Address Translation in Telecommunication Features

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

23 December 2003

Abstract

Address translation causes a wide variety of interactions among telecommunication features. This paper begins with a formal model of address translation and its effects, and with principles for understanding how features should interact in the presence of address translation. There is a simple and intuitive set of constraints on feature behavior so that features will interact according to the principles. This scheme (called "ideal address translation") has provable properties, is modular (explicit cooperation among features is not required), and supports extensibility (adding new features does not require changing old features). The paper also covers reasoning in the presence of exceptions to the constraints, limitations of the theory, relation to real networks and protocols, and relation to other research.

keywords: component architecture, feature interaction, formal methods, network addressing, network protocols, network security, requirements, telecommunications

1 Introduction

Telecommunications is networking with an emphasis on real-time communication among people. 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 voice (voice mail) and text (electronic mail). Mail is included both because it is simply buffered person-to-person communication, and because mail is often used as a backup when an attempt at conversation fails.

The functions that a telecommunication network performs for its users, on top of basic services, are called *features*. As these networks evolve, new features are continually being added. They often interact in unexpected or problematic ways, which has given rise to a great deal of research on the *feature-interaction problem* [5, 6, 7, 9, 11, 20].

The goal of research in feature interaction is to manage feature interactions, which means preventing the bad ones and enabling the good ones. Although this has proven to be difficult for many reasons, two prominent ones are the need for extensibility and the lack of telecommunication requirements.

Extensibility is at the heart of the feature-interaction problem. Many of the new features added to a telecommunication network are based on concepts or technologies that were not anticipated when software development for the network began. Their number quickly mounts into the hundreds or even thousands. Yet it must be possible to add new features without undue effort, and without compromising the future extensibility of the software.

There are no widely accepted requirements for telecommunications, in the sense of well-defined behavioral properties that all telecommunication networks should satisfy. This situation is due to many factors, among them the long life cycle and incremental nature of software development, the conflicting goals of various users, and the ambiguity of telecommunication concepts (see below). This lack exacerbates the feature-interaction problem by making it difficult to distinguish good interactions from bad ones, or, in other words, to determine how a telecommunication network *should* behave.

The goal of this work is to manage one category of feature interactions, those caused by address translation.

1.1 The problem of feature interactions caused by address translation

In every telecommunication protocol, requests for communication carry at least two addresses: a *source*

address indicating which object is making the request, and a *target address* indicating which object's participation is being requested. *Address translation* is a function performed by some features; it consists of modifying a request for communication by changing its source address, target address, or both.

Address translation is a very common feature function (any kind of "call forwarding" is translation of the target address). It also causes a wide variety of feature interactions. Thus address translation causes a huge number of feature interactions—possibly more than any other feature function.

A typical question about the effects of address translation begins, "If calls to a are being forwarded to b, then should . . . . " Such a question is usually impossible to answer because it does not tell us what a or b identifies or represents, nor does it tell us what the forwarding is supposed to accomplish on behalf of a. This is an example of the pervasive ambiguity of telecommunication concepts.

Once the addresses and features are associated with concepts and purposes in the users' domain, the ambiguity is reduced and judgments become possible. Then examples of bad feature interaction due to address translation are easy to find.

In one example, a customer calls a sales group. The call is forwarded to a sales representative; since the sales representative is not available, his voice mail offers to take a message. It would be much better for the failure to re-activate the group feature to find another representative. Because of forwarding, both group and personal features are invoked, and they interact badly.

In another example [14], two people with addresses *user1@host1* and *user2@host2* correspond by electronic mail. Since *user2@host2* wishes to remain anonymous in this correspondence, he is known to *user1@host1* as *anon2@remailer*, and the anonymous remailer retargets electronic mail for *anon2@remailer* to *user2@host2*.

However, *user2@host2* also has an autoresponse feature set to notify his correspondents that he is on vacation. When electronic mail arrives with source address *user1@host1* and target address *user2@host2*, it immediately generates a response with source address *user2@host2* and target address *user1@host1*. When *user1@host1* receives the response, he learns the identity of his anonymous correspondent. Thus the autoresponse feature undermines the purpose of anonymous remailing.

1.2 Outline of a solution

The first component of a proposed solution is a formal model of the aspects of telecommunications related to address translation (Section 2). The model makes it possible to formalize address translation and its effects, so that feature interactions can be predicted and management mechanisms can be defined.

The model itself is part of the solution, as it has a capability that is not found in many telecommunication protocols. Section 2 explains its relation to real telecommunication protocols, including how its extra capability can be simulated with other protocols.

The second component of a proposed solution is a classification of feature interactions caused by address translation, along with principles for evaluating these interactions as desirable or undesirable (Section 3). The principles balance conflicting desires and design criteria, so that all can be satisfied to a reasonable degree. They are an attempt to work toward true requirements for telecommunications.

The third and central component of a proposed solution is the concept of *ideal address translation* (Section 4). This is a form of address translation intended to capture intuitively what address translation could and should be. It constrains *how* address translation is performed, without constraining significantly *what* address translation can accomplish.

Ideal address translation is based on *address categories*. Address categories reduce the usual ambiguity about why features are functioning, and on whose behalf. It then becomes easier to apply the principles from Section 3, and to understand how features should interact.

A feature set in which all features obey the constraints of ideal address translation automatically satisfies desirable properties derived from the principles. It is modular, in the sense that features do not need to know about each other or to cooperate explicitly. It is extensible, in the sense that adding or deleting features or other objects does not require changing the existing or remaining ones. Its organization facilitates reasoning about other properties as well.

In real telecommunication networks, exceptions to ideal address translation are inevitable. Most commonly, they will occur because of legacy subsystems that cannot be changed, or because of discrepancies between the formal model and a real network protocol. Section 5 shows how the general-purpose reasoning embedded in ideal address translation can be adapted to special-purpose reasoning in the face of exceptions.

Section 6 summarizes the arguments for the valid-

ity of ideal address translation, and enumerates its limitations. Section 7 relates it to other research.

2 A formal model of request chains

2.1 Informal description

In telecommunications, an address can be associated with a device such as a telephone, with a set of features, or both. The distinction between a device and a feature set is important because a device is an interface to a person, while a feature set is a program.

All telecommunication protocols support *requests* for communication. A request carries at least two addresses: a *source address* indicating which object is making the request, and a *target address* indicating which object's participation is being requested.

All telecommunication connections are set up by *request chains*. A request chain is a chain consisting of requests and modules, where each module is an *interface module* or a *feature module*. An interface module provides an interface to a telecommunication device. A feature module instantiates a feature set.

Figure 1 shows an example of a request chain. The device with address *s1* is requesting communication with address *t2*. Its interface module initiated the request chain with those addresses. A network router routed the first request to the *source feature module of s1*, containing those features applicable to a chain whose source is *s1*.

A module *continues* a request chain by making an outgoing request that corresponds to an incoming request it has already received. The source module of *s1* continued the chain, in doing so changing the source address from *s1* to *s2*. As a result of this change, a network router routed the outgoing request to the source feature module of *s2*.

The source feature module of *s2* also continued the chain. With no additional source modules to route to, a network router routed its outgoing request to the *target feature module of t2*. It contains those features applicable to a chain whose target is *t2*.

The target feature module of *t2* continued the chain, first changing the target address from *t2* to *t1*. The chain was routed to the target feature module of *t1*, and then finally continued to the interface module of *t1*.

Every feature module in Figure 1 is optional. The *source region* of a chain contains all its source feature modules, while the *target region* contains all its target feature modules.

The request protocol can create a two-way *signaling channel* between the sender and receiver of the request. Such a channel persists until it is deliberately destroyed, which can be done from either end.

It is common for a feature module, on receiving a signal through the signaling channel of a request, to send the same signal out on the signaling channel of the request's continuation. Because of this common behavior, there can be a two-way *signaling path* traversing the entire length of the chain. This is the only path on which signals can travel among the modules of the chain.

2.2 Formal definition

There is a set of *addresses*. Each address can be associated with an interface, a source feature set, a target feature set, or any subset of the three (including the empty set).

There is a set of *modules*. A module is an *interface module* or a *feature module*. An interface module is an instantiation of an interface. A feature module is an instantiation of a feature set.

A request for communication has four mandatory fields. The *source* and *target* fields contain addresses. The *region* field has two possible values, which are `srcRegn` and `trgRegn`. It indicates whether the request is part of the source region or target region of a request chain. The *status* field has two possible values, which are `changed` and `unchanged`. As will be shown, it provides history that is necessary for request chains to unfold correctly.

A module behaves as a concurrent process that can participate in the request protocol. Whenever a module issues a request for communication, the request goes to a router which executes the routing algorithm specified below to find a next module to route the request to. The request goes to the next module, which then replies directly to the requesting module.

A request chain can be initiated by an interface module or by a feature module. A module that initiates a chain emits a request with:

```
source == <address of initiating module>
region == srcRegn
status == changed
```

A request chain that has led to a feature module can be *continued* by the feature module. The feature module has received an incoming request; it continues the chain by emitting a corresponding outgoing request. In preparing the corresponding outgoing request, the module may change *source, target,* both, or neither. The module must, however, obey the following rules:

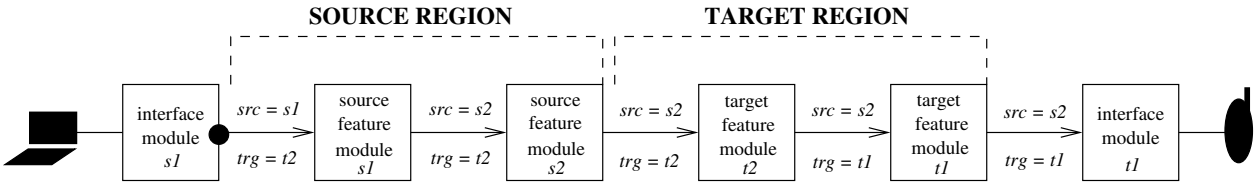

Figure 1: A request chain.

```
if (region == srcRegn) {
   if (status == changed && <source address has a source feature set>) {
      nextMod := <instance of source address's source feature set>;
      return;
   }  else {
      region := trgRegn;
      status := changed;
   }
}
// regn == trgRegn
if (status == changed && <target address has a target feature set>)
   nextMod := <instance of target address's target feature set>;
else if (<target address has an interface>)
   nextMod := <instance of target address's interface>;
else nextMod := <instance of error handler>;
return;
```

Figure 2: The routing algorithm (pseudocode is delimited by angle brackets).

