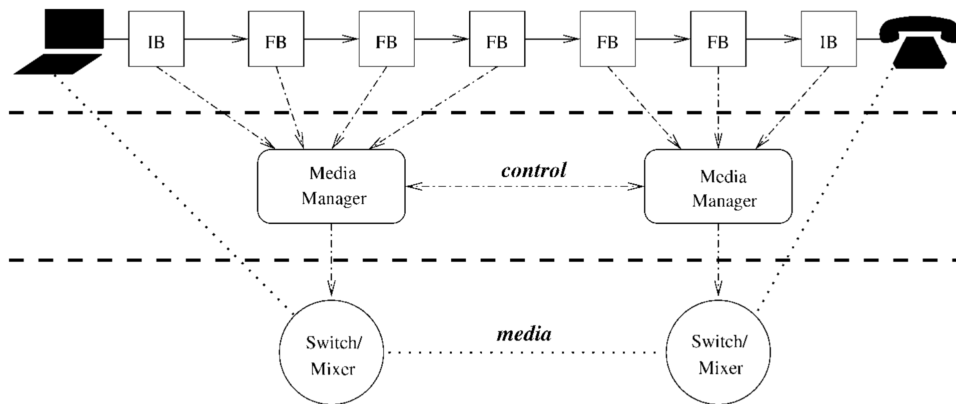

Fig. 10. The BoxOS media implementation has three layers.

Figure 3 gives a simple instance. Depending on the current states of the Call Waiting *and* Prepaid Card features, device *x* might be voice-connected to device *z*, device *w*, or nothing. Device *z* might be voice-connected to device *x* or nothing. Device *w* might be voice-connected to device *x*, the VoiceXML server, or nothing. The pipeline arrangement of feature boxes and internal calls determines how both signaling behaviors and media behaviors of feature boxes compose to determine externally observable behavior. In Figure 3, device *x* will be voice-connected to device *w* if and only if both Prepaid Card and Call Waiting are in the correct state for this behavior.

Section 7.2 explains how we separate the media implementation into layers. Below the top layer containing service logic, each module can implement the media composition of many feature boxes. This allows us to have any number of implementation modules that we choose, and to locate them in any place that we choose. Section 7.3 discusses various optimizations that fit into this implementation scheme.

7.2 Media Layers

For each medium, there is some number of media managers. Each media manager controls a switch/mixer, so the total number of switch/mixers is the same as the number of media managers, as shown in Figure 10.

Each box is assigned to a manager for each medium at the time it is created. As the box executes, it reports its state changes related to a medium (creation and destruction of call channels and intra-box links) to its manager for that medium. The media manager is responsible for implementing these state changes.

To see how media managers work, let us assume that all the feature boxes in Figure 3 except Call Waiting and Prepaid Card are transparent to voice, and that all are assigned to a single voice manager. Each voice manager maintains an internal model based on the reports it has received. Figure 11 shows the state of this manager's model when devices *x* and *z* are talking, and device

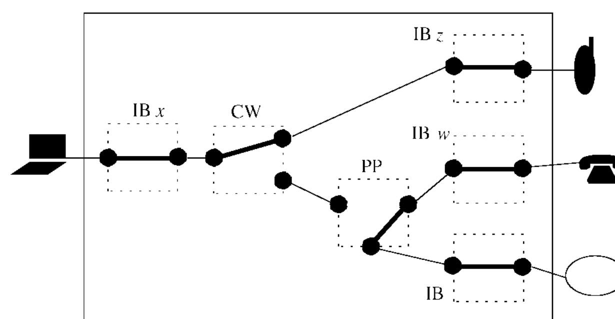

Fig. 11. A model state in a voice manager reflects the voice states of many DFC boxes. Because it has this information, the voice manager can direct its switch/mixer to implement the composition of media control in all features.

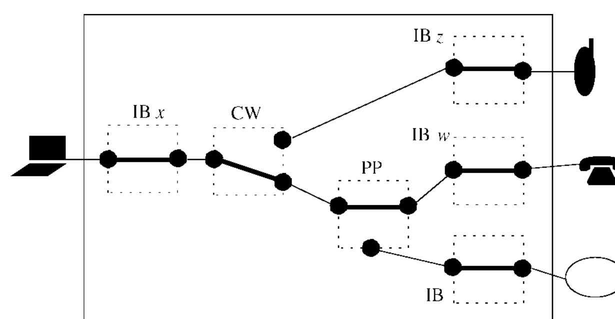

Fig. 12. Because individual features have changed their parts of the model, end-to-end voice connectivity is now different.

w is interacting with the VoiceXML server. Dots represent box ports, heavy lines represent intra-box voice links,[6] and light lines represent voice channels in calls. The dotted box outlines are for clarity only, as boxes do not appear in media models.

Figure 12 shows the voice manager's model when devices *x* and *w* are talking. Note that, even when *x* has *z* on hold as in this figure, *w* will not be voice-connected to *x* when the Prepaid Card feature interrupts the connection to play *w* an announcement that credit has nearly run out. During the interruption, the Prepaid Card box will revert to the media state it had in Figure 11.

If the functions of a feature box are transparent with respect to some medium, then the box need not appear in a model at all. This is why the other feature boxes in Figure 3 are not reflected in Figures 11 and 12.[7]

[6]In reality intra-box media links are one-way, so that media flow can be controlled separately in each direction. In this simplified diagram, all links and channels are two-way.
[7]Recall the assumption that the other feature boxes are transparent to voice. This assumption is absurd for Voice Mail—it is introduced just to keep the diagrams simple.

