From Architecture to Requirements: A Success Story

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

Abstract

User requirements for telecommunication systems are difficult to understand because they are obscured by a long history of ad hoc feature development and technological limitations. The presence of a viable modular architecture for telecommunication features gives us a fresh start. Working within this framework, we can discover desirable properties that ought to be requirements for all telecommunication systems.

1 Introduction

This paper tells a story. By the end of the story there is a close relationship between requirements and architecture in the telecommunication domain. The recognition of true requirements developed from the architecture, however, rather than shaping the architecture as might be expected.

Although this appears to be a success story, it is far from over, and there is much work yet to be done.

2 No telecommunication requirements

The life of the Public Switched Telephone Network (PSTN) began in 1875. Since the 1960s telephone switches have been controlled by software, which has enabled and encouraged addition of a steady stream of features of all kinds.

In this paper, "requirements" refers to user requirements, descriptions of the behavior of the system as a user observes it. Requirements concerning performance, reliability, resources, administration, *etc.*, are not discussed.

Beyond the most basic requirement of allowing people to talk to each other at a distance, the PSTN seems to have no requirements, or at least no requirements in the sense of desirable properties that are globally satisfied. Every desirable property one can think of has many exceptions. To give a simple but rich example, consider this property: If the subscriber owning address x subscribes to a feature that blocks all calls to or from address y, then the owner of x is never talking to the owner of y through the telephone network.

This property might not be satisfied because x and y are associated with devices, and at least one of their owners is using a different device. This is typical of the ambiguity we see everywhere in the PSTN—addresses identify many things, but never what we really want to identify, which is people.

Alternatively, the property might not be satisfied because of an interaction among features. For example, the owner of y might cooperate with the owner of z so that calls to z are forwarded to x. If the forwarding feature sets the source of the call to z at the same time that it sets the target of the call to x (which is the most common behavior), and if x is not blocking calls from z, then the owner of y can call z and be connected to the owner of x. In this case the behavior of the call forwarding feature subverts the intention of the blocking feature.

The property could also be violated by interaction with a large-scale conference feature. Such conferences have their own addresses; participants can join the conference by calling the conference address, or can choose ahead of time to be called by the conference. Either way, the blocking feature of x cannot prevent its owner from joining a conference in which the owner of y is also a participant.

The lack of *satisfied* requirements in the PSTN is not surprising, given its long history, incremental development, technological limitations, and geographical and administrative distribution. There is also, however, a near-complete lack of *understood and agreed upon* requirements, whether satisfied or not. Simply put, we do not know how telecommunication systems should behave.

Beyond the obstacles to requirements already mentioned, the goals of subscribers often conflict, and there is no consensus about how to balance them. In addition, many people appear to believe that a telecommunication system should behave toward each subscriber exactly as that subscriber might wish during every moment of his life, without acknowledging the impossible complexity of reaching such a goal.

There is now an industry-wide trend away from circuit-switched networks, and toward packet-switched IP networks. This change is removing many technological limitations, but it is not getting us any closer to understanding requirements for telecommunications. On the contrary, the IP community is much less aware of the issues than the PSTN community is. There is an unfortunate "Internet boom" arrogance that leads newer entrants in the telecommunications arena to believe that they have nothing to learn from the past.

3 The feature-interaction problem in telecommunications

As features are added to telecommunication software, they interact with old features, often in subtle, unpredictable, or disastrous ways. This *feature-interaction problem* makes telecommunication software extremely expensive to develop. PSTN software is not unreliable, but only because reliability is so important that switch manufacturers take heroic measures to ensure that failures are contained.

The feature-interaction problem has been recognized since the late 1970s, and there has been quite a bit of research attempting to solve it [2, 3, 4, 5, 9].

Ultimately, to manage feature interactions properly we need to understand what they are, prevent the bad ones, and enable the good ones. Unfortunately, distinguishing the bad ones from the good ones depends on having requirements for desirable global behavior, which is just what we do not have.

