Reusable Features for VoIP Service Realization

Thomas M. Smith
AT&T Labs Research
180 Park Avenue
Florham Park, NJ 07932
USA
tsmith@research.att.com

ABSTRACT

Telecommunication services vary greatly in their behavior. However they often can be decomposed into tightly-focused components, each designed to accomplish a certain limited function. In some cases, these functions are repeated across many services that seem quite disparate at first glance. We examine some components that have proven to be highly reusable, and demonstrate how they can be composed into a variety of interesting services.

Keywords

Telecommunications, VoIP, features, application composition, patterns

1. INTRODUCTION

A wide variety of telecommunication services provide familiar behavior to users everywhere. Common examples include voicemail systems, conference calling services, IVR "phone-tree" applications commonly employed by businesses, and end-user features such as call waiting and three-way calling. Some services of limited scope may be provided by a standalone software implementation of modest complexity. However, when the number of supported features increases, they may be implemented in highly complex, even byzantine, software systems. Such systems complicate the task of maintaining and extending the service logic.

It has long been considered useful to consider models in which system behavior can be decomposed into standalone modules that can be independently specified, developed, and tested. Such modules can then be combined to provide more functionality without undue complexity [8, 9, 14, 17, 10, 21]. Such a design ethic is evident in currently-emerging standards such as the IP Multimedia Subsystem (IMS) [19] and the SIP Servlet 1.1 specification [11].

This paper draws on experience developing Voice over IP (VoIP) software modules to distill a number of highly useful software components, each performing a limited function. In Section 2, the function of each component is described and illustrated by example. In Section 3, the components are employed to compose some familiar services. Section 4 contains some preliminary thoughts on different software mechanisms that can be used as compositional patterns, and the appropriateness of each in different settings. We conclude in Section 5 with a description of future work.

2. FEATURES

In this section, we will describe a number of telecommunication modules that represent common functionality that can be found in a number of different contexts. In traditional telecommunication parlance, these are designated as *features.* More generally, these features embody common *design patterns* of telecommunication logic. Software design patterns have been studied extensively [7, 16, 12], and there is a body of work concerning the use of design patterns in telecommunications [18, 20]. The features described here exhibit patterns of call-control logic. Note that although these features can be characterized as patterns, they have also been reified as functional, standalone software components.

Note that the increment of functionality embodied in these features is small; the scope of an individual feature is on the order of "call waiting," not "IP-PBX." A system with a scope of the latter could be built out of a large number of features with the scope of the former, however. This set of features is certainly not meant to be comprehensive, but rather illustrative of how small components can be composed to realize a variety of outcomes.

The features described here can be viewed through the lens of *scope, commonality, and variability* [6], though these decompositions are the result of informal iteration rather than systematic procedure. Encounters with repeated call-control patterns have suggested the scope of each feature. The commonality between instances of the features is generally dictated by the call processing message classes being acted upon (either sent or received); and the variability usually applies to parameters used to form the details of the message contents, rather than the message classes. Generally these parameters take the form of addresses, timeout values, etc.

Each of the features is described in isolation below. Then we will consider how to compose these features into useful services using *pipe-and-filter composition* [2, 10, 13]. This style of composition allows all components (features) to remain independent of the others; in fact the components may not even be aware of the existence of other components. The graph that results from the runtime composition of multiple features may be achieved by direct addressing, in which