In any model state whatsoever, the voice output at each external port of the model is a mix of the voice inputs at some set of external ports. The media manager computes the model outputs, and recomputes them whenever the model changes [Cheung et al. 2002]. Each media manager uses the model outputs to control its corresponding switch/mixer. Currently our implementation uses Megaco technology for voice switching and mixing [Cheung et al. 2002].

If there are multiple managers for a medium, then media channels of some calls cross manager boundaries, as shown in Figure 10. Media managers communicate with each other in a straightforward way to implement these channels.

The models in the media managers perform the all-important function of preserving feature modularity—features know nothing about each other, yet their individual behaviors are correctly composed. Separation of the media implementation into layers sets the stage for a variety of optimizations, as will be discussed in the next section.

It is also very important that the control interface between box programs and media managers is based primarily on the definition of DFC, specifically on the capabilities defined in DFC for media manipulation. The definition of DFC is very stable, which means that the control interface is very stable. As a result, we will not have to change or recompile existing box programs when the lower two levels of the media implementation changes.

7.3 Media Optimization

A few optimizations are derived from the models themselves. For example, imagine that the global media state is as shown in Figure 12, and the interface box of z is assigned to a different media manager than the interface box and Call Waiting box of x. Then there is a two-way voice connection between the switch of x and the switch of z, yet any media packets received by the switch of x from the switch of z will be discarded. To save bandwidth, the manager of x can signal to the manager of z not to transmit anything until further notice.

Another possible optimization concerns hairpin removal. Assume that addresses x, y, and z in Figure 1 are all located at different hosts, and that each host has its own voice manager. Since the Voice Mail box of y is not actually voice-transparent, the usage of Figure 1 will be reflected in the models of all three voice managers , and the voice path will pass through all three switch/mixers.

Eventually the usage may reach a state in which all the feature boxes of y are committed to permanent voice transparency (Voice Mail has been disabled for the remaining lifetime of this usage). The control interface allows box programs to “lock” some part of their media processing by telling their media managers that they will make no further changes to that part of the model. From this point on, voice should be transmitted directly between the hosts of x and z, without a hairpin through the host of y. The media manager of y should initiate a distributed algorithm in which it collaborates with the other media managers to remove the hairpin.

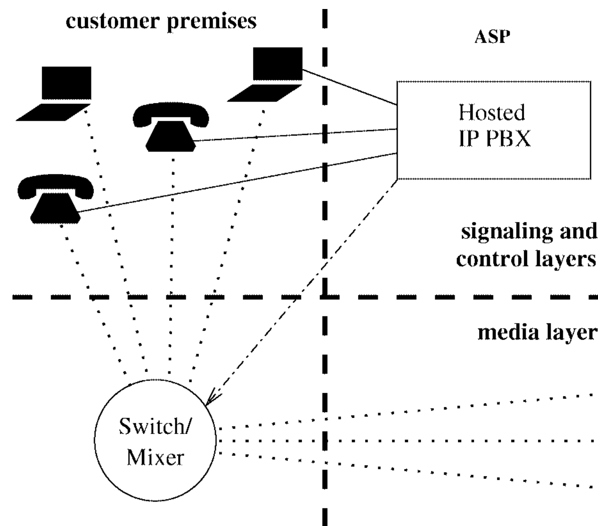

Fig. 13. While a hosted IP PBX resides at an ASP site, its switch/mixer resides on the customer premises. This exploits the separation between the lower two layers of the media implementation to make intra-PBX calls more efficient.

Design of this distributed algorithm seems feasible but tricky. The voice quality must be protected during the transition. The algorithm must work correctly even if two media managers that are adjacent on a media path try to remove themselves from the path concurrently. Hairpin removal will be tackled in future work.

The media paths that connect devices and switch/mixers can be chosen according to considerations of QoS, bandwidth, and latency. Virtually any optimization technique can be built into the media managers for this purpose.

It might seem possible to optimize the locations of switch/mixers as part of the path optimizations, as in Choi et al. [2001]. However, this is probably not the case. A typical media manager will be assigned all the interface and feature boxes of some set of addresses. This means that its switch/mixer will be part of many different media paths simultaneously, which argues for a larger-grained, more static approach to the placement of switch/mixers.

Figure 13 illustrates an interesting case. An enterprise contracts with an application service provider (ASP) for the services of an IP PBX. Because the IP PBX is hosted, it resides with the ASP rather than on the customer premises. The PBX's switch/mixer, on the other hand, is located on the customer premises. This avoids hairpinning intra-PBX calls off the customer premises.

7.4 Interacting with VoiceXML

As shown in Figure 3, when a feature box needs an interactive voice-response dialogue with a user, it places a DFC internal call to a VoiceXML [McGlashan et al. 2002] server. As shown in Figure 11, during the actual dialogue the feature box connects a voice channel to the server with a voice channel to the user. The

feature box uses the signaling channel between itself and the VoiceXML server to tell the server which script to interpret, to receive the results of user input, and to abort the script when necessary.

Currently there is a great deal of controversy concerning whether and how to add call control to VoiceXML. It is motivated by the desire to enjoy the many benefits of VoiceXML for implementing voice services, yet to create services that do more complex call control than VoiceXML allows.

The BoxOS solution is an alternative to extending VoiceXML. We use VoiceXML only for what it is really meant for, which is *sequential* voice-response dialogues. DFC feature boxes handle call control, which is highly *concurrent.* The separation of concerns is intuitive and effective; it keeps both programming models conceptually simple and easy-to-use.