- The outgoing *region* is the same as the incoming *region*.
- If the *region* is `srcRegn`, then if the feature module has changed *source* the outgoing *status* is `changed`. Otherwise the outgoing *status* is `unchanged`.
- If the *region* is `trgRegn`, then if the feature module has changed *target* the outgoing *status* is `changed`. Otherwise the outgoing *status* is `unchanged`.

In summary, feature modules can change the *source, target,* and *status* fields, but cannot change the *region* field. If a module has changed either the source or target address while continuing a request chain, it has performed the function of *address translation*.

A router contributes to request chains by receiving requests emitted by modules, and routing them to next modules chosen by the algorithm in Figure 2. In addition to choosing a next module, the algorithm can also change the *region* and *status* fields of the request, so that the next module may receive an altered request. A error handler handles routing errors. For the purposes of this paper, it is indistinguishable from an interface of the target address.

In the source region, the algorithm chooses a source feature module if and only if the source address has a source feature set *and* the source address was changed by the feature module that issued the request. If the source address is unchanged, then an instance of the source address's source feature set is already in the request chain—it is, in fact, the module that issued the request.

If the algorithm does not choose a source feature module, then the source region is complete; the router advances to the target region and continues the routing process. Routing for the target region is similar, except that if there is no target feature module to route to, then the target region is complete, and the router routes to the target's interface module.

A *source region* of a request chain (if any) is a maximal subchain in which every request has its final *region* equal to `srcRegn` and every module is a source feature module to which one of those requests is routed. A *target region* of a request chain (if any) is a maximal subchain in which every request has final *region* equal to `trgRegn` and every module is a target feature module to which one of those requests is routed. Note that if a request chain is initiated by a feature module, the initiating feature module does not belong to either region.

The most fundamental property of request chains is established by the Region Lemma.

Region Lemma: A request chain has at most one source region and at most one target region. The source region precedes the target region.

Proof:

(1) A request chain is initiated by a request with *region* equal to `srcRegn` (definitions).

(2) The *region* field is propagated unchanged through the request chain, by both feature boxes and executions of the routing algorithm, except that the routing algorithm can change its value from `srcRegn` to `trgRegn` (definitions). Clearly this can only happen once.

(3) If the final *region* of a request is `srcRegn` [`trgRegn`], then if it is routed to a feature module, it is a source [target] feature module (definitions).

(4) Only a feature module can continue a request chain, so if a request is routed to an interface module or an error handler, the request is the last element of its request chain. □

The formal model of request chains does not constrain the number of instances of any feature set or interface, but the number may be constrained by the network. For example, it is common for the address of a device to have exactly one interface module, to which all requests are routed.

All signaling channels among interface and feature modules are created by the request protocol, so that signals can travel only along the paths of request chains. The formal model does not constrain how long any signaling channel persists before it is destroyed. Signals among interface and feature modules are used to determine when and where media channels are needed, but direct control of media channels takes place at a lower level of abstraction.

2.3 Other examples

Figure 3 shows a source feature module that provides a Three-Way Calling feature. It continues the same incoming request from its subscriber twice, once when the subscriber initiates the request chain, and once when the subscriber invokes the feature. As it maintains both continuations simultaneously, the source feature module and its incoming request belong to two distinct request chains.

Request chains for a broadcast would look the same, except that there would be many more of them. The main difference between a broadcast and a conference is that, in a broadcast, media flows in only one direction.

Figure 4 shows a feature module that provides a Call Waiting feature. Here the same module instantiates both the source and target feature sets of x. The incoming request from z arrived after the outgoing request to y was continued by the feature module. The feature module, knowing that the interface module of x is busy, does not continue the request from z, but rather sends x a signal that a call is waiting. As a result, the feature module is the end of the request chain with source z.

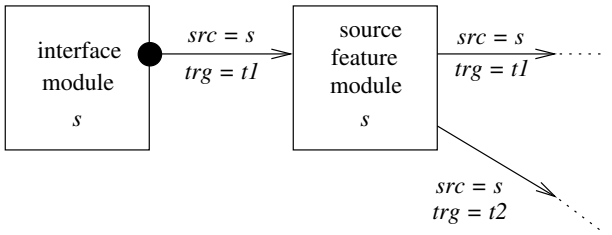

Figure 3: A feature module with Three-Way Calling.

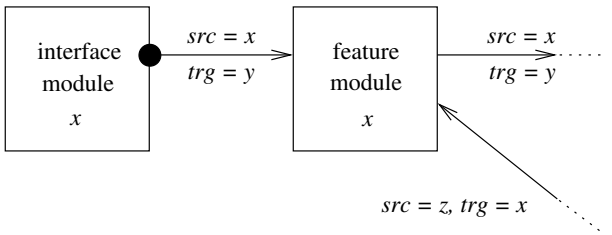

Figure 4: A feature module with Call Waiting.

Often, feature modules access persistent data stored in the network. As the distinction between a feature associated with an address and data associated with an address is not significant here, an address's data is simply viewed as part of the address's feature sets.

Figure 5 shows request chains for electronic mail using SMTP [21] and related protocols. The example is like the electronic-mail example in Section 1.1, except that no anonymity has been introduced.

The telecommunication device is a user agent running on a personal computer. "Mail host *user1@host1*" is both the source and target feature module of *user1@host1*, and "mail host *user2@host2*" is both the source and target feature module of *user2@host2*.

To write mail, the user agent initiates a request chain which is first routed to its source feature module. This feature module buffers the mail, so that the signaling channel between the user agent and the mail host can be destroyed as soon as the mail has been transmitted.

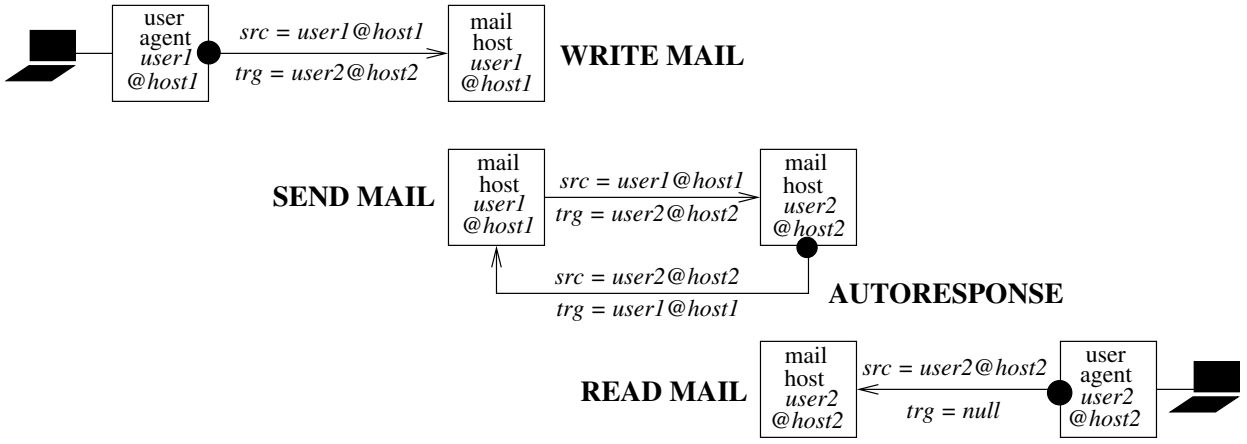

Figure 5: Electronic mail. The three snapshots arise during three different phases in the life of a message.

Electronic mail is sent by continuing the original request chain from the source feature module of the source address to the target feature module of the target address. A mail host acting as a target feature module only continues an incoming request if it is forwarding the mail. If it is not forwarding the mail, it buffers it until the addressee chooses to read it.

The target feature module of *user2@host2* includes an autoresponse feature which is currently enabled. When the module receives a request for *user2@host2*, the feature initiates a new chain with the source and target fields of the incoming request reversed. The black dot indicates the initial request of the chain. The feature module is acting automatically on behalf of its owner, and the new chain is treated exactly as if it had been initiated by the owner of *user2@host2*.

To read mail, a user agent initiates a request chain with a *null* target address.[1] Like all other request chains initiated by the user agent, this is routed to its source feature module. The chain goes no further, as its only purpose was to connect the agent with its mail host.

2.4 Relation to real telecommunication protocols

In general, the formal model of request chains is an abstraction of real telecommunication protocols. This section points out significant exceptions to the generalization.

[1] The *null* value is a distinguished value of type *address*. It is often found in the source region of a request chain, either because the chain is not intended to extend beyond the source region, or because a source feature module will interact with the caller to produce a real target address.

Most telecommunication protocols do not provide for multiple source feature modules in a request chain. The list of such protocols includes the protocols of the public switched telephone network (PSTN), SMTP [21] and other protocols for electronic mail, and the Session Initiation Protocol (SIP) for creating, modifying, and terminating multimedia IP sessions [25].

When a request chain begins, these protocols route to a source feature module, if any, associated with the initiating address. If this module continues the request chain, the next request is routed in the target region.

When services built on these protocols require multiple source feature modules, they simulate them. For example, to simulate Figure 1, the source feature module of *s1* continues the request chain with *target* equal to *s2*. Next a router routes to a target feature module of *s2*, which proceeds to behave like a source feature module.

The obvious disadvantage of this simulation is that the true target address *t2* is lost, and must be recovered with additional signaling and possibly additional user input. These cases are considered in detail in Section 5.1.

For services built using SIP, the default assumption is that a request chain will inform two interface modules of each other's addresses, after which they will send signals directly to each other, rather than along the path of the request chain. The disadvantages of this are discussed in Section 7.4.

The formal model of request chains is most closely matched by Distributed Feature Composition (DFC), which is a formally defined modular architecture for description of telecommunication services [18, 19]. DFC has an IP-based implementation [4].

Routing in DFC differs from the formal model here primarily in being finer-grained. In DFC a source or target feature set associated with one address can be instantiated by many independent modules, each one implementing a single feature. In addition to connecting feature modules with different addresses, the DFC routing algorithm also connects single-feature modules belonging to the same address. In other words, an atomic feature module in this paper corresponds to a subchain of requests and feature modules in DFC.