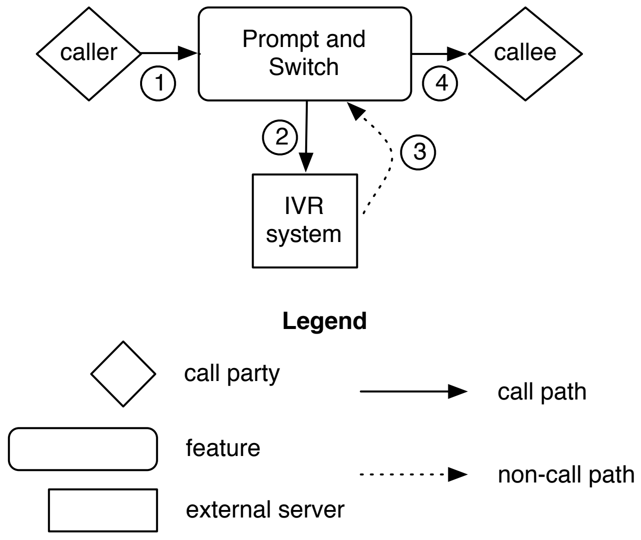

Figure 1: Prompt-and-Switch feature

features translate addresses in order to route calls to other features, or through a more sophisticated subscription mechanism, as in Distributed Feature Composition (DFC) [10]. In either case, the mechanics of composition are achieved through telecommunication protocol means, with no changes required to feature logic. Call-time composition is also used in IMS [19] and the SIP Servlet 1.1 specification [11]. Note than anywhere that a *call party* appears in the representation of a feature call flow, that party may be another feature, not necessarily an end user. From the vantage point of a feature, it does not know or care what entity is placing or receiving a call.

2.1 Prompt and Switch

The *Prompt-and-Switch* (PS) feature, shown in Figure 1, provides the ability to answer an incoming call with an automated audio dialog as commonly found in Interactive Voice Response (IVR) systems. After a period of time during which the calling party interacts with the automated dialog, the logic of the IVR system can specify an address to which the calling party should be switched. This function should be very familiar to anyone who has called the main number of a business, only to encounter an automated menu listing the parties (or departments) that can be reached via an interaction with the menu. This feature takes care of the required signaling to redirect the call to the IVR system, as well as any further signaling required to tear down the IVR call and switch the (connected) caller to a new (unconnected) address.

As presented, this feature could perform a number of familiar standalone functions:

- It can provide an "automated attendant" function, prompting the caller with a menu of departments that can be reached.

- It can be the basis of a prepaid calling service, where calls are placed to an IVR system to collect a prepaid account number and desired number to call. When the called party hangs up, the caller is reconnected to the IVR system and allowed to make more calls.

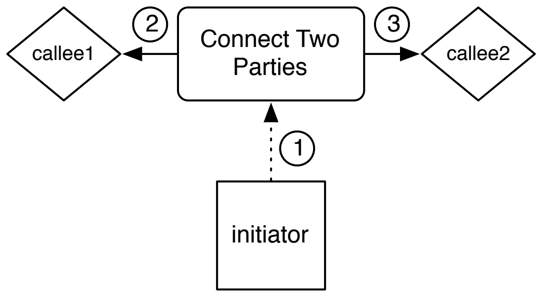

Figure 2: Connect-Two-Parties feature

- It can support "remote authentication," where subscribers call in and authenticate themselves before being allowed access to some protected functions (including placing an outbound call). Such a function is present in AT&T's Callvantage Service [4].

Figure 1 shows a graphical representation of this feature. The incoming call, labeled (1), is directed to the IVR system (2). The IVR system returns a command (3), causing the feature to switch the caller to an address specified by the IVR logic (4).

In terms of [6], the *commonality* is as described above. One element of *variability* in the call-processing logic is that the feature may be optionally configured to detect call termination from the callee and to reconnect the caller to the IVR system for further instructions. This behavior could be used to allow a caller to make multiple sequential calls after entering a prepaid calling card number or performing remote authentication.

2.2 Connect Two Parties

While a great many features are activated by an incoming call, there are occasions where the initial event is something other than a call-related event. An example of this would be a corporate web page with a button that reads, "Click here to be connected to one of our representatives." In this case, the initiating event would be a web click, not a call event. This function can be implemented in a *Connect-Two-Parties* (CTP) feature that is activated by the non-call event, as shown in Figure 2.