8. OTHER IMPLEMENTATION EXPERIENCE

8.1 Interface and Feature Boxes

Naturally we implemented a variety of interface boxes for telecommunication devices such as PSTN telephones, voice-over-IP clients, and instant-messaging clients. We also implemented a variety of interface boxes for media-processing resources to support audio recording and playback, interactive voice response (IVR) via Voice XML scripts, automated speech recognition (ASR), and text-to-speech (TTS).

Surprisingly, there are many other purposes for interface boxes. We had not anticipated beforehand that we would need all of the following:

- For system testing, we have developed a command-line interface box that uses a `telnet` connection to simulate placing a voice call.
- To support Web-based application interfaces such as click-to-dial, we have an HTTP interface box that accepts HTTP requests from client browsers. The effect of these HTTP requests is to establish and manipulate calls.
- In a more experimental vein, we have developed interface boxes that provide user presence information. One such box is interfaced to a cellphone charging cradle. When a cellphone is placed in the cradle, the interface box is notified. When a call is made to the cellphone's owner, a feature box calls the cradle interface to query its state, and forwards the call to the owner's desk telephone if the cellphone is in its cradle. Otherwise the call is forwarded to the cellphone.

An enhanced feature set creates a need for enhanced user-interface signaling to control the features. In theory this is not a problem in IP telecommunications, because a voice-over-IP client can easily provide any kind of signaling desired. In practice it is a terrible problem. Most of our users have PSTN telephones with very limited user interfaces. Even IP telephones can be difficult to program and riddled with old-fashioned assumptions. Voice-over-IP clients running directly on PCs are seldom used because of poor voice quality and overall performance degradation.

The ultimate solution is a major programming effort to produce PC clients or programmable telephones with the flexibility and extensibility that is really needed. In the short term we rely on a signaling-only pseudodevice created from an instant-messaging client with its interface box, a Web browser with its HTTP interface box, and a feature box to coordinate the two. Output to the user appears in an instant-messaging window. The user makes choices by clicking URLs in the output; these choices enter the system through the HTTP interface. The PC running these clients is usually located on the same desktop as the telephone it is controlling.

Dozens of feature boxes have been implemented and executed on BoxOS. There have been no cases in which we conceived of a feature but could not fit it into our architectural framework.

Generally we have found feature development to be a simple activity. Feature debugging is particularly straightforward because the logic of each feature box is easy to isolate from other feature boxes in the usage, as well as the infrastructure on which the boxes run. As promised by DFC, features tend to interact properly without extra planning, testing, or debugging.

Typically a new feature developer first learns the fundamentals of DFC, then learns about the development environment, and finally studies the design and implementation of a very simple feature box with transparent behavior. Learning about the transparent feature box exposes developers to the ECharts language and a number of common state-machine fragments that the developer will be likely to reuse when developing new features. The training period typically takes one week, after which a developer can complete a simple feature in a few hours.

Voice Ring is a target-zone voice feature that prompts the caller, "Tell me why you're calling." The caller's answer is recorded, and immediately played back by the callee's telephone *to replace normal ringing.* Instead of hearing a meaningless bell, the callee is alerted by the caller's voice, telling him what the call is about.

Recently a colleague implemented this unusual feature on BoxOS in one day. It took half a day to program the feature box and the VoiceXML script. It took the other half-day to program a special interface box for his telephone, enabling the telephone to substitute a different action for normal ringing.

More sophisticated features, which exploit fewer reusable components, have taken up to a week to develop. We are finding that we currently do not have adequate support for dealing with a dynamically changing number of calls and the related issue of handling the dynamic aspects of multimedia channels. For example, we have found that generalizing the basic call waiting feature to accommodate n parties and full multimedia has proven to be a complex design task. Note that it is significantly more difficult to manipulate multimedia calls in the context of a feature than it is to support simple featureless multimedia calls.

In the long run, higher-level, more domain-specific programming abstractions will be absolutely essential for feature development. As described in Section 4.2, language development is a high priority.

8.2 Preliminary Performance Evaluation

Performance has been a vexing problem for our project; this is not an unusual situation for a research prototype being transformed into a production-quality system. Since we undertook the optimization task, we have reduced the call setup time by an order of magnitude, to approximately 0.5 seconds.

Our approach to improving system performance is to profile the code to identify performance bottlenecks and impediments to scalability. Because the majority of the system is written in Java, we have benefited from performance improvements in new versions of Java virtual machines and their just-in-time compilers. Furthermore, there are a number of outstanding bottlenecks, so there is still room for substantial improvement.

To understand the throughput performance of BoxOS, one must be familiar with the measurements used for traditional telephony. A common metric for telephony system throughput is busy-hour call attempts (BHCA). Generally, this metric indicates the maximum number of successful call attempts that can be processed over the course of one hour. However, the definition is frequently distorted for marketing purposes.

For example, to artificially increase this value, measurements may reflect only the availability of a dial-tone. While such a measurement takes into account the resources necessary to attempt a call, a more accurate measurement will take into account the resources necessary to establish, maintain, and tear down a call.

A variation on this measurement might also incorporate the overhead associated with features in a call. however, today there is no accepted notion of a standard feature set that can be used as the basis for comparison with other systems that support features (see below). Finally, when considering system performance, one must also consider the cost of the system software and hardware that achieves that performance.

We have just begun to gather performance data for our system under load. As it is currently deployed, the system supports a sustained rate of 2.5 calls per second, or 9000 BHCA. Here, callers and callees are SIP endpoints that do not subscribe to features, and a call includes the resources necessary to set up, hold, and tear down a call. Assuming that, in a typical office environment, every phone makes one call during the busiest hour of the day, our system can support an office with 9000 phones. This is achieved on an entry-level Sun server with two 900 Mhz CPUs, 2 GB of RAM, and a list price of approximately 11,000 U.S. dollars.