Another refinement present in the DFC routing algorithm is the distinction between *bound* and *free* feature modules. When a router needs an instance of a free feature module, it creates a new, interchangeable one. When a router needs an instance of a bound feature module, it finds the unique, persistent module instantiating that feature set for that address. The use of bound feature modules joins request chains into graphs, because it allows a request to be routed to a module that is already participating in another chain.

For example, in DFC an address that subscribes to Call Waiting must have the feature in both its source and target feature sets, and a feature module instantiating Call Waiting must be bound. This is how, in Figure 4, the same module instantiates both source and target feature sets, and an incoming request is routed to that module.

3 Principles for feature interaction

3.1 Ownership

Each address has one or more *owners* who are responsible for it and have rights concerning it. An owner is usually a person.

An address may have an *authentication secret*, which is assumed to be known only to its owners. If necessary, an owner must produce the secret to prove that he is an owner and thus gain access to his right to use the address.

With respect to knowledge, there is no point in distinguishing between a feature module and any owner of its address: if a secret is revealed to a feature module, the feature module can store it as data, and the data can be examined by any owner; if an owner knows a secret, he can insert it into the code or data of his feature module, so that the feature module can use it.

3.2 Address-translation functions

In telecommunications, the fundamental addresses are the addresses of telecommunication devices. However, addresses are used to identify many things besides devices.

A *group* address identifies a group of things, such as the departments of an institution, or the people of a work team. *Representation* is a feature function[2] that translates a group target address to the address of an appropriate representative of the group. *Affiliation* is a feature function that changes the source address of a request to the address of a group. This allows a representative access to the source features and data of the group.

A *mobile* address identifies a mobile object such as a person. *Location* is a feature function that translates a mobile target address, such as a personal address, to the address of a device where the person is located. *Positioning* is a function that changes the source address of a request from a device address to a personal address. This allows the person access to his personal features and data from any device.[3]

A *role* address identifies a role that can be played by another object such as a group, person, or device. *Assumption* is a feature function that changes the source address of a request to a role address, thus allowing the caller to assume that role. *Resolution* is a function that translates a target role address to the address of the object playing the role. Roles are assumed as identities, intended to reveal or conceal. For example, in Section 1.1, *anon2@remailer* is a role address.

Often these concepts are combined in the meaning of an address. For example, the address of a physicians' office identifies a group of people. It also serves as an identity (role) that is more recognizable to patients than a physician's home address, and that a physician would rather give to patients.

3.3 Ordering

The previous section showed that addresses can be categorized by what they identify. The categories mentioned are just examples; in practice, there can be many other categories.

The primary mechanism for management of feature interactions in this paper is an ordering on address categories.

[2] In this paper, a "function" is a task performed by a feature or a responsibility of a feature, rather than a mathematical function.

[3] This is software-based mobility, in contrast to *network-based* or *device* mobility, in which a device moves without changing its network address.

This ordering is called *abstraction* because it is a convenient and familiar name. In many cases, the abstraction ordering and the connotations of "abstraction" coincide. For example, it makes sense to say that a group address is more abstract than the personal mobile address of a member of the group, and a personal address is more abstract than the address of a device that the person is using. A device address is more concrete than any other kind.

On the other hand, in some cases the order of two address categories may seem unrelated to the word "abstraction." The categories are ordered so that features will interact in the most desirable ways, and for no other reason. In these cases, it is best to think of "abstraction" as an arbitrary name for a specific relation.

3.4 Identification

People and feature modules use the addresses that they know to *identify* the parties with whom they are communicating. A feature that performs address translation interacts with other features by affecting the identification information they receive in requests.

As discussed above, privacy is a motive for address translation. When it comes to privacy, there is always an inherent conflict of interests between those who wish to know and those who wish to conceal. It seems that this conflict can be resolved fairly with the following two principles:

- *Privacy:* A person should be able to conceal a more private address that he owns behind a more public address that he owns.
- *Authenticity:* A person should not be able to pose as an owner of an address he does not own.

A private address is a more concrete address, while a public address is a more abstract address.

Achieving authenticity often requires *authentication,* another feature function related to address translation. An authentication function demands an authentication secret in the form of a password, voiceprint, or some other proof that a user is an owner of a particular address.

Authentication might be needed in many different feature modules, as shown in Figure 6. In this chain, *d1* and *d2* are device addresses. The source feature module of *d1* assumes the source role of *r1*, while the target feature module of *r2* resolves that role to *d2*.

In the absence of physical protection, anyone might walk up to device *d1* and start using it. If this is not acceptable, the source feature module of *d1* must authenticate that the user is authorized to use the device, which (in the simple authorization model used here) means that the user is an owner of *d1*. The brackets indicate that the authentication secret is obtained through a dialogue between the feature module and the device/user on its left. If authentication does not succeed, the source feature module of *d1* does not continue the request chain.

The assumption of source role *r1* can only be performed by a feature module of some address other than *r1*, so anyone can program his source feature module to assume the identity of anyone else! This serious problem is solved by putting authentication that the caller is an owner of *r1* into the source feature module of *r1*, as shown in the figure. This is secure because, once the source address has been changed to *r1*, the routing algorithm *must* route the request chain to the source feature module of *r1*.

Target feature modules can also authenticate that the person who answers the telephone is the expected person. The figure shows authentication of both *r2* and *d2*, through dialogues with the device/user on the right. If authentication fails, the feature module breaks the connection before the unauthenticated person can talk to the initiating end of the chain.

3.5 Contact

People and feature modules use the addresses that they know to *contact* the parties with whom they wish to communicate. A feature that performs address translation interacts with other features by affecting the contact information they receive in requests.

The important principle for contact is:

- *Reversibility:* A target feature module or callee should be able to call the source address of a request chain and thereby target the entity that initiated it.

The principle must be worded carefully because the source address may be abstract. For example, if the chain was initiated on behalf of a group, then the group is the initiating entity, rather than the group representative who actually picked up a telephone.

Another potential contact principle is *reproducibility*—the notion that if a caller calls the same abstract address twice, his call should be directed to the same concrete address both times. Although reproducibility has certain attractions, it is a bad idea in general. It undermines the freedom of representation functions, and it makes mobility impossible. Section 4.5 shows how a modest amount of reproducibility can be provided within the guidelines.

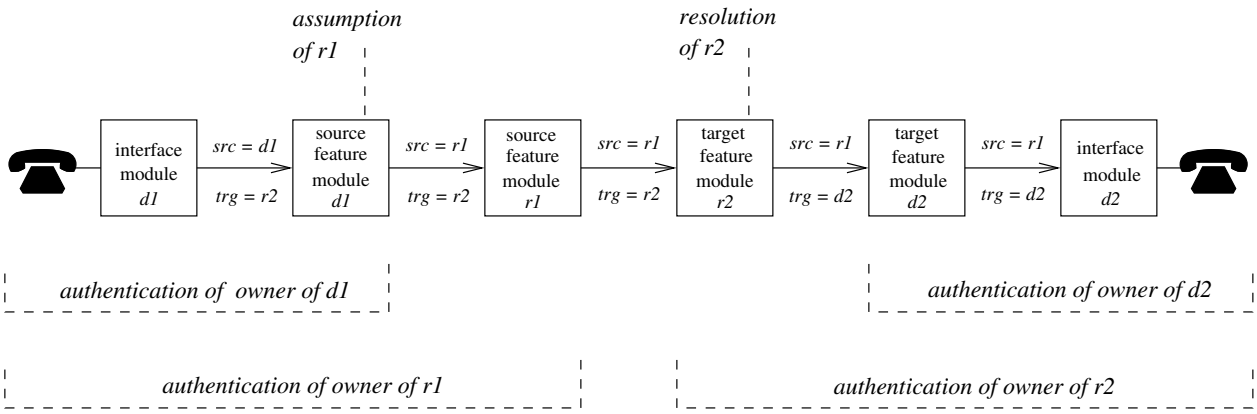

Figure 6: Many different feature modules may perform authentication.

3.6 Invocation

Through the routing algorithm, the addresses in a request chain determine which feature modules are in the chain. A feature that performs address translation interacts with other features by affecting which feature modules are invoked by appearing in the chain.

Address translation causes the invocation of multiple source feature modules and multiple target feature modules. The worst case is having an unbounded number of them, because of an address-translation loop. The need for the following principle is obvious:

- *Boundedness:* The number of source feature modules in a source region should be bounded, and the number of target feature modules in a target region should be bounded.

A more subtle issue is that address translation can cause the invocation of several feature modules, in the same region, with competing features. This is often desirable, but it requires some coordination so that competing features interact properly.

For example, often features compete with each other to handle the same situation. In the target region, several features may be treatments for *unavailability* of the target. The target interface module (or the feature module acting on its behalf) may generate a signal indicating that the target is unavailable (busy or not connected to the network). This signal acts as a trigger for any unavailability treatment that receives it.

This sharing of a triggering signal is an effective coordination mechanism for competing features in a modular, extensible system with distributed authority. Because a triggering signal must travel on the signaling path of a request chain, it acts as a token. When a feature module receives a triggering signal, it has permission to execute any feature triggered by the signal. The module itself decides when, and if, to propagate the triggering signal. When it propagates the signal, it is passing the permission token to other modules.

If competing features do not already share a triggering signal, it may be necessary to force them to. For instance, many features are treatments for a *no-answer* condition. If the telecommunication protocol being used does not have a no-answer signal, a feature triggers itself by setting a timer for some locally determined no-answer interval. This makes it very difficult to coordinate separate no-answer treatments—even if the intervals are coordinated, race conditions may disrupt any intended prioritization. The straightforward solution to this problem is to have one timer that generates a no-answer signal, which is shared among features in the same way that other failure signals are.

This coordination mechanism automatically gives higher priority to features that are closer to the source of the shared triggering signal. Many shared triggering signals are generated by the devices and interface modules at the outer ends of request chains. Source features triggered by these signals have higher priority if they are closer to the outer (initiating) end of the chain, and target features triggered by these signals have higher priority if they are closer to the outer (terminating) end of the chain. A useful organizational structure is imposed by the following principle:

- *Monotonicity:* In a region, the feature modules of more concrete addresses should be closer to the outer end of the region than feature modules of more abstract addresses.

