Experience with Component-Based Development of a Telecommunication Service

Gregory W. Bond, Eric Cheung, Healfdene H. Goguen, Karrie J. Hanson, Don Henderson, Gerald M. Karam, K. Hal Purdy, Thomas M. Smith, and Pamela Zave

AT&T Laboratories—Research, Florham Park, NJ 07932, USA
bond,cheung,hhg,karrie,don,karam,khp,tsmith,pamela@research.att.com

Abstract. AT&T CallVantage^{SM} service is a consumer broadband voice-over-Internet-protocol (VoIP) service. Its feature server has a component-based architecture. This paper is a brief report on our experience with building and deploying advanced telecommunication features using component-based technology.

1 Introduction

Distributed Feature Composition (DFC) is a component-based software architecture for the development of telecommunication services [4]. In AT&T Research we have built an Internet-based implementation of DFC [2]. We have also built the iStudio platform for constructing Web services with an emphasis on reuse [7], and integrated the two service platforms as an application server called V+Plus.

AT&T CallVantage^{SM} service is a consumer broadband voice-over-Internet-protocol (VoIP) service whose advanced features are built and deployed on V+Plus. The service architecture uses the well-known VoIP protocol SIP [5]. V+Plus functions within the service architecture as a SIP application server.

The AT&T CallVantage^{SM} service was launched in March 2004, and is now available in most of the United States. It has received a great deal of favorable press coverage, particularly for its advanced features and voice quality. It has merited a VoIP Service Provider Award from *Internet Telephony* magazine, and *PC Magazine's* Editors' Choice Award.

This paper is a brief overview of our experience with specifying, developing, deploying, and maintaining the service's advanced features, beginning in May 2003. It focuses on the use of components in this software.

2 Components in DFC

In telecommunication software, a *feature* is an increment of functionality added to the basic communication capability. Features are both the work units for software development and the concepts through which a telecommunication service is explained to its users.

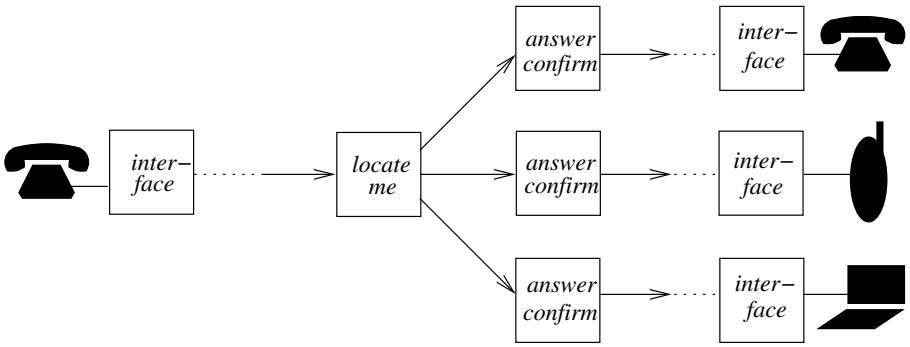

Fig. 1. The Locate Me feature is implemented with two box types.

The telecommunication world has long acknowledged the difficulty and importance of being able to add, delete, and modify the features of a complex telecommunication service. This calls for a style of modularity in which the modules are features. It is also well-understood that features have many interactions, and that managing these interactions is critical to reliability and user satisfaction. This calls for structured composition of feature modules, so that their interactions can be predicted and controlled.

The DFC architecture was designed to provide feature modularity and structured feature composition. It is an adaptation of the idea of *pipes and filters* [6] to the application domain of telecommunications.

In DFC, a request for service is satisfied by a dynamically assembled graph of *boxes* and *internal calls*. A box (filter) is a concurrent process implementing interface or feature functions. An internal call (pipe) is a featureless, point-to-point connection containing a two-way signaling channel and any number of media channels.

In the context of this paper, a DFC internal call is like a plain, old-fashioned telephone call. So the simplest useful graph has two interface boxes, each representing a telephone or other device, connected by one DFC internal call.

More typically, many features apply to each request for service. When features apply, the dynamically assembled graph contains feature boxes implementing these features. Figure 1 shows a fragment of such a graph. Note that a typical connection path between two interface boxes is a chain containing many feature boxes and internal calls.

When a feature box is inactive it behaves transparently. For a feature box between two internal calls, transparent behavior consists simply of connecting their media channels and relaying signals between them. When a feature box is active, on the other hand, it has the autonomy and power to affect communication in any way required. It can place, receive, and tear down internal calls. It can manipulate media channels. It can also absorb, generate, or alter signals as well as propagating them transparently.