The SIP community is beginning to adopt a set of metrics under the name of SIPstone [Schulzrinne et al. 2002]. The purpose of these metrics is to evaluate SIP proxy, registrar, and redirect servers.

This is relevant to BoxOS because BoxOS can run as a SIP application server (Section 9.3). On the subject of application servers, the SIPstone whitepaper states that until there is better operational understanding of what type of programmable services will be used, it is premature to define metrics for application servers. Indeed, an informal investigation reveals that, today, SIP application server vendors do not disclose performance data on their web sites.

9. RELATIONSHIP TO THE SESSION INITIATION PROTOCOL (SIP)

9.1 Comparison to Service Creation in SIP

SIP is an IETF protocol for creating, modifying, and terminating multimedia sessions [Rosenberg et al. 2002b; Rosenberg and Schulzrinne 2002]. It is emerging as a leading protocol for Internet telecommunication, and is being recommended for service creation [Kristensen; Rosenberg and Shockey 2000] as well as basic infrastructure.

With respect to service creation, there is a very large number of differences between SIP and DFC. Such a large number of differences can only result from fundamental differences in design philosophy. This is indeed the case. The primary design goal for SIP was conformance with Internet principles [Clark 1988]. The primary design goals for DFC were feature modularity and structured feature composition. These umbrellas cover numerous, more specific, differences.

For purposes of this discussion, a *feature module* is a software module offering feature functionality; it is associated with an address, so that communication requests including that address may be routed to that module. A SIP proxy or redirect server is a feature module, as is a DFC zone containing all the feature boxes subscribed to by a particular address.

9.1.1 *The Programmer's Viewpoint.* SIP is intended to run directly on the Internet. Naturally it has responses indicating syntactic or protocol errors, network failures, and server failures.

DFC was always envisioned as running in a protective execution environment that shields the network from programmer errors and the programmer from network errors. The DFC protocol assumes the absence of both types of problem.

This gap in viewpoint is easily bridged. Higher-level APIs for SIP service creation are now being developed [Kristensen]; these facilities protect the network from the programmer by eliminating syntactic and protocol errors. Although we usually assume that a DFC feature programmer would not want to deal with network or resource failures, we have discussed the possibility of enhanced APIs that give the programmer more low-level control.

9.1.2 *Addressing.* One design goal of SIP is to provide direct support for discovering and locating entities that can be communicated with. In support of this goal, it includes a subprotocol for registering clients with servers.

The goal of DFC is to enable all feature functions without constraining any of them. For this reason, DFC is designed to be as featureless as possible; no feature functions are built into DFC.

SIP's built-in approach is a problem with respect to service creation because it embodies a definite bias toward two-level addressing. A lower-level address represents a device, while a higher-level address usually represents a person. Yet there can be additional levels. For example, Figure 4 shows a group address *g2* being used as a target. Such an address can also be used as a source address

in DFC, so that a sales agent working from home could identify himself as a sales agent and gain access to the billing, directories, and features belonging to that role.

9.1.3 *Signaling.* SIP is designed so that servers can be stateless with respect to individual sessions. The SIP approach values stateless servers for their efficiency and scalability. On the other hand, statelessness creates problems that seem to outweigh its value in telecommunication systems.

SIP messages carry an address stack so that SIP servers along an end-to-end signaling path can implement the path without maintaining any state concerning it. This propagates addresses along the entire signaling path, which undermines many privacy features that operate by localizing address information [Zave 2002]. It also means that address headers can become quite large, which is causing implementation problems.

It might be possible to hide some address information through encryption. However, this was deprecated by the original SIP standard, and is a potential source of security leaks. Also, it does not solve the large-header problem.

The default assumption is that a SIP server participates in the setup of a signaling path, but does not persist in it throughout the duration of the call.[8] If a server does not persist in the signaling path, it cannot provide mid-call functions, even though mid-call features are common in telecommunications. Also because of the default assumption, SIP messages commonly carry information that enables one server to pass responsibility to another. For example, SIP has formal syntax to indicate "the intended user cannot be found at address *a*, but can instead be found at address *b*."

The cost of passing responsibility is a virtually open-ended signaling language in which servers send ever-more-elaborate diagnostic information, and must be ever-more-elaborate to receive and use it properly. Returning to the example "the intended user cannot be found at address *a*, but can instead be found at address *b*," SIP syntax makes it possible to indicate whether the move is permanent or temporary, and if temporary, how long it will last.

DFC has no such syntax. It can be simpler because there is rarely any passing of responsibility—the same feature box that diagnoses a condition remains in the signaling path and treats the condition.

For example, if a DFC feature box receives an internal call with target *a* and has the knowledge "the intended user cannot be found at address *a*, but can instead be found at address *b*," then the feature box simply continues the usage by placing an outgoing call with target *b*. When *a* stops being located at *b*, the feature box will stop forwarding *a*'s calls there. The feature box has no need to tell any other part of the system how long *a* will be located at *b*.

9.1.4 *Media Control.* Through the use of SIP, a media device learns which other device it should be communicating with, and the desired characteristics of the RTP stream. Then the devices themselves set up the RTP stream. Usually

[8]This default can be overridden with the use of the `RecordRoute` header.

this happens because one of the devices initiated the communication. It can also be the result of *third-party call control* [Rosenberg et al. 2002a], which means that the communication was initiated by a server. The server used SIP to contact both devices.