In the shorter term, there is plenty of value to being able to add and change telecommunication features easily, in a way that is modular and guaranteed not to break the system. This is a huge improvement over adding features by patching monolithic code, with all its attendant difficulties and dangers, even if feature interactions can still cause behavior that is undesirable to users. This short-term goal has been reached with an architectural approach.

4 An architecture for telecommunication services

Distributed Feature Composition (DFC) is a component architecture for telecommunication services [7, 8]. It was designed for feature modularity, feature composition, structured feature interaction, and generality within the telecommunication domain.

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

The fundamental concept of DFC is pipe-and-filter modularity [10]. Each feature box behaves transparently when its functions are not needed. Each feature box has the autonomy to carry out its functions when they are called for; it can place, receive, or tear down internal calls, it can generate, absorb, or propagate signals traveling on the signaling channels of the internal calls, and it can process or transmit media streams traveling on the media channels of the internal calls. A feature box interacts with other features only through its internal calls, yet does not know what is at the far ends of its internal calls. Thus each feature box is largely context-independent; feature boxes can easily be added, deleted, and changed.

In DFC there are exactly two mechanisms for feature interaction (or *component coordination,* to use a more architectural term). One is the signaling through internal calls, which is governed by the DFC protocol. The other is the DFC routing algorithm, which routes each internal call to a box, thus determining the configuration of boxes in each usage as it grows, shrinks, and reshapes itself. Feature boxes can influence the routing in specific ways, which is how routing becomes a mechanism for feature interaction.

DFC has been notably successful at reaching its goals. It is not easy to say how one could reproduce the success in another application domain, but here are a few observations that seem relevant:

- Michael Jackson and I began work on DFC at the beginning of 1997. At that time one or both of us had been studying the telecommunication domain, on and off, since 1982. Trying to tame the complexity of feature interactions, we had run up seemingly every possible blind alley.
- We were completely content to be domain-specific; we had no interest in any other domain, believing that telecommunications was more than enough challenge for us. More general applications of the ideas in DFC are just now beginning to emerge.
- In important ways, DFC is low-level: it is close to the true building blocks of telecommunication implementations. This accounts for its generality.
- At the same time, DFC is abstract enough to be formally defined in a few pages. This makes the application of formal methods to DFC tractable.

Since 1997 we have made a number of changes to the original DFC architecture [7]; these are documented in the manual [8]. Some changes are refinements, while others extend DFC to cover aspects of telecommunications not originally considered, for example multimedia. Nevertheless, the central ideas of the original architecture are still present and essentially unchanged.

5 Experience with the architecture

Since 1999 we have been implementing and exploiting DFC within AT&T Research. Our BoxOS system [1] is an IP implementation of DFC with excellent interoperation capabilities; for example, it can be packaged as a SIP application server.

We have used this environment to create interesting voice-over-IP services. In 2002 alone we implemented features for personal mobility, mid-call movement from one device to another, switching, small-scale conferencing, transfering, augmenting a telephone with a graphical user interface on a nearby personal computer, call logging, voice mail, speed dialing, click-to-dial, voice signaling, reaching a representative of a group, and large prescheduled conferences.

This rapid feature development creates relentless pressure to understand feature interactions better. When a feature developer is faced with a seemingly arbitrary choice of feature behavior, he wants to know the consequences of each choice. Which choice will cause the feature to interact best with all other features, present and future?

Although the pressure to understand feature interactions returned us to the seemingly hopeless problem of discovering the requirements for telecommunications, we returned to it with some additional weapons in the arsenal. The improvement was due to the presence of a viable architecture. Because of the architecture, we could implement features quickly and plan ambitiously; this gave us a broader base of knowledge about features, how they can interact, and what people are trying to use them for. Also because of the architecture, we had a tractable formal framework in which to reason, without loss of generality, about features and their interactions.