Monotonicity serves two purposes. First, it auto-

matically gives the feature modules of more concrete addresses default priority over the feature modules of more abstract addresses in handling triggers from the outer ends of request chains. This means that most feature modules interact correctly without extra effort.

Consider, for example, a person who uses several role addresses (work, family, youth-group leader), all of which are translated by target feature modules to the same personal mobile address. The role addresses are more abstract than the personal address.

If someone calls one of the role addresses, if the feature module of the personal address attempts to locate the person at a particular device, and if the call to the device is not answered, the no-answer signal is received first by the personal feature module. It may be able to handle the no-answer condition successfully by trying another device which the person *does* answer.[4] If it cannot handle the condition, it replaces the *no-answer* signal (referring to the device) by the *unavailable* signal (referring to the person), and sends it upstream. There it triggers the feature module of the role in which the person is being called, which makes a role-based decision about what to do.

Second, monotonicity makes it possible to override the default priority in the rarer cases when default priority does not result in correct feature interaction. Consider the sales-group example in Section 1.1. A group address is more abstract than a personal address, but group features should have priority over personal features in handling the unavailability of a person. A group consists of several people, so the quickest and most effective treatment for the unavailability of one person is to find another person.

In the distributed setting of a request chain, a feature module in a target region cannot preempt a more concrete feature module in the target region in responding to a signal from the target. It can only ask the more concrete feature modules to relinquish their priority, and hope that they will cooperate.

Monotonicity is needed to accomplish this in an extensible setting. The group feature module sends a *relinquish* request downstream. Because of monotonicity, the feature module knows that the feature modules the signal will reach are exactly those feature modules in the target region whose addresses are more concrete than its own. It knows this without knowing anything about which feature modules are actually present or absent.

[4]Note that in this case the feature module is making two sequential continuations of the same request chain.

4 Ideal address translation

4.1 Addresses and address categories

In telecommunications, the usual meaning of an address is an entity that can place or receive calls. In a system with ideal address translation, there must be a global, one-to-one mapping between addresses and meanings.

In a system with ideal address translation, there is also a finite set of *address categories*. Each address belongs to exactly one category.

There is an *abstraction* relation on address categories. It is transitive (hence, an order) and irreflexive. An address can have multiple associations with other addresses, but they must all be compatible with respect to the abstraction order.

For an example of a violation that is easy to fall into, consider a new mobility service offered to office workers. Each worker already has an office telephone number, which is printed on his business card. So a worker subscribes to the mobility service by forwarding his office telephone number to his new mobile telecommunication address.

Now the office telephone number is a public role address, known to all, which is resolved to the more private mobile address. In this sense the office telephone number is more abstract than the mobile address. At the same time, the office telephone number represents a device which is sometimes the location of the mobile address. In this sense the mobile address is more abstract than the office telephone number. In ideal address translation, these two relationships between office telephone numbers and mobile addresses are incompatible.

Categorizing and ordering addresses tends to make it very clear what they identify. The resultant lack of ambiguity is very helpful in designing appropriate feature behavior and understanding feature interactions. It also has other beneficial side-effects, such as making presence information [3], which is necessarily based on addresses, more meaningful.

4.2 Constraints

A system with ideal address translation adheres to the following constraints.

The first constraint supports reversibility and other goals. If a target feature module is allowed to change the source address of a request chain, it is replacing source information with something else, for some other purpose. With true source information gone, the chain cannot be reversed.

- *Constraint 1:* A target feature module in a request chain does not change the *source* field.

Constraint 1 has no counterpart in the source region because it would be too restrictive (not because it would be useless). One of the most common functions of source feature modules is creating and modifying target addresses.

The second constraint supports all the desirable properties of ideal address translation. It recognizes that the true purpose of address translation is to change to a different level of abstraction, as all the translation functions in Section 3.2 do. The constraint forces an orderly progression through the abstraction order. The constraint is symmetric across regions and has two parts, one for each of the two regions.

- *Constraint 2s:* If a source feature module in a request chain changes the *source* field, the new source address is more abstract than the old one.
- *Constraint 2t:* If a target feature module in a request chain changes the *target* field, the new target address is more concrete than the old one.

If addresses only appeared in the *source* and *target* fields of requests, then Constraints 1 and 2 would be sufficient. Addresses also appear, however, in other signals sent on signaling channels. A third constraint is necessary to deal with them.

An address in a signal must be constrained only if it is an *alternative source/target of this chain.* Such signals are relatively common; in this paper they can be found in Figures 8 and 10. The additional constraint does not apply to addresses transmitted for purposes not directly related to the particular chain. For example, a user might employ the signaling path to update his personal data. An address transmitted only so that it can be put in a database is not directly related to the particular chain along which it is transmitted.

Constraint 3 supports the goal of privacy. Like Constraint 2 it has two parts, one for each region.

- *Constraint 3s:* A source feature module in a request chain does not transmit downstream, as an alternative source of the chain, any address more concrete than its own.
- *Constraint 3t:* A target feature module in a request chain does not transmit upstream, as an alternative target of the chain, any address more concrete than its own.

4.3 Properties

The purpose of the constraints is to support the goals of privacy, authenticity, reversibility, boundedness, and monotonicity. Specifically, the constraints guarantee the properties stated in this section.

Constraints 2s and 2t are by far the most important ones. They guarantee boundedness and monotonicity properties, as well as contributing to the proofs of other properties.

- *Source Boundedness Theorem:* In a request chain that satisfies Constraint 2s, the number of feature modules in its source region is less than or equal to the depth of the abstraction order on address categories.
- *Target Boundedness Theorem:* In a request chain that satisfies Constraint 2t, the number of feature modules in its target region is less than or equal to the depth of the abstraction order on address categories.
- *Source Monotonicity Theorem:* In a request chain that satisfies Constraint 2s, if *m1* and *m2* are feature modules in its source region, and *m1* precedes *m2*, then the address of *m2* is more abstract than the address of *m1*.
- *Target Monotonicity Theorem:* In a request chain that satisfies Constraint 2t, if *m2* and *m1* are feature modules in its target region, and *m2* precedes *m1*, then the address of *m2* is more abstract than the address of *m1*.

These theorems are straightforward consequences of the definitions and constraints.

The principle of reversibility says that it should be possible for a target of a request chain (feature module or interface module) to target the entity on whose behalf the chain was initiated. The best identification of this entity is the most abstract source address in the chain.

- *Reversibility Theorem:* In a request chain that satisfies Constraints 1 and 2s, there is no value of a *source* field more abstract than the value of the last *source* field. If the chain has a target region, this is the value of the field throughout the target region.

Proof:
(1) If the *source* field is changed in the course of the request chain, it is changed by a source feature module (Constraint 1).
(2) If a source feature module changes the *source* field of a request chain, it changes it to a more abstract address than the previous value (Constraint 2s).
(3) Consequently, the *source* field grows monotonically more abstract throughout the source region, then stays the same throughout the target region. □

The authenticity of an address cannot be guaranteed unless its feature modules perform authenticity functions, as described in Section 3.4. The proofs of authenticity properties rely on these functions as well as on constraints.

- *Source Authenticity Theorem:* If *s2* fills a *source* field in the target region of a request chain that satisfies Constraints 1 and 2s, and if *s2* has a source

feature module with unconditional authentication, then either an owner of *s2* is present at the initiating device, or an owner of *s2* also owns another *source* address *s1* of the chain, and *s1* is more concrete than *s2*, and the source feature module of *s1* contains the authentication secret of *s2*.

- *Target Authenticity Theorem:* If *t2* fills a *target* field in the target region of a request chain that satisfies Constraint 2t and extends all the way to an interface module on the terminating side, and if *t2* has a target feature module with unconditional authentication, then either an owner of *t2* is present at the terminating device, or an owner of *t2* also owns another *target* address *t1* of the chain, and *t1* is more concrete than *t2*, and the target feature module of *t1* contains the authentication secret of *t2*.

The proofs of these theorems abstract away the temporal dimension. For example, the Target Authenticity Theorem applies to a request chain that "extends all the way to an interface module on the terminating side," even though there is a bounded interval in which the the chain has reached the terminating interface module, but no user has answered yet, or there has not been sufficient time for the authentication process to occur. The proof assumes that this interval is past.

Furthermore, both theorems say "an owner of [the authenticated address] is present at the [relevant] device," even though the owner may only have been present long enough to enter the authentication secret, and then walked away. In practice, the only way to prevent this is to make the authentication function demand re-authentication periodically, and to break the connection if it does not succeed.

Finally, an owner of *s2* or *t2* might put its authentication secret in a feature module of *s1* or *t1* for his own convenience. (Provided that the same person owns both addresses, this does not violate ownership assumptions about keeping secrets.) *s1* or *t1* might be the address of a device used by no one else. If its feature modules have the authentication secret of the more abstract address, then he can use the private device and the abstract address without bothering to enter the secret manually.

However, contrary to the owner's expectations, someone else might use this configuration of equipment and features. This is how the assumptions of the theorems might be satisfied, yet the owner of *s2* [*t2*] is not present at the initiating [terminating] device.

The authenticity theorems are proved as follows.

Proof of Source Authenticity Theorem:

(1) The request chain contains a source feature module of *s2*. This follows from:

(1a) *s2* fills the *source* field of the unique request in the chain with initial *region* equal to `srcRegn` and final *region* equal to `trgRegn` (Region Lemma, Constraint 1).

(1b) When issued by a box, this request had *status* equal to `changed` or `unchanged` (definitions).

(1c) If `unchanged`, then the issuing box is a source feature module of *s2* (definitions). If `changed`, then *s2* has no source feature set, which contradicts the assumptions of the theorem (definitions).

(2) Because the chain has been continued past the source feature module of *s2*, that module received from upstream the authentication secret of *s2* (behavior of authentication function).

(3) If there is a feature module upstream of the source feature module of *s2*, it is a source feature module of an address *s1*, where *s1* is more concrete than *s2* (Constraint 2s, Source Monotonicity Theorem).