The difference between this and media control in BoxOS is profound, because BoxOS is compositional with respect to media, and SIP is not. Section 7 explains in detail how multiple DFC features can manipulate the same media stream simultaneously, and how overall BoxOS behavior reflects the correct composition of those features.

Consider, on the other hand, the problem of implementing the usage of Figure 3 using SIP. Call Waiting is provided by the server of x, and Prepaid Card is provided by the server of y. The Call Waiting logic instructs x to switch its media stream between z and w. The Prepaid Card logic instructs w to switch its media stream between r and x. The devices have no intelligent way to compose or choose among these various conflicting instructions, so the probability of correct overall operation is very small.

Because the signaling between device x and device w is likely to travel through the servers of both features, it would be possible to solve the problem by explicitly programming Call Waiting and Prepaid Card to cooperate with each other. However, this cooperation will work reliably only when Call Waiting and Prepaid Card are the only features in the usage that manipulate media. This is not realistic given SIP's goal of "integrating multimedia communications, such as voice, with Web, e-mail, buddy lists, instant messaging, and online games. Whole new sets of features, services, and applications become conceivable" [Rosenberg and Shockey 2000]. The complexity of programming media cooperation explicitly is sure to increase exponentially with the number of features.

9.1.5 *The User Interface.* An attempt to place a call usually has an outcome. A voice-only device communicates this outcome to its user through tones such as busytone, fast busy, and ringback.

The actions of features can complicate this picture in two ways. First, they can cause a new outcome phase to occur after there is already an outcome (as in Section 2) or when the call is already in the talking state. Second, they can cause an endpoint *that received the call* to later receive outcome signals. In other words, the direction of the outcome can be reversed.

A Click-to-Dial feature does both. It calls the first party. After the first party has answered, the feature places a second call and connects it to the first, so that the first receives outcome signals from the second.

The design of SIP does not anticipate these possibilities. As a result, once a voice call has reached a talking (connected) state, there cannot be further outcome signals. The design of DFC anticipates these possibilities, and does not incorporate any unnecessary restrictions.

9.1.6 *End-to-End Principles.* It may seem that the SIP approach to signaling conforms to end-to-end principles [Saltzer et al. 1984], while the DFC approach does not.

This is a little unfair, for two reasons. Although SIP allows stateless servers, in practice they are becoming rare. SIP application servers are appearing everywhere, and they are as stateful as DFC feature boxes and BoxOS hosts.

Also, there are two possible meanings for *end* in "end-to-end." One meaning distinguishes user endpoints from the nodes in the network that provide services for them. By this definition, DFC and BoxOS are not end-to-end.

The other meaning distinguishes applications from the lower network layers shared by all applications. All of BoxOS is a network application. By this definition, BoxOS does conform to "end-to-end" principles such as putting reliability in the application rather than the underlying network.

9.2 SIP-Based Media Resources

When the BoxOS media implementation (Figure 10) was first designed about two years ago, the two open standard protocols for the control of RTP switch/mixers were MGCP and Megaco, both originated by the IETF. Because the connection model of MGCP is more limited and did not satisfy our requirements, and because Megaco was intended to replace MGCP as the IETF standard, we selected Megaco as the protocol between media managers and switch/mixers.

At the time, no commercial products were available, so we developed our own Megaco protocol stack and Megaco media gateway that supports RTP switching capability.

It turned out that the adoption of Megaco has not been as swift as hoped. To date there are only a few commercial Megaco media gateways with mixing capability, none of which fully meet our needs. So we have been left with no viable implementation of a full Megaco switch/mixer.

During the same period, there has been considerable development of SIP-based media resources. A new breed of RTP mixers that uses SIP as the control protocol (called a media or conference server in SIP terminology) has begun to appear in the market. Most SIP devices today support switching of media connections mid-call via the re-INVITE mechanism.

We are now using SIP conference servers for our mixing technology. The cost of this change is that it is not currently possible to provide mixing as a primitive to box programmers. To mix media streams, a box program must call a conference resource, switch the media streams so that they go to the resource at the right time, and control the resource to obtain the right mix.

On the bright side, the fact that a "switch/mixer" no longer mixes, opens up the possibility of using the switching capabilities of SIP devices. Each SIP device could have its own media manager, to which all the boxes associated with the device are assigned. The media manager could control the SIP device as its own switch/mixer. This architecture provides the most direct media paths possible.

One aspect of media processing in which all voice-over-IP technology is still very immature is carriage of DTMF tones. Standards are not yet in place, and implementations are lacking altogether. Presumably DTMF has been neglected because people have been relying on the unlimited signaling potential of IP.

However, as mentioned in Section 8.1, compatibility with the PSTN is still a very real requirement.

Overall, this experience highlights the risk of relying on emerging standards, and the importance of designing for change. Because of the three-tier architecture as shown in Figure 10, the feature boxes are insulated from most changes in the media and switch layers, and can remain unchanged while the underlying implementation is overhauled drastically.

9.3 Using BoxOS as a SIP Application Server

A SIP *application server* is a powerful, stateful SIP server. The need for application servers is becoming more evident to the SIP community, as an application server can perform many vital functions that SIP devices and stateless servers cannot, for example:

- It can provide media processing, such as mixing, monitoring, recording, and playback, that most user devices do not support.
- It can provide mid-call features.
- It can initiate new calls (this is third-party call control).
- It can use data that may not be available at every endpoint.

Application servers also allow service code to be maintained centrally, which is clearly advantageous because it is complex and changes frequently.

Our original conceptions of DFC and BoxOS were as stand-alone systems, in full control of all their endpoints and the features that affect them. Not surprisingly, these conceptions have yielded to steady pressure from a world full of other telecommunication networks, equipment, and protocols.