The initiating event must specify the addresses of the two parties that should be connected. To reduce confusion, it is desirable that the parties be connected one-at-a-time; that is, the second party will not be called until the first party answers. Accordingly, execution should terminate if the first leg of the call fails.

Figure 2 shows a graphical representation of this feature. A non-call event (1) prompts the CTP feature to place outgoing calls to first one party (2), then the other party (3). These parties are located at addresses that are specified in the initiating event.

Note that the endpoints of this call need not both be live people; rather they can be any callable entity, including media servers and other features.

2.3 Redirect on Failure

The *Redirect-on-Failure* (RF) feature provides a simple but crucial capability: it continues an incoming call toward its destination, but in the event that the call cannot reach its destination for any of a variety of reasons (busy, no answer,

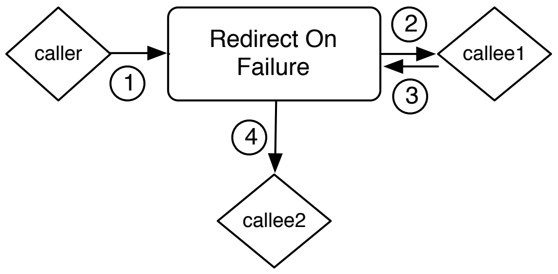

Figure 3: Redirect-On-Failure feature

network error), this feature does not propagate the failure back to the calling party. Instead, it continues the call to a different address, which may be explicitly specified through provisioning, or may be algorithmically determined based on the addresses of the parties in the call. The basic functionality may be enhanced by the optional ability (variability) of only redirecting for certain classes of failure responses, and otherwise propagating the failure back to the calling party.

The canonical use of this feature is to redirect a failed call to a resource that will record a voicemail message for later retrieval by the initial called party. Another use is to provide a "Safe Forwarding Number" in case of failure from an endpoint [4].

Figure 3 shows a graphical representation of this feature. An incoming call (1) is continued toward its destination (2). A failure response from callee1 (3) causes the RF feature to continue the original call to callee2 instead (4).

2.4 No Answer Timeout

The ability to detect call timeouts is important in a variety of contexts. A familiar example is the redirection of an incoming call to a voicemail server after a specified "Ring No Answer" timeout. However the ability to detect and act upon such timeout conditions is more general and can be used in other contexts, as we will see below. Therefore a simple *No-Answer-Timeout* (NATO) feature which detects such a condition and can signal its occurrence to other parties can be valuable. A failure response may be the mechanism it uses to signal the outcome to other components.

No graphical representation should be necessary for the understanding of this feature. It works with a calling party and a called party, and behaves transparently unless the no-answer condition is satisfied, in which case it ends the call to the called party and signals the calling party of the condition.

3. SERVICES

This section discusses *services*, by which we mean a grouping of *features* used to accomplish a task of interest. Informally, a service represents a unit of functionality that could be marketed and sold. The functionality of each example service will be described, and a possible implementation using pipe-and-filter composition of the preceding features will be presented.

These services can be viewed as products in a *software product line* [15]. In this context, the set of features presented in Section 2 comprise the *platform*; customization for individual products occurs through the selection and relative arrangement of the appropriate features from the platform as well as parameter-based specialization of those features.

3.1 Conference Calling Service

A typical business-oriented *Conference Calling Service* operates as follows. A user calls a well-known address (the *bridge number*) and interacts with an IVR system to provide details about the conference that the user wishes to join. This information often takes the form of a personal identification number (PIN) for the desired conference. Upon successful entry of this information, the user is switched into a conference call, in which the media streams from all users are mixed.

The core of such a service can be realized through the use of a PS feature for initial PIN processing in conjunction with a media server to provide the media mixing after successful completion of the IVR dialog. In addition, a capability could be added to enable users to connect to the conference via clicking on a web link instead of dialing the phone. This can be enabled through the introduction of a CTP feature to call out to the user and then connect the user to the IVR system.

Figure 4 shows an architectural diagram of the service, composed of the individual features.