(4) The source feature module of *s2* could only have received the authentication secret from the initiating device or from a feature module upstream, which must be a source feature module of *s1*. This results in the following two cases:

(4a) If the secret was received from the initiating device, then an owner of *s2* is present at it to enter the secret (ownership assumptions).

(4b) If the secret was received from a source feature module of *s1*, then *s1* must be owned by an owner of *s2* (ownership assumptions). □

Proof of Target Authenticity Theorem:

(1) The request chain contains a target feature module of *t2*. This follows from:

(1a) There is a request in the chain with *target* equal to *t2* and final *region* equal to `trgRegn`.

(1b) The final *status* of this request is `changed` or `unchanged` (definitions).

(1c) If `changed`, then the request is routed to a target feature module of of *t2* (definitions). If `unchanged`, then the request was issued by a target feature module of of *t2* (definitions).

(2) Because the chain has been allowed to exist after reaching the terminating device, that module received from downstream the authentication secret of *t2* (behavior of authentication function).

(3) If there is a feature module downstream of the target feature module of *t2*, it is a target feature module of an address *t1*, where *t1* is more concrete than *t2* (Constraint 2t, Target Monotonicity Theorem).

(4) The target feature module of *t2* could only have received the authentication secret from the terminating device or from a feature module downstream,

which must be a target feature module of *t1*. This results in the following two cases:
(4a) If the secret was received from the terminating device, then an owner of *t2* is present at it to enter the secret (ownership assumptions).
(4b) If the secret was received from a target feature module of *t1*, then *t1* must be owned by an owner of *t2* (ownership assumptions). □

The privacy of an address is protected by concealing it with a more abstract address and its features.

- *Source Privacy Theorem:* If *s1* fills a *source* field in a request chain that satisfies Constraints 2s and 3s, and if *s1* has a source feature module that changes the source address to *s2* in this chain, and if *s2* has a source feature module, then *s1* is not observable as a source of this chain downstream of the source feature module of *s2*.
- *Target Privacy Theorem:* If *t2* fills a *target* field in a request chain that satisfies Constraints 2t and 3t, and if *t2* has a target feature module that changes the target address to *t1* in this chain, then *t1* is not observable as a target of this chain upstream of the target feature module of *t2*.

It is very important to note that the proofs of these theorems ignore the possibility that a source address could appear in a chain for reasons unrelated to its source region, and a target address could appear in a chain for reasons unrelated to its target region. For example, the Target Privacy Theorem would be violated (nominally) by a chain in which the caller dialed *t1*, a source feature module changed *t1* to *t2*, and a target feature module changed *t2* to *t1*. The violation is only nominal because knowledge of *t1* was not leaked from the target region into the source region.

The proofs examine and eliminate all the ways that a private address could be leaked by the region that is supposed to conceal it. This is the sense in which the proofs "ignore" other possibilities.

The privacy theorems are proved as follows.
Proof of Source Privacy Theorem:
(1) All the source feature modules in the chain downstream of the source feature module of *s2* have addresses more abstract than *s2* (Constraint 2s, Source Monotonicity Theorem).
(2) An address can only be observable as a source of a request chain because it fills a *source* field of a request, or because it appears in a signal as an alternative source of the chain (definitions). This leads to the following two cases:
(2a) A request issued by the source feature module of *s2* or any source feature module downstream of it has *source* more abstract than *s2*, which is more abstract than *s1*, so it cannot be equal to *s1* (Constraint 2s).
(2b) *s1* is not transmitted downstream as an alternative source of the chain by the source feature module of *s2* or by any source feature module that follows it (Constraint 3s). □
Proof of Target Privacy Theorem:
(1) All the target feature modules in the chain upstream of the target feature module of *t2* have addresses more abstract than *t2* (Constraint 2t, Target Monotonicity Theorem).
(2) An address can only be observable as a target of a request chain because it fills a *target* field of a request, or because it appears in a signal as an alternative target of the chain (definitions). This leads to the following two cases:
(2a) *t1* cannot fill a *target* field of a request in the target region upstream of the target feature module of *t2*, because if it did, some target feature module changed a target address to a more abstract one, in violation of Constraint 2t.
(2b) *t1* is not transmitted upstream as an alternative target of the chain by the target feature module of *t2* or by any target feature module that precedes it (Constraint 3t). □

In addition to the direct proofs, some of the theorems and proof steps above have been checked for a large number of instantiations by Greg Dennis, Daniel Jackson, and Rob Seater using the Alloy Constraint Analyzer [16, 17].

4.4 Example: The anonymous correspondent

Figure 7 shows how anonymous electronic mail and an autoresponse feature can interact well within the framework of ideal address translation. The anonymous address *anon2@remailer* is a role address, used to conceal the more concrete personal address *user2@host2*. In the top half of Figure 7, the target feature module of *anon2@remailer* resolves the role address to the personal address.

The target feature module of *user2@host2* contains the autoresponse feature, set to notify all correspondents that the owner of this address is on vacation. The autoresponse initiates a new chain, which is first routed to the source feature module of *user2@host2*. The source feature module contains a list of correspondents in which *user1@host1* is marked as a correspondent with whom anonymity must be preserved. Because the target of the chain is *user1@host1*, the source feature module performs assumption, changing the source address of the chain to *anon2@remailer*.[5]

[5]The correspondent list may seem to be an *ad hoc* mech-

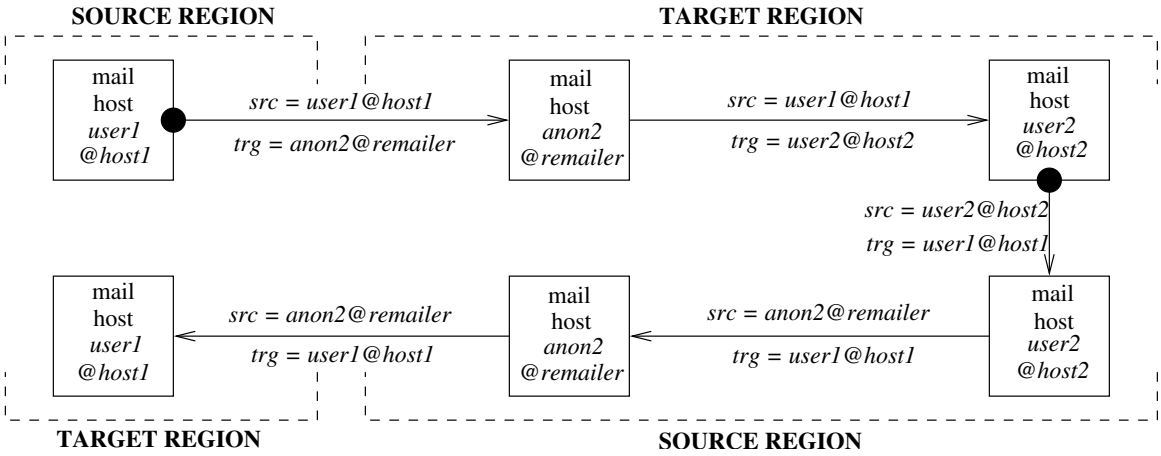

Figure 7: Anonymous electronic mail and autoresponse, in an ideal setting.

There can be a source feature module associated with *anon2@remailer*, to authenticate its use, if desired. The owner of *user2@host2* and *anon2@remailer* must encode his secret in the source modules of both addresses, so that the source module of *user2@host2* can send it automatically, and the source module of *anon2@remailer* can validate it automatically.

Because the Source Privacy Theorem applies, *user2@host2* is not observable as a source of the autoresponse chain downstream of the source feature module of *anon2@remailer*. This is what the owner of *user2@host2* cares about most.

Obviously an autoresponse feature relies completely on the reversibility of the source address it receives. Because the Reversibility Theorem applies, autoresponse works.

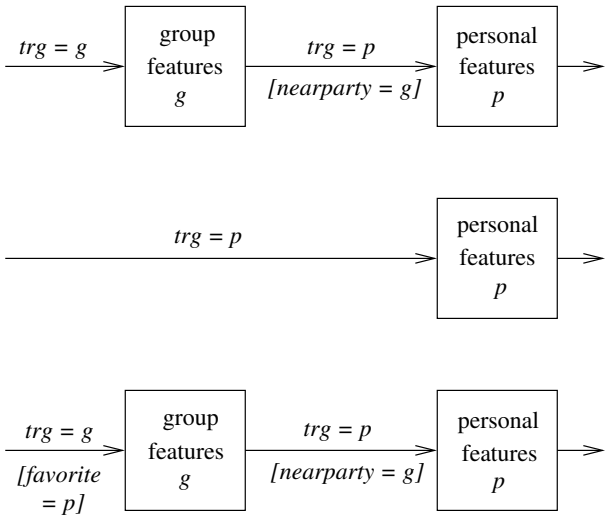

Figure 8: Three target regions containing the feature module of a sales representative.

4.5 Example: The sales representative

Figure 8 shows three ways that the voice mail features of a sales group and of the members of the sales group can be coordinated successfully. Each snapshot is the target region of a request chain.

In the top snapshot, the caller has called the group address g. The feature module of g arbitrarily selects the sales representative with personal address p, and continues the request chain to that address. Both feature modules have voice mail as a failure treatment.

As explained in Section 3.6, group failure treatments should have priority over personal failure treatments, because this is really a call to the sales group. The feature module of g seizes priority by sending downstream a signal that *nearparty* $= g$. This signal indicates that g is an alternative target address of the chain.[6] Note that this signal does not violate Constraint 3t because the alternative target is being sent downstream rather than upstream.

A *nearparty* signal has many uses, and should be

anism for solving this problem, but it is not. Some messages from *user2@host2* are anonymous, and some are not. Some messages from *user2@host2* are generated automatically, and some are generated manually. A correspondent list is the only mechanism for distinguishing anonymous messages that works for both automatically and manually generated messages.

[6]The term *nearparty* is used to indicate an alternative address from the same region, here the target region. This is in contrast to an alternative *farparty* address, which in the target region would be from the source region.

propagated downstream by any feature module that receives it. In this example it is a request to any target feature module to relinquish its voice mail feature. Another potential use is "callee identification." If the request eventually rings a shared home telephone, and if the device displays the nearparty address, it will help family members know who should answer the call.

The middle snapshot of Figure 8 simply reminds us that address p can be called directly, and that the target features of p will be unconditionally invoked.