6 Example: call forwarding

As an example of the subtleties of telecommunication behavior, let's return to the example of call forwarding as introduced in Section 2. When the owner of y calls z and the features of z forward the call to x by changing its target address to x, should the source address of the call also be changed to z?

Most forwarding features make the source change, both in telephony and in electronic mail [6]. If some error occurs in attempting to reach the new target address x, then the error should be reported to the features or owner of z, which know about x. If the error is reported to the features or owner of y, the result may be confusion or a violation of privacy, since y may know nothing about x. There is also a vague concern that if the source address is not changed and no trace is left of the role of z, there might be security problems.

On the other hand, changing the source address during forwarding has negative consequences. The source address is no longer a reliable indication of who the callee will be talking to when he answers the call, which is why the blocking feature is undermined. Also, x may have features that automatically return a received call, by placing a new call to its source address, under various circumstances. If one of these features is activated while z is still forwarding its incoming calls to x, then a forwarding loop will be created.

It might seem attractive to solve this problem by maintaining a complete address history within the signals of the call protocol, rather than just two addresses. Unfortunately, this also has many negative consequences.

Because address histories can grow quite long, they place a heavy burden on the infrastructure. In fact, the voice-over-IP protocol SIP, which maintains an address history for reasons other than the ones discussed here, is causing implementation problems due to very long headers. Address histories do not interoperate well with the existing telecommunication infrastructure, all of which is based on calls with two addresses.

Equally important, address histories can violate privacy. Consider a physician calling patients from home. He has a feature that allows him to change the source address of his calls to his office address. A complete address history would reveal to patients the address of his home telephone, which is the original source of each call. Yet the physician has a legitimate right to keep this information private.

7 Ideal address translation

It should be clear by now that if we accept the telecommunication domain as it exists today, the call-forwarding problem has no solution. Any choice we make about its behavior violates some desirable property that should be a global requirement.

One way out of this dilemma is to concentrate on an ideal version of telecommunications in which there are no legacy constraints, and both the infrastructure and the features behave in the right way. This gives us the freedom necessary to figure out what the right way might be.

Address translation is the function performed by a feature when it changes the source or target address of a call. Call forwarding performs address translation, as do many (perhaps even most) other features. For the feature interactions caused by address translation, a search for the ideal has succeeded, yielding two important and highly intertwined results [11]:

- Requirements that a telecommunication system should satisfy.
- Constraints on the infrastructure and on feature behavior that guarantee satisfaction of the requirements without sacrificing functionality.

The constraints on the infrastructure are architectural, and are inspired by DFC.

It is outside the scope of this paper to present the principles of ideal address translation. As a substitute, here is a brief, informal explanation of how the conflicts of the previous section can be resolved.

In the recommended infrastructure, a call is implemented by a chain of requests, feature modules, and interface modules as shown in Figure 1. Each request has source and target addresses. The chain has a *source region* in which there are (optional) feature modules associated with source addresses. For example, *s1* might be the address of the physician's home telephone, and *s2* might be the physician's office address. The source feature module of *s1* changes the source address of the call to *s2* at the physician's request.

The source region is followed by a *target region* in which there are (optional) feature modules associated with target addresses. For example, *t2* might be z, and the target feature module of *t2* might do call forwarding by changing the target address from *t2* to *t1*.

There is an *authenticity* requirement that the source address of a request chain should reveal to the callee the entity at the other end of the call. Call forwarding is not changing the source of the call in any way, and therefore must not change the source address. It if does, the authenticity requirement will certainly be violated.

This constraint on the behavior of call forwarding and other target-region features is necessary but not sufficient for authenticity. For example, an unauthorized person might use the physician's home telephone, or might even program the features of his own telephone to set the source address of the call to *s2*!

The authenticity of *s2* as a source address can only be secured if the source feature module of *s2* contains an authentication feature that demands a password or other proof of identity. The infrastructure guarantees that any request chain containing *s2* as a source address must pass through this module and therefore be subject to authentication.