3.2 Voicemail/Do Not Disturb Service

A *Voicemail* service is used to capture incoming calls when the intended recipient is not available. The two common cases that result in voicemail treatment for a call are when the called party is *busy* or when there is *no answer*. Either of those conditions, as well as various network failure conditions, can be viewed a failure, so the use of the RF feature is natural. The NATO feature can be employed to generate a failure response when the no-answer condition is detected.

Do Not Disturb treatment is intended to reduce or eliminate unwanted incoming calls to a user. The simplest implementation rejects all calls while active. A less draconian form may use a *blacklist* (or *whitelist*) approach, where a list of addresses is deemed as undesirable (or desirable), and calls from those addresses are consequently rejected (or accepted), while all others are accepted (or rejected). Rejected calls may be redirected to a voicemail system.

A more sophisticated implementation may include a notion of a *graylist*, indicating that the caller is required to interact with an IVR system before the decision is made as to whether or not the call can "ring through." It is this version of the service that we describe in this section.

Figure 5 shows an architectural diagram of the service, composed of the individual features. The presence of an IVR system with conditional switching based on the outcome of the IVR dialog suggests the presence of a PS feature. Note that the NATO feature is not adjacent to the RF feature, as it might be for a pure voicemail service. In this case, the NATO feature should not be invoked until the IVR system has determined that the caller is authorized to ring through. This dictates the placement of NATO between the PS feature and the callee. Now three different conditions can cause the RF feature to send the caller to voicemail:

- Call rejection from Do Not Disturb logic
- Failure response from callee (e.g., busy condition)
- No answer from callee

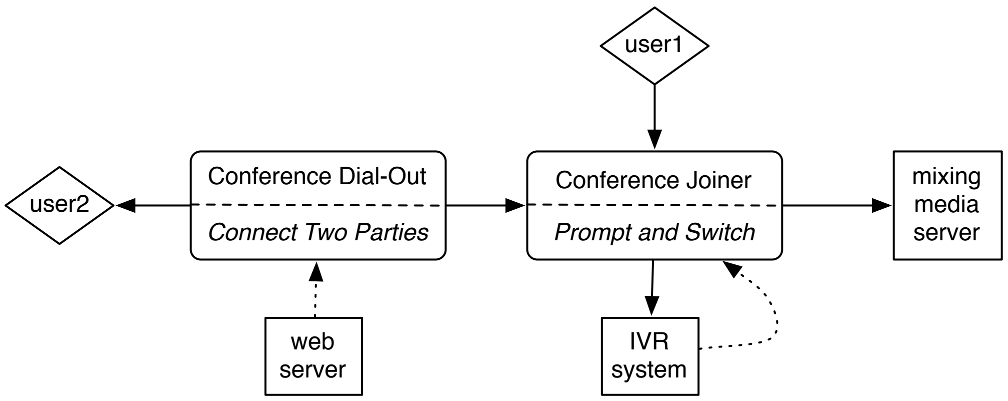

Figure 4: Conference Calling Service Architecture

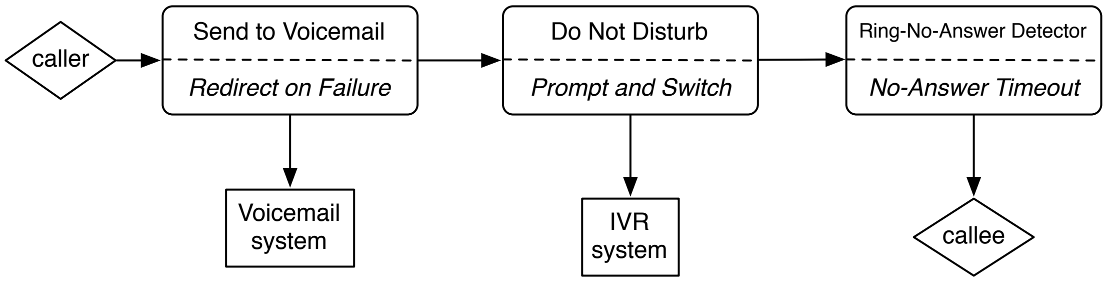

Figure 5: Voicemail/Do Not Disturb Service Architecture

The RF feature, which sends the caller to the voicemail system, is not concerned with which of these conditions has occurred. Its role is the same, regardless of what has happened in the rest of the call path. This is the essence of modularity.