The bottom snapshot of Figure 8 provides more complex behavior, combining the advantages of the top two. The caller's favorite sales representative is p. He calls g, also sending a signal indicating a preference for p. The feature module of g selects p as the sales representative; unlike the representation decision in the top snapshot, this decision is reproducible, so it has the advantage of the middle snapshot. If p is unavailable, however, p will relinquish treatment, and the unavailability will be treated by the feature module of g. Note that the signal *favorite* $= p$ (which comes from a module in the source region) does not violate Constraint 3s because it provides an alternative target rather than an alternative source.

Figure 9 shows how the sales representative can use his home telephone. The source feature module of the device address h includes an assumption function, allowing the sales representative to assume his personal identity p.

The source feature module of h also includes a screening function, preventing certain kinds of outgoing calls. Screening applies to continuations of the request chain in which the source is unchanged, and not to continuations in which the source is changed. Thus the sales representative can make unscreened calls, while his children cannot.

The source feature module of p includes an authentication function, which is important to prevent the children's misusing the telephone. It also has an affiliation function, so that the sales representative can make a call with source address g.

The source feature module of g also has an authentication function. After authenticating, it alters the billing so that work-related calls are billed to the company rather than the sales representative.

4.6 Modularity and extensibility

Ideal address translation is modular. A feature module can always satisfy the constraints without cooperating explicitly with other feature modules, and without even knowing which other features are present. This means that each feature module can be designed independently of other feature modules.

Because there is no requirement that the abstraction order is total, new address categories can be added to the domain without relating them to all the old categories in the domain. This is important because a real telecommunication system could have a large number of categories. In practice, most address translation works within totally ordered clusters of associated address categories; across clusters, these categories may be unordered. If so, request chains can pass from cluster to cluster only where they change from source to target region.

Ideal address translation is also extensible. Address categories can be added (or deleted) as described above. Adding (or deleting) compliant features does not require changing the existing (or remaining) ones, because their coordination is not based on explicit cooperation.

5 Reasoning about exceptions

Reasoning about exceptions is typically a kind of refinement of the general-purpose reasoning presented as part of ideal address translation. We bring in more information about the features and the context in which they are being used. Based on stronger assumptions, we prove weaker, more specialized results.

In the worst case, there is no additional reasoning, but we know which constraints are violated. From this information we can trace which desirable properties will not be preserved in the presence of the exception, and thus contain its bad effects.

The remainder of the section presents examples of common exceptions and how they can be dealt with.

5.1 Exception: Remote identification

Section 2.4 explained that, when using most real telecommunication protocols, multiple source feature modules are not available directly and must be simulated. Because the simulation might be used for any source-translation function, including assumption, positioning, or affiliation, it goes by the more generic name of *remote identification*.

Unfortunately, remote identification violates Constraint 1. This section shows how remote identification is used to implement the example of Section 4.4 on today's electronic-mail protocols. Then it considers how to reason about it despite the exception.

Figure 10 shows how to approximate Figure 7 with electronic mail as it is today. Figure 10 contains only the autoresponse chain; everything leading up to the initiation of this chain is the same as in Figure 7.

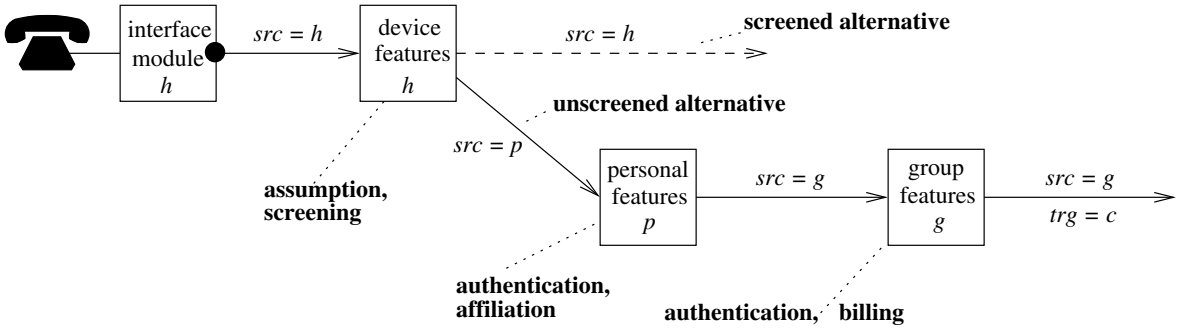

Figure 9: The sales representative makes work-related calls from a shared household telephone.

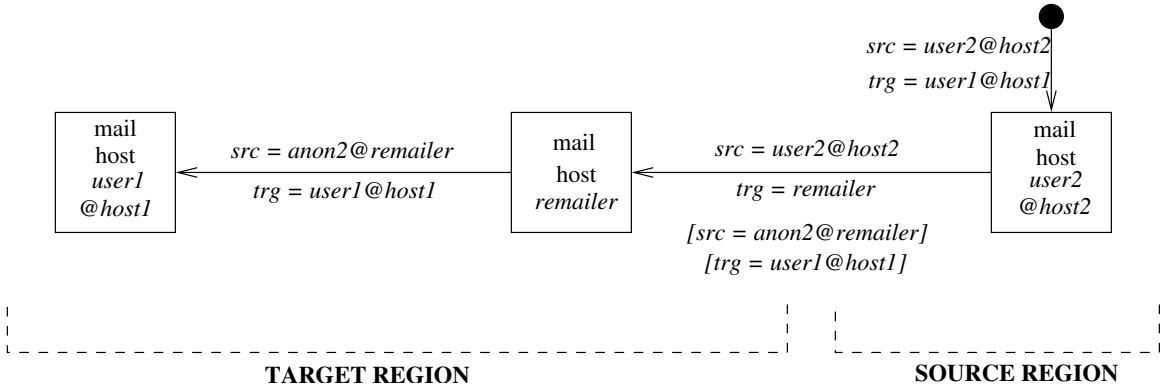

Figure 10: Anonymous electronic mail, achieved with remote identification.

To "anonymize" the request, the address book feature sends the request to *remailer*, the address of a remailing service. It also sends the truly-intended target *user1@host1*, in a signal, as an alternative target. It may also send the intended source *anon2@remailer* as an alternative source, although this may be unnecessary because *user2@host2* has an account with the remailing service. The remailer continues the chain with the intended source and target addresses.

The remailer is a target feature module that changes a source address, violating Constraint 1. Although this compromises reversibility and source authenticity in their most general forms, because the general theorems assume Constraint 1, the general theorems can be customized to fit the situation. In the following theorems, let the *remote-identification prefix* be the prefix of the chain up to and including the remailer module.

- *Remote-Identification Reversibility Theorem:* In a request chain with the remote-identification prefix, such that its suffix satisfies Constraint 1, *anon2@remailer* is the source address of the chain throughout the suffix.
- *Remote-Identification Source Authenticity Theorem:* In a request chain with the remote-identification prefix and some continuation of it, if the remailer module has unconditional source authentication, then *anon2@remailer* and *user2@host2* have a common owner.

The proofs are straightforward. Note that even if *anon2@remailer* had a source feature module it would not be present in the request chain, because once that address becomes the source of the chain, routing has already progressed to the target region. Note also that all the remailer's customers must be owners of it, in the sense of the simple ownership assumptions used here. The awkwardness of this is another disadvantage of simulating multiple source feature modules, rather than providing for them in the service protocol.

Another problem with remote identification is that the Source Privacy Theorem does not apply, because the module that conceals *user2@host2* is not a source feature module. It can be replaced by the following specialized theorem.

- *Remote-Identification Source Privacy Theorem:* In

a request chain with the remote-identification prefix, such that the remailer module does not transmit *user2@host2* downstream as an alternative source of the chain, *user2@host2* is not observable as a source of this chain downstream of the remailer module.

This theorem is also easy to prove.

5.2 Exception: Noncompliant legacy

Section 4.5 discussed a shared household telephone. A household may have been using a single device address h for a long time. Then new technology comes along, and everyone in the family acquires a personal address and personal features. Yet the device address h is still widely known and frequently called.

When a call to h fails, the target feature module of h prompts the caller to identify the family member he is calling. Then the module retargets the call to the personal address p of that family member. This violates Constraint 2t, as a personal address is more abstract than a device address.

The target feature module of p might locate that person elsewhere, or might record voice mail in a personal mailbox. Unfortunately, it will be common for the target feature module of p to locate p at (retarget to) h. In this situation it is quite difficult to avoid unbounded looping and other problems.

The only solution is to program the target feature modules of h and p to make explicit exceptions for each other. Each feature module should send a *nearparty* signal downstream, to alert other feature modules that it is present in the request chain. Then the programs make the following exceptions:

- If the target feature module of p receives *nearparty* $= h$, then it must not retarget to h.
- If the target feature module of h receives *nearparty* $= p$, then it must propagate a failure upstream rather than providing any failure treatment.

Taking these exceptions into account for the personal addresses of all family members, it is possible to prove by case analysis that no request chain has more than one target feature module of h, and no request chain has more than one personal target feature module of a family member. Thus target boundedness is preserved, but target monotonicity is not.

In the ideal setting, callers would call personal addresses rather than h, and h would have no person- or family-oriented features. As an addition or alternative, there could be a family address f with both family features and a feature enabling the caller to select a family member. The difference between f and h is that f, being a group address, is more abstract than personal addresses. Because h is a device address, it *cannot* be more abstract than personal addresses.

5.3 Exception: Delegation

Delegation is a feature function that translates a target address, such as a personal address, to another address in the same category. Delegation can be performed automatically, as a failure treatment. It can also be performed at the command of the callee, either while the call is ringing at the callee's telephone or after the callee has answered.

Delegation violates Constraint 2t, and is a very common exception to ideal address translation. For every instance of delegation, it is possible to concoct a story in which there is a more abstract address that the caller could have used to express his purpose in calling, and in which the target feature module of that address retargets to various personal addresses, so that Constraint 2t is satisfied. According to this story, delegation occurred because the functions that belong to this hypothetical target feature module were performed by people instead.

Although such stories explain how and why delegation is an exception to ideal address translation, the exception will always remain. The purposes, categories, and rules involved are too dynamic and informal to be automated on the time scale on which addresses are published and features are subscribed. People would much rather make their own decisions on the time scale on which feature data is updated and calls are handled.