As a result of the interoperability of BoxOS with SIP, we have been able to package and deploy a single-host BoxOS domain as a SIP application server. In this context, BoxOS and Boxware act as a service-creation environment for SIP services. The services are invoked by provisioning SIP addresses to subscribe to them, and by routing SIP calls through the BoxOS domain. The domain is transparent to calls to which no feature applies.

Service creation for such an application server can make full use of the modularity and compositionality of DFC and BoxOS, at least within the scope of the server. Obviously there is no way that BoxOS, packaged within SIP in this way, can solve SIP's more global service-creation problems.

10. COMPARISON TO OTHER RELATED WORK

10.1 The Intelligent Network Architecture

The Intelligent Network (IN) architecture [Duran and Visser 1992; Garrahan et al. 1993] is a standard of the International Telecommunication Union and the European Telecommunication Standard Institute. Its original goal was to separate feature code from PSTN switches, so that PSTN service providers could upgrade their switches without rewriting their (very expensive) feature code. Feature code is isolated in network nodes called Service Control Points

(SCPs); these nodes have signaling connections with switches, but no media connections.

IN is relevant to IP telecommunication because it is still the best-known approach to feature development and feature interaction in telecommunications. For this reason, it is often proposed as a way to add features to "softswitches," i.e., IP-based telecommunication switches.

The centerpiece of the IN Conceptual Model is a fixed finite-state machine called the Basic Call Process (BCP).[9] The BCP represents switch code, and defines control points such as the point where a destination address has been received from a calling user. When an instance of the BCP arrives at a specific control point, this can trigger a remote procedure call from the switch to an SCP. The procedure call in the SCP can perform actions such as translating the address received from the user to another address stored in a database. When the procedure call returns, execution of the BCP continues, using the translated address returned by the procedure call.

The IN standard specifies the exact interface between BCPs and SCPs. Several vendors have proprietary architectures for SCPs, providing some feature composition in the procedures it provides.

Roughly speaking, the scope of a BCP is a device and the features to which its address subscribes. Thus a fixed set of states must capture everything important about a device and its features, at least from the perspective of signaling and call control. For example, each version of the BCP specifies a maximum number of parties that can be connected. Also, "connecting" a party means connecting exactly one voice channel on behalf of the party.

The corresponding state of a DFC usage is distributed across any number of feature boxes, can connect any number of parties, and can be expanded arbitrarily by adding more feature boxes. It also provides for the possibility that a call has any number of media channels.

Furthermore, in a DFC usage there can be feature boxes not directly associated with any device, because they are subscribed to by mobile addresses. Mobile addresses are an important and much-used part of DFC.

Clearly these differences mean that DFC is able to describe a far richer feature set than the IN Conceptual Model. The ambitions for IP telecommunications are much too broad to be encompassed by the IN Conceptual Model.

10.2 The Pattern Language Architecture

Several Nortel switches, supporting both cellular (GSM) and IP (SIP, H.323) voice protocols, have a service architecture based on design patterns [Utas 1998]. This architecture has many similarities to DFC. It shares DFC's goals of feature modularity and easy extensibility.

In the Pattern Language (PL) architecture, as in DFC, each feature is implemented as a finite-state machine, and many feature pairs interact with each other by means of internal calls using a common ISUP-like protocol. As in

[9]More precisely, it is fixed for each IN Capability Set, which is a version or release of the IN Conceptual Model.

DFC, multi-port features create forks and joins in the usage graph of features and calls.

In the PL architecture, the originating or terminating half of a basic call is a *parent*. A parent can have a number of *modifiers* that override or augment its behavior. Some of the features are modifiers in this sense.

One important difference between the PL architecture and DFC concerns the assembly of a usage graph. In the PL architecture the graph configuration is largely fixed, although it is altered occasionally by commands issued by one feature and naming another feature to be added. In DFC no part of the graph configuration is fixed. It is completely determined by the action of the routing algorithm on the setup signals emitted by boxes as they execute. Each address can subscribe to different features, yet no feature ever names another, so the feature set of an address can be changed without changing any code.

Another important difference concerns protocol conversion. The PL architecture has no interface boxes. The conversion between various external protocols and the common internal protocol is performed several steps into the usage graph, so that features near the border of the usage graph are dependent on the external protocol they are serving. In DFC the only thing dependent on an external protocol is the interface box for it, which needs to be programmed only once.

Another important difference concerns the mechanisms for feature interaction. In DFC the *only* mechanism is the internal call. Since DFC is formally defined, and everything not expressly permitted is forbidden, it is possible to anticipate all the ways that other features can interact with a feature of interest. In the PL architecture there are additional mechanisms, including:

- A modifier feature can override and augment the behavior of its parent.
- A modifier feature can execute an event handler in the context of its parent, thus changing the state of its parent.
- All the features applied to a basic call share a database associated with that call.
- A feature can prevent the initiation of another feature.
- The features of a call running on behalf of the same user form a non-preemptable scheduling unit, enabling them to ignore many race conditions that would otherwise arise.

In fact the PL architecture is defined by informal patterns that act as guidelines for features. The feature-interaction mechanisms are not completely specified, nor are they guaranteed to be exhaustive.

Systems built with the PL architecture may be easily extensible by their inner circle of developers. Because of the differences between PL and DFC, however, they are unlikely to be programmable by customers.

10.3 ICEBERG

The ICEBERG project [Raman et al. 1999; Wang et al. 2000] has many goals in common with Building Box. Most notably, both projects have the

following goals:

- There should be a high degree of interoperation.
- The system should be extensible, and service creation should be easy.
- The system should provide a rich variety of services. For example, all personal services should be available to their customer from all locations and devices.