3.3 Record and Send

The *Record and Send* service permits a subscriber to record a message which will then be automatically delivered to a list of addresses via an outbound call. Such a service can be used for event reminders (*"Don't forget to vote in today's school elections"*) as well as timely notifications (*"Today's lacrosse game is cancelled due to poor field conditions"*). The subscriber could possibly set up the service over an IVR session, but the complexity of managing a list of addresses to notify lends itself to a visual medium, such as a web interface. This variation is described in this section.

To use the service, the subscriber logs into a service website. A list of addresses may be maintained between sessions, so that certain addresses may be selected from a list of previously-used addresses (or from an address book); new addresses can be added as required. Once the list is complete, the subscriber indicates whether the notifications should be made right away, or at a scheduled time. Finally, the message must be recorded. When the subscriber chooses to record, the web server can send a request to a CTP feature to connect the subscriber and an IVR dialog which will prompt the user to record the notification message. This is in case the link is clicked in error when the subscriber is not near the telephone, and this prevents the device ringing again and again, when under normal circumstances the subscriber should be able to answer quickly. This can be provided via the NATO feature on the leg of the call going to the subscriber.

Once the outgoing message has been recorded, the service logic can place calls to the specified list of addresses, once again via a CTP feature. Each instance of the CTP feature will connect one notification address with an IVR dialog in order to play out the message. The service could do these all in parallel, or could stagger them in time in order to meet resource constraints. Such a decision has no affect on the logic of the features. A Ring Stopper can be employed if desired on the outgoing calls.

Figure 6 shows an architectural diagram of the service, composed of the individual features.

Note that the initial Connect-To-Recorder function can be reused in other contexts. Voicemail systems typically allow a subscriber to record an *outgoing message* that will be played to callers when the subscriber is unavailable. Sometimes there are multiple outgoing messages, each to be played when certain conditions are met (subscriber does not answer, subscriber is on the phone, etc.). The user interface employed to record these messages is typically an IVR system which is used to manage all aspects of the voicemail service, including message retrieval and management of settings such as the number of rings allowed before redirection. As such, the functions used to record outgoing messages are typically located within a fairly complex IVR dialog.

There is a trend, particularly for VoIP systems, for providing web-based interfaces to voicemail systems in addition to the traditional IVR interface. These web interfaces typically allow access to recorded messages, as well as to configuration options. By converting the serial presentation of an IVR dialog to a visual presentation, usability can be greatly