So the important question is how to adapt the reasoning behind the Target Boundedness Theorem, Target Monotonicity Theorem, Target Authenticity Theorem, and Target Privacy Theorem to this less constrained situation. Specifically, instead of Constraint 2t, we have:

- *Constraint 2t':* If a target feature module in a request chain translates the target address, the new target address is more concrete than, or as concrete as, the old one.

When Constraint 2t' holds but 2t does not, the Target Monotonicity Theorem must be weakened:

- *Weak Target Monotonicity Theorem:* In a request chain that satisfies Constraint 2t', if *m2* and *m1* are feature modules in its target region, and *m2* precedes *m1*, then the address of *m2* is more abstract than, or as abstract as, the address of *m1*.

The Target Authenticity Theorem is easy to adapt. If we change the phrase "*t1* is more concrete than *t2*" to "*t1* is more concrete than, or as concrete as, *t2*" then the theorem can be proved with Constraint 2t' and weak target monotonicity.

The Target Privacy Theorem is also easily adapted. We must add to the theorem the additional assumption that *t2* is more abstract than *t1*, which should certainly be true if *t2* is being used to hide *t1*. (Requiring the particular address *t2* to be more abstract than the particular address *t1* is different from requiring strong monotonicity along the whole target region.) In Step (1) of the proof, the target feature modules preceding that of *t2* may have addresses as abstract as, or more abstract than, *t2*. Either way, the rest of the proof still holds.

The implicit assumption behind the explanation of feature coordination in Section 3.6 is that a region contains at most one feature module at each level of abstraction, so that each module is its level's unique representative. With only weak monotonicity we cannot make this assumption. Feature coordination becomes harder, which usually means that it requires more assumptions about feature behavior.

Consider, for example, delegation among personal addresses, each of which offers personal voice mail in its target feature module. How can we guarantee that each request chain stimulates at most one offer to record voice mail? One way is to stipulate that a personal target feature module in a request chain can delegate or offer voice mail, but not both. Then even if the request chain contains several personal target feature modules, only the last one can offer voice mail. All of the other personal target feature modules have delegated control of the call to another module, and control includes the right to offer voice mail.

By far the hardest problem caused by delegation is the loss of the Target Boundedness Theorem, which cannot be patched or weakened. Without Constraint 2t, boundedness proofs are difficult. The proofs must present an order on addresses that plays the same role as the abstraction order, while being more dynamic and *ad hoc*.

6 Validity and limitations

Ideal address translation is yet another illustration of the well-known saying, "There is no problem in computer science that cannot be solved with another level of indirection." When features are not interacting properly, the problem can usually be solved by introducing another layer of addressing.

The value of ideal address translation has been proven in practice. For one example, Hall's study of 10 common electronic-mail features [14] revealed 26 undesirable feature interactions. Of these 26, 12 have nothing to do with address translation (they concern encryption and crude filtering based on domain names). All the remaining 14 are predicted by the principles presented here, and would be eliminated by adherence to ideal address translation.

For another example, at AT&T we recently developed a set of features for personal mobility, multiparty control, and device augmentation [33]. Depending on personal preferences, device characteristics, and network capabilities, these features can be composed in many different ways—so many that we have used only a small fraction of the possible configurations. Yet the address categories are ordered by abstraction, and the features satisfy the constraints of ideal address translation, so we can be sure that many potential problems do not arise in any configuration.

The formal model has four important limitations which need to be removed by further work. First, the assumptions about ownership are simplistic. Second, the model represents a closed network, even though almost all real telecommunication networks are open, interoperating with other networks that may have different address spaces and different policies. The model also has a flat address space, even though many networks are hierarchical.

The third limitation is particularly interesting. This paper concerns only request chains in which all the requests are going in the same direction, from the initiating user or his proxy to the terminating user or his proxy. In practice, it is common for a connection path between two endpoints to consist of a concatenation of segments, where the segments are little request chains initiated in alternating directions.

To understand why, imagine a long-standing connection path from one endpoint device into the network, routed through several interesting feature modules. During the life of the path, due to the actions of the features, segments of the path might be replaced by segments connecting to different places, or destroyed and later restored. The initiation of such an action is not the same as the initiation of the original request chain, so it need not come from the same end of the path.

The routing algorithm of Distributed Feature Composition (DFC) [18, 19] has been augmented to accommodate such paths in an orderly way. Despite their complexity, there has been some initial success at extending the organization and results of ideal address translation to them [32].

The fourth limitation concerns addresses in signals. They must be constrained for privacy properties to hold, but it is difficult to state constraints precisely. Many signals have address fields. Addresses are transmitted for many purposes, including billing,

data update, security, and correct feature interaction. Most protocols allow *ad hoc* signals containing addresses.

As a result, Constraints 3s and 3t are somewhat vague. It will not be possible to state them more precisely until we know more about the purposes for signaling among telecommunication features. These purposes may lead to a rigorous classification of signals, upon which a better statement of Constraints 3s and 3t can be based. It is likely that there will be an enforced distinction between addresses revealed to the infrastructure and addresses revealed to user-oriented features.

Even when these limitations are removed, there will always be exceptions to ideal address translation. Given that fact, what is its justification? First and foremost, it is useful as a design guideline. It provides a global coordinating convention that makes feature interactions easy to manage, and guarantees certain kinds of extensibility.

It also leads to better features. For example, ideal address translation makes it impossible to have a target feature that blocks all calls from payphones; if some source feature changes the payphone address to a more abstract address, the payphone address is concealed from all target features.

If one were planning to implement such a blocking feature, one might be tempted to ignore the constraints of ideal address translation. But it would be better to consider them first. Ideal address translation *does* allow a target feature that blocks all payphone calls not identified as having an abstract source address. The latter feature is an improvement over the former, as it is not desirable to block calls from close friends just because they are calling from payphones.

Even when an exception is necessary, ideal address translation is useful as a reasoning guideline. Reasoning about exceptions is usually a matter of specializing the more general reasoning in Section 4.3. The principles of Section 3, which tell us something about what to prove, are in themselves new and valuable.

7 Relation to other research

7.1 On requirements engineering

True requirements are derived from user goals [22]. They concern only the environment of a proposed system, describing how the environment should behave when the system is deployed and interacting with it [34]. As noted in Section 1, agreed-upon requirements for telecommunication systems are badly needed.

The principles in Section 3 are not true requirements because they are vague and informal. The properties in Section 4.3 are precise and formal reflections of the principles, but they are not true requirements because they are all stated in terms of request chains. Request chains are internal artifacts of system architecture, and are not observable in the system's environment. Thus a telecommunication network without internal request chains could not satisfy the properties, regardless of its external behavior.

Undoubtedly, it is possible to find the true requirements that lie between the principles and properties provided in this paper. To do so, it will be necessary to identify the phenomena in the environment that determine the internal structure of request chains, and to formalize the principles in terms of those phenomena instead of in terms of request chains.

7.2 On feature interaction in telecommunication systems

Early research on feature interaction, as characterized by Velthuijsen [30], focused on detection of bad feature interactions. Formal methods for doing this use formal descriptions of base systems B, formal descriptions of features F_1 and F_2, a feature-composition operator $\bigoplus$, and correctness assertions ϕ_1 and ϕ_2. The features are correct individually if $B \bigoplus F_1 \models \phi_1$ and $B \bigoplus F_2 \models \phi_2$. There is an undesirable interaction if $(B \bigoplus F_1) \bigoplus F_2 \not\models \phi_1 \wedge \phi_2$ or $(B \bigoplus F_2) \bigoplus F_1 \not\models \phi_1 \wedge \phi_2$. Numerous formal languages have been used to describe the systems and/or properties, and numerous tools have been used to perform the analysis.

The approach to feature interaction in this paper is different in important ways:

- It recognizes the existence of good feature interactions, such as the interaction between the autoresponse feature in the target feature module of *user2@host2* and the correspondent-list feature in the source feature module of *user2@host2* (Figure 7).
- The "detection" of feature interactions is performed by applying experience rather than formal methods. The application of formal methods has proven difficult and not very productive for this purpose [30]. When experience is available, it is a far superior route to understanding what interactions might occur. Experience can be generalized to cover all features in a large class, including features that have not been specified or implemented yet.
- This work goes beyond diagnosis to cure and prevention.

More recent research on feature interaction is much more like the work here in focusing on architectures and policies for managing interactions. However, most architectures do not single out the issues raised by address translation. For example, one agent architecture [35] is based on the following separations of concerns: separating users from terminals, calls from connections, user sessions from services, and services from resource management. The "roles" in this architecture include *owner, subscriber, payer, caller,* and *callee.* The roles of *owner* and *subscriber* overlap with the concepts of address translation, but all the other concepts of this architecture are orthogonal to the concepts of address translation.

A Mitel architecture incorporates some of the concepts of address translation, but they are still in flux. In one version [1], the roles a person can play are simply the motivations for various rules in the person's agent. In another version [2], there are separate agents for devices, persons, and roles, and an address can identify a role as well as a person. The relationships among these addresses have not been described, however.

A BNR architecture [8] incorporates a limited form of ideal address translation. There are *roles* that correspond to the group addresses, mobile addresses, and role addresses of Section 3.2. These roles resolve to *sub-roles.* Although a sub-role may have some feature functions associated with it, it is also associated with a device address. Thus the BNR architecture satisfies Constraints 2s and 2t, and enjoys (as the authors note) boundedness. The limitation is that addressing has exactly two levels. It is not compositional (as the authors also note) because, for instance, a sub-role of a group address must be a device address and cannot be a mobile address.

Like ideal address translation, the Negotiating Agents approach to feature interaction [12] is motivated by the ambiguity of telecommunication features, which often makes it impossible to tell why a function is being performed, and on behalf of whom (Section 1.1). In one example motivating Negotiating Agents, a person wishes to place a call, revealing her name but not her telephone number to the callee. In another example motivating this approach, a security feature can be associated with a device or a person. If it is associated with a person, then only calls to that person invoke security, regardless of which device they are routed to. If it is associated with a device, then only calls to that device invoke security, regardless of which person they are for.

It should be clear that these examples are directly concerned with address translation. Ideal address translation achieves their desired behavior with the correct organization of addresses and the correct association of features with addresses.