There is a *reversibility* requirement that a target-region feature or targeted user should be able to call the source address of a request chain and thereby target the entity at the source of the request chain. Clearly this is another reason why call forwarding must not change the source address.

In this example, the source address *s2* that reaches the target region identifies the physician in his role as a physician. It is more abstract than *s1*, which is only the address of a particular device. This is why the reversibility requirement is stated in terms of "the entity at the source of the request chain" rather than "the device at the source of the request chain."

In the formal definition of the reversibility requirement, an abstract address such as *s2* is considered to identify a truer source of the request chain than a concrete address such as *s1*. This has important consequences. If a patient misses the physician's call and calls back later, the physician may no longer be at home, and the *s1* address would not reach him. However, his role address *s2* can subscribe to a location feature that will locate him wherever he is now.

There is a *privacy* requirement that use of a more abstract address such as *s2* effectively conceals a more concrete address such as *s1*. This requirement is also guaranteed by the constraints in [11].

Note that privacy and authenticity balance the conflicting goals of knowing and concealing. The effect of privacy is a person can conceal an address that he owns behind another address that he owns. The effect of authenticity is that an address can only be used by a person who owns it.

Errors are signaled back through the request chain. So if target address *t1* turns out to be unknown, then the error signal will first reach the target feature module of *t2*, which should handle the error if possible, and conceal *t1* if necessary.

8 Future work

There are many other areas of feature behavior and feature interaction besides address translation. It is important to attack them with the same weapons, in the hopes that they, also, will yield their secrets.

The infrastructure that supports ideal address translation is generally similar to all telecommunication protocols in use today. At the same time, it is different in crucial ways from all of them except DFC as implemented in BoxOS. So a gap has been opened between theory and practice that must be bridged in some way. This will require the utmost creativity, pragmatism, and patience.

Even though current telecommunication systems fall short of satisfying them, the requirements discovered so far are simple, compelling, and convincing. They would have been discovered long ago, except for the complications of a long history that has made them as difficult to see as to satisfy.

9 Acknowledgments

No success would have been possible without the many contributions of my colleagues Greg Bond, Eric Cheung, Healfdene Goguen, Karrie Hanson, Don Henderson, Michael Jackson, Gerald Karam, Hal Purdy, Chris Ramming, and Tom Smith.

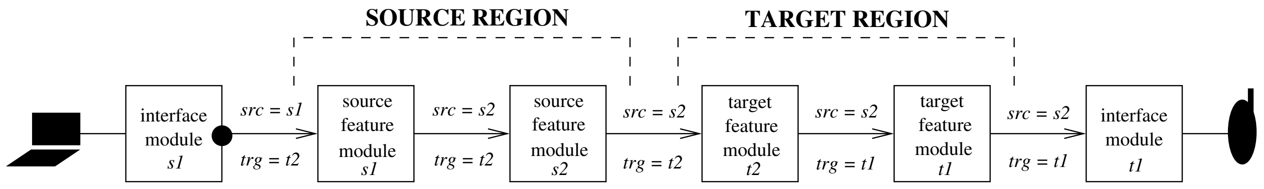

Figure 1. A request chain.

References

[1] Gregory W. Bond, Eric Cheung, K. Hal Purdy, J. Christopher Ramming, and Pamela Zave. An open architecture for next-generation telecommunication service. Submitted for publication, 2002.

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

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

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

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

[6] Robert J. Hall. Feature interactions in electronic mail. In [3], pages 67-82.

[7] 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.

[8] Michael Jackson and Pamela Zave. The DFC Manual. AT&T Research Technical Report, August 2001. Available at `http://www.research.att.com/info/pamela`.

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

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

[11] Pamela Zave. Ideal address translation: Principles, properties, and applications. In *Feature Interactions in Telecommunications and Software Systems VII,* IOS Press, Amsterdam, 2003, to appear.