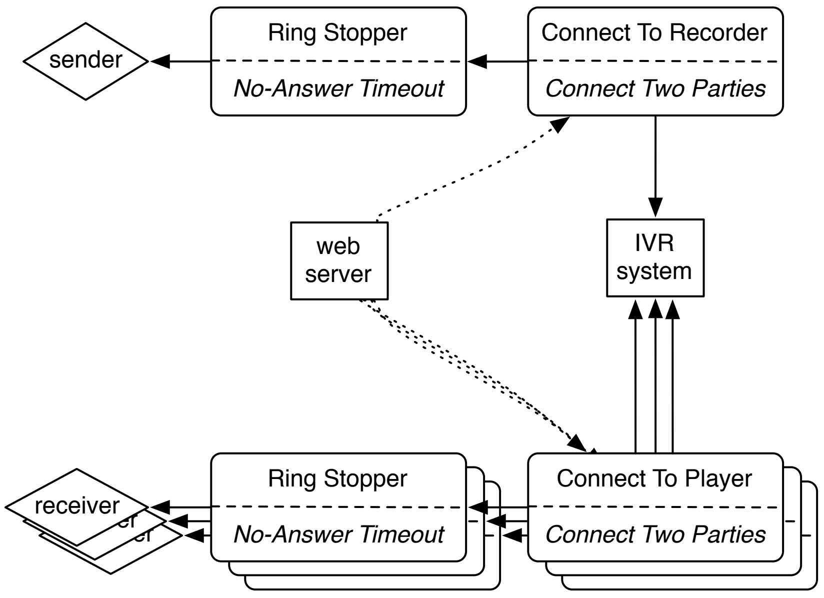

Figure 6: Record and Send Architecture

enhanced. Some systems, such as AT&T's CallVantage Service, allow the user to click a link on this web interface in order to record an outgoing message. When the link is clicked, the subscriber's phone will ring; when the call is answered, the subscriber is placed precisely at the point in the IVR dialog where the greeting is recorded. This allows the use of a web browser for viewing and choosing configuration options, coupled with the use of the telephone as a ubiquitous audio input device.

Since the IVR component is presumed to exist already, the web-based greeting recording function can be added using the same Connect-To-Recorder composition shown in Figure 6.

4. OTHER COMPOSITIONAL PATTERNS

The previous section illustrated the creation of service logic through *pipe-and-filter* composition. This design pattern is well known in software engineering, and can be seen in Unix process pipelines, data processing, and web development frameworks [2, 1]. The Distributed Feature Composition (DFC) architecture [10] employs the pipe-and-filter architecture to great advantage in controlling feature interaction. By proper use of address translation [22], a variety of DFC *usages*, each containing an appropriate feature chain, can be assembled. DFC provides a theoretical basis that can be used for realizing composed services, like those in the previous section. By having a toolbox full of general features, services can be easily composed by manipulating addresses and feature subscription, instead of writing new code.

However, pipes and filters are not the only mechanism for software re-use. Perhaps the most common software re-use mechanism is that of a *software library* made available to applications via a documented API. Software libraries are routinely used during almost any software development process. Some of the functions defined in the preceding sections could also be implemented as library modules.

There are differences in the patterns of re-use between composing complete feature modules, as discussed in Section 3, and using software libraries. One key difference is that feature composition occurs at runtime, and library composition occurs at compile time.

There are tradeoffs associated with these two patterns of re-use. For example, use of a software library may well have less overhead associated with a call to a library routine than with the instantiation of a feature instance and the overhead of having more call protocol instances to manage. Compile-time checking could also ensure that the function is being invoked from an appropriate context. On the other hand, run-time assembly of the software components means that no recompilation is required to change behavior. This means that the software composition can be controlled purely through configuration of the desired call graph.

In practice, it may not be clear which of these compositional patterns is more suitable for a particular implementation. Some experiences indicate that pipe-and-filter composition may be most appropriate when designing a new service for the first time, as this level of composition can be achieved through configuration rather than code changes [5]. When an implementation is more mature and stable, it may be desirable to migrate certain functions to library routines. This can potentially improve performance characteristics of the system as a whole, presuming that calling a library routine is less computationally expensive than performing the incremental call processing that would be required for a standalone feature. A good rule of thumb may be that if a component needs to create or absorb call-path messages in order to perform its function, it should be realized in a feature, not a library routine. There are at least two justifications for this design rule. First, if one component is linked to another at compile time, it prevents run-time interpolation of a third component between the other two and thus constrains flexibility for re-use. Also, inter-feature messages are a critical mechanism for analyzing and controlling feature interaction.

5. FUTURE WORK

Extensive experience building VoIP services has resulted in the extraction of common call-control patterns into the features described in this paper. There is every reason to expect that further experience will result in continuing insights into interesting decompositions. Investigation of systematic techniques such as [6] could yield new insights. The applicability of software product line engineering techniques [15, 3] to this domain should be explored.