One important difference between ICEBERG and BoxOS is that service concepts, efficiency considerations, and reliability considerations are all intertwined in the design of ICEBERG. Also, ICEBERG runs on a particular high-availability cluster-computing platform. In BoxOS these concerns are separated as much as possible. Because we would like to make it easy for any Internet node to be an BoxOS host, BoxOS software runs on several of the most common platforms. We hope to improve efficiency and reliability gradually by working on the layers under the service layer, without changing the service layer or existing services.

ICEBERG is based on service concepts that are more powerful and inclusive than the ones in DFC. The building blocks of DFC are simpler and lower-level. The advantage of higher-level concepts is that more functionality is built in. The advantage of lower-level concepts is that they are more flexible and general.

For example, Wang et al. [2000] describes ICEBERG as achieving all communication through the primitive, ubiquitous concept of a multidevice/multiparty call. Participation in such a call is by invitation, and any call participant can invite new devices/parties to join the call. This concept can implement many features, but it cannot implement Call Waiting, because the essence of Call Waiting is that an unexpected and uninvited call joins your communication cluster.

In contrast, the communication primitive of DFC is a point-to-point call. More interesting structures such as conferences are built up by feature boxes. A conference or other linkage formed by one feature box composes easily with structures formed by other feature boxes, through point-to-point calls, so that the overall achievable functionality is unlimited.

For another example, ICEBERG is based on the concept of personal communication services. An address represents a person or a device, and each person has a rich set of communication services.

In DFC it is common to have personal addresses that represent people, and for those addresses to subscribe to a rich set of features supporting personal communication needs. At the same time, there can be other addresses representing such entities as groups, roles that people play, anonymous aliases, and scheduled meetings. These addresses also subscribe to appropriate features. Their use composes easily with the use of personal addresses. Figure 4 shows a call to a group being directed by a group feature to a person who is currently representing the group. Section 9.1.2 points out that a person who is placing a call can assume the group identity and use its calling features. It is not clear how a service-creation environment based solely on personal services can include such capabilities.

11. CONCLUSION

We have described an architecture to support the next generation of telecommunications, which will be based on IP.

The emphasis on feature composition makes this architecture radically different from related architectures. Compositionality is a discipline that permeates all aspects of the system, from its highest-level programming interfaces to its lowest-level resource management.

Compositionality is the right place to start a design for telecommunications, because isolated application programs are nearly useless for this purpose. We have developed solutions for problems that all IP-based telecommunication systems will encounter when they have developed enough features and had enough experience with real users.

Obviously, compositionality exacts a cost in performance. Although we have just begun to address performance issues seriously, we are seeing rapid improvements, and we are confident that competitive performance is achievable.

ACKNOWLEDGMENTS

We are very grateful for the contributions of our colleagues Michael Balk, Andrew Forrest, Healfdene Goguen, Glenn Hochberg, Michael Jackson, Gerald Karam, Andrea Skarra, Tom Smith, and Xiaotao Wu.

REFERENCES

BLUMENTHAL, M. S. AND CLARK, D. D. 2001. Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world. *ACM Trans. Internet Tech. 1*, 1, 70–109, August.

BOND, G. W. 2000. Fault management issues for a next-generation IP telecom services architecture. In *Workshops and Abstracts of the International Conference on Dependable Systems and Networks (DSN 2000)*, pages D-11–D-13, June.

BOND, G. W. AND GOGUEN, H. 2002. ECharts: Balancing design and implementation. In *Proceedings of the 6th IASTED International Conference on Software Engineering and Applications*, pages 149–155. ACTA Press.

BOND, G. W., IVANČIĆ, F., KLARLUND, N., AND TREFLER, R. 2001. ECLIPSE feature logic analysis. In *Proceedings of the Second IP Telephony Workshop,* pages 49-56. Columbia University, New York, NY, April.

BOUMA, L. G. AND VELTHUIJSEN, H. ED. 1994. *Feature Interactions in Telecommunications Systems.* IOS Press, Amsterdam.

CALDER, M. AND MAGILL, E., EDS. 2000. *Feature Interactions in Telecommunications and Software Systems VI*, IOS Press, Amsterdam.

CAMERON, E. J., GRIFFETH, N. D., LIN, Y.-J., NILSON, M. E., SCHNURE, W. K., AND VELTHUIJSEN, H. 1993. A feature-interaction benchmark for IN and beyond. *IEEE Comm. 31*, 3, 64–69, March.

CHENG, K. E. AND OHTA, T., EDS. 1995. *Feature Interactions in Telecommunications Systems III*, IOS Press, Amsterdam.

CHEUNG, E., JACKSON, M., AND ZAVE, P. 2002. Distributed media control for multimedia communications services. In *Proceedings of the 2002 IEEE International Conference on Communications*, Symposium on Multimedia and VoIP—Services and Technology, IEEE Communications Society, 02CH37333C.

CHOI, S., TURNER, J., AND WOLF, T. 2001. Configuring sessions in programmable networks. In *Proceedings of IEEE Infocom.*

CLARK, D. D. 1988. The design philosophy of the DARPA Internet protocols. *Comput. Comm. Rev. 28*, 4, 106–114, August.

DINI, P., BOUTABA, R., AND LOGRIPPO, L., EDS. 1997. *Feature Interactions in Telecommunication Networks IV.* IOS Press, Amsterdam.

DURAN, J. M. AND VISSER, J. 1992. International standards for intelligent networks. *IEEE Communications 30*, 2, 34–42, February.