The Negotiating Agents approach is more complex. It includes agents, negotiators, arbitrators, policy specifications, proposals, counter-proposals, and a hierarchy of goals. It requires a platform that implements a negotiation process based on these concepts.

7.3 On component architectures

The formal model of Section 2 is a component architecture. It has modularity in the style of a pipes-and-filters architecture [28]: a filter (feature module) interacts with others only through pipes (requests and the signaling channels they set up); a filter does not know what is at the other end of its pipes; each filter is optional and context-independent. This shows that the pipes-and-filters concept applies to interactive systems, although it was not originally thought to do so [28].

There is currently a great deal of interest in mobile agent systems for electronic commerce and other distributed applications. Griss defines a [mobile] agent system as a component system with some of the characteristics of adaptability, autonomy, collaboration, knowledgeability, mobility, and persistence [13].

Mobile agents systems have rich behavior, but there is no reason why the formal model of Section 2 could not serve as one aspect of their architectures. Griss describes [mobile] agent systems as employing varying levels of "choreography" among agents. Sometimes agent communication is semantically rich and loosely constrained, while at other times or between other agents it is semantically rigid and tightly constrained—as telecommunication protocols are.

"Active networks" are another area of research at the intersection of software architecture and distributed applications [29, 31]. The goal of this research is to introduce application-specific processing at the network level, for example at the level of IP. So far this work has concentrated on mechanisms for extending network routers and on issues such as performance and infrastructure security. It has not yet reached the service-level issues of interest here.

7.4 On the Session Initiation Protocol

The Session Initiation Protocol (SIP) is an IETF protocol for creating, modifying, and terminating multimedia sessions [25]. It has emerged as the leading protocol for voice-over-IP services.

SIP has now become large and complex—the new standard has 269 pages, and there are so many extensions being proposed that the primary creators of

SIP have written a set of guidelines for extending it [24]. Many equipment manufacturers are using SIP as the basis of their products.

With this explosion of activity, it is difficult to pin down what SIP is and is not. There are many special-interest groups using SIP in a particular way and deprecating how it is used by others. SIP is certainly rich enough to do anything anyone wants it to do. So the only thing that can be said with confidence about SIP and ideal address translation is that some of the uses that have been recommended for SIP in the literature violate the principles of ideal address translation.

For one example, SIP is designed so that servers can be stateless with respect to individual sessions [24, 26]. SIP requests carry an address stack so that SIP application servers (feature modules) along a request chain can implement an end-to-end signaling path without maintaining any state concerning it. Whenever a request is continued by a feature module (server), the module adds its own address to the stack. The reply stimulated by the request chain is routed to the originator of the chain, along the same path, by successively peeling addresses off the stack.

One problem with the address stack is that it violates Constraint 3s. The address stack can be encrypted so that the routing infrastructure sees it while user services do not, but encryption was deprecated in the original SIP standard [15], and it is not required now.

For another example, most SIP scenarios [10, 26] show SIP being used to inform two interface modules of each other's addresses so that they can communicate directly, without any intervening servers.

This has two major disadvantages. First, it has the same effect as violating Constraints 3s and 3t, so that all address privacy is undermined. Second, it makes mid-call features impossible, simply because there is no feature module in the mid-call signaling path to implement them. This is particularly serious because features active during calls are common and important. They include switching features such as Call Waiting, conferencing features such as Three-Way Calling, transfer features, special billing features, features with interactive voice-response menus, and many others.[7]

Various protocols for *mobile IP* also have the same characteristic [23]. The protocols enable an initiating host to contact a mobile host through its home agent, after which the home agent drops out and the initiating and mobile hosts communicate directly.

Needless to say, these protocols bypass the initial request chains for good reasons. In particular, this is what their designers felt was necessary for efficiency, scalability, and reliability [27]. However, there is an inherent conflict between this approach to efficiency and the requirements for telecommunication services, which include requirements for privacy and mid-call features in the network.

At AT&T we have been experimenting with a different approach to efficiency, one that does not undermine telecommunication requirements. For many years, the protocols of the PSTN have achieved network efficiency by separating signaling from media transmission. We are adapting this approach to the IP context, so that all signals follow request chains, but the high-bandwidth media transmission can take a shorter path [4].

7.5 On address spaces

The design of network address spaces is a complex engineering task, with ramifications at many levels of abstraction [23]. Ideal address translation makes it more complex, by introducing yet another set of goals.

The requirements of ideal address translation, particularly a globally unique set of addresses, and reliable global knowledge of address categories, are difficult to satisfy in a network as open as the Internet. In the near future, they will be much easier to guarantee in a protected subnetwork.

At the same time, there is definite research interest in how such assumptions might be imposed on the Internet [23]. Certainly the problems caused by network address translators (NATs) in the Internet are a motivating force. Ultimately, the design of a single address space for a network may be recognized as an over-constrained problem, one that can only be solved by separation of concerns and a layered design.

8 A final note on research methods

This work was initiated to solve a domain-specific problem. Although there was no other goal, the results have significance for requirements engineering, extensibility, and component coordination in a much wider range of applications than the original problem domain. This is evidence of the richness and potential of domain-specific research.

[7] The original SIP philosophy was to put all mid-call features in the endpoints. This is unrealistic for many reasons, including the need for persistent, shared data, the need for specialized resources, and the problems of software maintenance. Now most SIP-based feature development is for application servers.

The original problem was festooned with legacy constraints. I did not make any real progress on solving it until I allowed myself some freedom from that burden. This is evidence of the value of seeking the ideal in the real, provided that one is working with a large set of practical examples. A large set of examples keeps one grounded in reality, even without the weight of legacy constraints.

Acknowledgments

This work has benefited greatly from the contributions of my colleagues Greg Bond, Eric Cheung, Bob Hall, Michael Jackson, Hal Purdy, Chris Ramming, and Jennifer Rexford.

My special thanks to Greg Dennis, Daniel Jackson, and Rob Seater for checking the proofs. The suggestions of the reviewers improved the paper in many ways.

References

[1] Magdi Amer, Ahmed Karmouch, Tom Gray, and Serge Mankovskii. Feature-interaction resolution using fuzzy policies. In [6], pages 94-112.

[2] D. Amyot, L. Charfi, N. Gorse, T. Gray, L. Logrippo, J. Sincennes, B. Stepien, and T. Ware. Feature description and feature interaction analysis with Use Case Maps and LOTOS. In [6], pages 274-289.

[3] Russell Bennett and Jonathan Rosenberg. Integrating presence with multi-media communications. White paper, `http://www.dynamicsoft.com`.

[4] Gregory W. Bond, Eric Cheung, K. Hal Purdy, J. Christopher Ramming, and Pamela Zave An open architecture for next-generation telecommunication service. *ACM Transactions on Internet Technology,* to appear, 2004.

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

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

[7] E. Jane Cameron, Nancy D. Griffeth, Yow-Jian Lin, Margaret E. Nilson, William K. Schnure, and Hugo Velthuijsen. A feature-interaction benchmark for IN and beyond. *IEEE Communications* XXXI(3):64-69, March 1993.

[8] D. Cattrall, G. Howard, D. Jordan, and S. Buj. An interaction-avoiding call processing model. In [9], pages 85-96.

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

[10] C. Cunningham and S. Donovan. Session Initiation Protocol service examples. Internet Engineering Task Force work in progress, November 2002.

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

[12] Nancy D. Griffeth and Hugo Velthuijsen. The Negotiating Agents approach to runtime feature interaction resolution. In [5], pages 217-235.

[13] Martin L. Griss. Software agents as next generation software components. In G. T. Heineman and W. T. Councill, *Component-Based Software Engineering,* pages 641-657. Addison-Wesley, 2001.

[14] Robert J. Hall. Feature interactions in electronic mail. In [6], pages 67-82.

[15] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. IETF Network Working Group, Request for Comments 2543, 1999.

[16] Daniel Jackson. Automating first-order relational logic. In *Proceedings of the Eighth ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 130-139. ACM, 2000.

[17] Daniel Jackson, Ilya Shlyakhter, and Manu Sridharan. A micromodularity mechanism. In *Proceedings of the Ninth ACM SIGSOFT International Symposium on the Foundations of Software Engineering and the Eighth European Software Engineering Conference*, pages 62-73. ACM, 2001.

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

[19] Michael Jackson and Pamela Zave. The DFC Manual. `http://www.research.att.com/projects/dfc`, updated as needed.

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

[21] J. Klensin, editor. Simple mail transfer protocol. IETF Request for Comments 2821, 2001.

[22] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, pages 249-261. IEEE Computer Society, 2001.

[23] E. Lear and R. Droms. What's in a name: Thoughts from the NSRG. IRTF Name Space Research Group, work in progress, 2003.

[24] J. Rosenberg and H. Schulzrinne. Guidelines for authors of extensions to the Session Initiation Protocol (SIP). Internet Engineering Task Force work in progress, November 2002.

[25] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. IETF Network Working Group, Request for Comments 3261, 2002.

[26] Jonathan D. Rosenberg and Richard Shockey. The Session Initiation Protocol (SIP): A key component for Internet telephony. *Computer Telephony* VIII(6):124-139, June 2000.

[27] J. Saltzer, D. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems* II(4):277-288, November 1984.

[28] Mary Shaw and David Garlan. *Software Architecture.* Prentice-Hall, 1996.

[29] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications* XXV(1):80-86, January 1997.

[30] Hugo Velthuijsen. Issues of non-monotonicity in feature-interaction detection. In [9], pages 31-42.

[31] David Wetherall, Ulana Legedza, and John Guttag. Introducing new Internet services: Why and how. *IEEE Network Magazine*, July 1998.

[32] Pamela Zave. Ideal connection paths in DFC. AT&T Research Technical Report, November 2003.

[33] Pamela Zave, Healfdene H. Goguen, and Thomas M. Smith. Component coordination: A telecommunication case study. *Computer Networks,* to appear, 2004.

[34] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transaction on Software Engineering* XXII(7):508-528, July 1996.

[35] Israel Zibman, Carl Woolf, Peter O'Reilly, Larry Strickland, David Willis, and John Visser. Minimizing feature interactions: An architecture and processing model approach. In [9], pages 65-83.