Additionally, the thoughts on alternate compositional patterns are in the early stages. Other compositional patterns from the software engineering literature [2] should be explored for possible applicability to this domain. Further reflection and experience should lead to more concrete and rigorous design guidelines.

Finally, the topic of *converged services* has not been directly addressed in this paper. When non-telecommunication protocols (such as HTTP) are included as part of the service logic, the boundary lines of features, and thus compositional patterns, may shift.

6. CONCLUSION

By carefully designing telecommunication features to perform certain common functions, it is possible to realize complex service logic through call-time composition of the constituent features. There is growing industry momentum for such mechanisms as seen in the IMS architecture and the SIP Servlet 1.1 standard.

7. ACKNOWLEDGMENTS

The author gratefully acknowledges his close extended collaboration with Greg Bond, Eric Cheung, Karrie Hanson, Don Henderson, Gerald Karam, Hal Purdy, Venkita Subramonian and Pamela Zave, out of which this work was produced. Alicia Abella consistently provides excellent advice. Laura Dillon also provided valuable suggestions which improved this paper. Finally, the anonymous reviewers provided excellent suggestions about related work.

8. REFERENCES

[1] Apache cocoon site. `http://cocoon.apache.org/`.

[2] V. Ambriola and G. Tortora. *Advances in Software Engineering and Knowledge Engineering*. World Scientific, 1993.

[3] F. Bachmann and L. Bass. Managing variability in software architectures. *SIGSOFT Softw. Eng. Notes*, 26(3):126–132, 2001.

[4] G. W. Bond, E. Cheung, H. H. Goguen, K. J. Hanson, D. Henderson, G. M. Karam, K. H. Purdy, T. M. Smith, and P. Zave. Experience with component-based development of a telecommunication service. In *Proceedings of the Eighth International Symposium on Component-Based Software Engineering*, pages 298–305. Springer-Verlag LNCS 3489, May 2005.

[5] E. Cheung and T. M. Smith. Experience with modularity in an advanced teleconferencing service deployment. In *ICSE Companion*, pages 39–49. IEEE, 2009.

[6] J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. *IEEE Softw.*, 15(6):37–45, 1998.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[8] J. J. Garrahan, P. A. Russo, K. Kitami, and R. Kung. Intelligent Network overview. *IEEE Communications*, 31(3):30–36, March 1993.

[9] N. D. Griffeth and H. Velthuijsen. The Negotiating Agents approach to runtime feature interaction resolution. In L. G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunications Systems*, pages 217–235. IOS Press, Amsterdam, 1994.

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

[11] *JSR 289: SIP Servlet Version 1.1*. Java Community Process, 2008. Available from: `http://jcp.org/en/jsr/detail?id=289`.

[12] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[13] D. Mennie and B. Pagurek. An architecture to support dynamic composition of service components, 2000.

[14] M. Plath and M. Ryan. Plug-and-play features. In K. Kimbler and L. G. Bouma, editors, *Feature Interactions in Telecommunications and Software Systems V*, pages 150–164. IOS Press, Amsterdam, 1998.

[15] K. Pohl, G. Böckle, and F. J. v. d. Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[16] W. Pree. *Design patterns for object-oriented software development*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[17] C. Prehofer. Plug-and-play composition of features and feature interactions with Statechart diagrams. In D. Amyot and L. Logrippo, editors, *Feature Interactions in Telecommunications and Software Systems VII*, pages 43–58. IOS Press, Amsterdam, 2003.

[18] L. Rising, editor. *Design patterns in communications software*. Cambridge University Press, New York, NY, USA, 2001.

[19] F. Salm. Application servers and sip signaling in ims environments.

[20] D. C. Schmidt. Using design patterns to develop reusable object-oriented communication software. *Commun. ACM*, 38(10):65–74, 1995.

[21] R. Steenfeldt and H. Smith. Sip service execution rule language framework and requirements, November 2001.

[22] P. Zave. Address translation in telecommunication features. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 13(1):1–36, January 2004.