GANSNER, E. R., MOCENIGO, J. M., AND NORTH, S. C. 2003. Visualizing software for telecommunication services. In *Proceedings of ACM SOFTVIS 2003*, pages 151–157, 215–216.

GANSNER, E. R. AND NORTH, S. C. 2000. An open graph visualization system and its applications to software engineering. *Software—Practice & Experience*, *30*, 11, 1203–1233.

GARRAHAN, J. J., RUSSO, P. A., KITAMI, K., AND KUNG, R. 1993. Intelligent Network overview. *IEEE Communications 31*, 3, 30–36, March.

HALL, R. J. 2000. Feature interactions in electronic mail. In *Feature Interactions in Telecommunications and Software Systems*, M. Calder and E. Magill, Eds. pages 67–82.

JACKSON, M., AND ZAVE, P. 1998. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Trans. Soft. Eng. 24*, 10, 831–847, October.

JACKSON, M. AND ZAVE, P. 2001. The DFC Manual. AT&T Research Technical Report, August. Available at `http://www.research.att.com/projects/dfc`.

KIMBLER, K. AND BOUMA, L. G., EDS. 1998. *Feature Interactions in Telecommunications and Software Systems V.* IOS Press, Amsterdam.

KRISTENSEN, A. SIP Servlet API, Version 1.0. Dynamicsoft, Inc.

MCGLASHAN, S., BURNETT, D., DANIELSEN, P., FERRANS, J., HUNT, A., KARAM, G., LADD, D., LUCAS, B., REHOR, K., PORTER, B., AND TRYPHONAS, S. 2002. VoiceXML 2.0, W3C Working Draft, `http://www.w3.org/TR/voicexml20`, 24 April.

NORTH, S. C. AND WOODHULL, G. C. M. 2002. Online hierarchical graph drawing. In *Proceedings of the Symposium on Graph Drawing GD'01*, pages 232–246. Springer-Verlag LNCS 2265.

RAMAN, B., WANG, H. J., SHIH, J., JOSEPH, A. D., AND KATZ, R. H. 1999. The ICEBERG project: Defining the IP and telecom intersection. *IEEE IT Professional,* November/December, pages 38–45.

ROSENBERG, J., MATAGA, P., AND SCHULZRINNE, H. 2001. An application server component architecture for SIP. IETF Internet Draft, SIP Working Group, 2 March.

ROSENBERG, J., PETERSON, J., SCHULZRINNE, H., AND CAMARILLO, G. 2002a. Best current practices for third party call control in the Session Initiation Protocol. Internet Engineering Task Force, work in progress, 5 June.

ROSENBERG, J. D. AND SCHULZRINNE, H. 2002. Guidelines for authors of SIP extensions. Internet Engineering Task Force, work in progress, 1 March.

ROSENBERG, J., SCHULZRINNE, H., CAMARILLO, G., JOHNSTON, A., PETERSON, J., SPARKS, R., HANDLEY, M., AND SCHOOLER, E. 2002b. SIP: Session Initiation Protocol. IETF Network Working Group, Request for Comments 3261.

ROSENBERG, J. D. AND SHOCKEY, R. 2000. The Session Initiation Protocol (SIP): A key component for Internet telephony. *Computer Telephony 8*, 6, 124–139, June.

SALTZER, J., REED, D., AND CLARK, D. D. 1984. End-to-end arguments in system design. *ACM Trans. Comput. Syst. 2*, 4, 277–288, November.

SCHULZRINNE, H., NARAYANAN, S., LENNOX, J., AND DOYLE, M. 2002. SIPstone: Benchmarking SIP server performance, `http://www.sipstone.org`, 12 April.

SHAW, M. AND GARLAN, D. 1996. *Software Architecture.* Prentice-Hall.

SKARRA, A. H., HANSON, K. J., KARAM, G. M., AND ELLIOTT, J. S. 2001. The iStudio environment: An experience report. In *Proceedings of the International Workshop on XML Technologies and Software Engineering,* Toronto, Canada, May.

TSANG, S. MAGILL, E. H., AND KELLY, B. 1997. The feature interaction problem in networked multimedia services—present and future. *British Telecom Tech. J. 15*, 1, 235–246, January.

UTAS, G. 1998. A pattern language of feature interaction. In *Feature Interactions in Telecommunications and Software Systems*, K. Kimbler and L. G. Bouma Eds. pages 98–114.

WANG, H. J., RAMAN, B., CHUAH, C.-N., BISWAS, R., GUMMADI, R., HOHLT, B., HONG, X., KICIMAN, E., MAO, Z., SHIH, J. S., SUBRAMANIAN, L., ZHAO, B. Y., JOSEPH, A. D., AND KATZ, R. H. 2000. ICEBERG: An Internet core network architecture for integrated communications. *IEEE Pers. Comm.* August, pages 10–19.

ZAVE, P. 2002. Address translation in telecommunication features. To appear in *ACM Trans. Soft. Eng. and Meth.* 2004.

ZAVE, P. 2003. An experiment in feature engineering. In A. McIver and C. Morgan, eds., *Programming Methodology,* pages 353–377. Springer-Verlag, New York.

ZAVE, P., GOGUEN, H. H., AND SMITH, T. M. 2003. Component coordination: A telecommunication case study. To appear in Elsevier's *J. Comp. Netw.* 2004.

ZAVE, P. AND JACKSON, M. 2002. A call abstraction for component coordination. In *Proceedings of the Twenty-Ninth International Colloquium on Automata, Languages, and Programming: Workshop on Formal Methods and Component Interaction*.

Received October 2002; accepted July 2003