The formal definition of DFC [11] falls into three parts. The *protocol* governs how internal calls, signaling channels, and media channels are established and used. The *data model* partitions persistent data, which can be read and written by boxes. The *routing algorithm* controls how boxes of various types are assembled dynamically into connection graphs. The routing algorithm is invoked each time a box places an internal call, and it selects the type of box that will be instantiated or located to receive the call.

The routing algorithm uses data in two categories. *Subscriptions* indicate which addresses (telephone numbers) subscribe to which features. *Precedence* governs the order of features boxes along paths within a graph, and is the primary mechanism for managing feature interactions.

The DFC protocol and routing algorithm are designed so that each feature is optional and each feature box is context-independent—it does not know or need to know which other feature boxes are present. This is the fundamental source of modularity in a pipes-and-filters architecture.

To clarify terminology, a *box* is a dynamically created and assembled component in the architecture. The box's *type* corresponds to a program of which the box is an instantiation.

3 How DFC boxes are used in the service

3.1 Boxes as identified features

Considering all its versions, AT&T CallVantage^{SM} service has 25 features that have been identified and named, either externally (to users) or internally (as units of software development).

Of the 25 features, five (Caller Identification, Caller Identification Blocking, Call Forwarding, Call Waiting, and Three-Way Calling) are basic telecommunication features implemented in a VoIP switch. Two (International Billing, International Call Screening) are implemented in routing components. Three (Phone Book, Simple Reach Numbers, Voicemail eFeatures) are implemented wholly in the Web and data facilities of V+Plus. The remaining 15 are implemented, wholly or in part, by DFC boxes as shown in the table.

Most commonly, there is a one-to-one correspondence between telecommunication features and DFC box types. Occasionally a box type implements more than one feature. For example, a *voice portal* box implements the Personal Call Manager feature. The same box type also implements the Speed Dial function, which is identified to users as a feature.

Also, some features must be implemented using boxes of more than one type. A good example is Locate Me, as shown in Figure 1. An instance of the *locate me* box type can place internal calls—in parallel—to several possible telephones where its subscriber might be located. When one of these attempts succeeds, *locate me* aborts the others and connects the answered telephone with the caller.

DFC Box Type	Features Implemented	Other Purpose
answer confirm	Locate Me	
blind transfer	Add Callers	
call blocking	Call Blocking	
call log	Call Log	
click-to-dial	Click-to-Dial, Record and Send	
conference manager	Personal Conferencing	
de-identification		adaptor
do not disturb	Do Not Disturb	
identification		adaptor
iStudio interface		adaptor
join	Personal Conferencing	
locate me	Locate Me	
mid-call move	Switch Phones^{SM}	
mid-call offer	Switch Phones^{SM}	
phonebook name	Phonebook Name	
rendezvous	Add Callers	
remote identification	Personal Call Manager	
safe forwarding number	Safe Forwarding Number	
send to voicemail	Send to Voicemail	
SIP interface		adaptor
ten-way calling	Add Callers	
tone generator		adaptor
voice mail	Voice Mail	
voice portal	Speed Dial, Personal Call Manager	
voice user interface		adaptor
VoiceXML interface		adaptor

An *answer confirm* box performs another function of the Locate Me feature. If the callee telephone is answered, it prompts for a confirmation that the phone has been answered by the person requested by the caller; if it does not receive a confirming response, it does not propagate a success signal upstream to the *locate me* box. There is an instance of *answer confirm* for each parallel attempt, which is why it must be separated from the *locate me* box.

The *answer confirm* function is particularly valuable when one of the telephones is part of a cellular network with its own Voice Mail feature. Without *answer confirm*, every time the cellphone is unavailable, the call to it will be "answered" by cellphone Voice Mail. This "answer" will probably precede all other answers, aborting the other attempts and subverting the purpose of *locate me*.

3.2 Boxes as reusable building blocks

From another perspective, DFC boxes are reusable building blocks for building telecommunication services. There is reuse at several levels.

At the highest level, we reused whole features from prototype systems we have built in the past. Some required minor modifications to fit into the environment of the new service.

At a lower level, the design of a feature is sometimes influenced by the existence of box types that can be used as generic components to implement it. This is the primary reason why Add Callers—a complex feature that allows the spontaneous formation of conferences of up to ten people—is implemented using three DFC box types.

At a lower level yet, programs are reused to create new box types. We have a "redirect on failure" box program that is used, with modifications for failure type and redirection address, to create the *voice mail* and *safe forwarding number* box types. We have an "address translation" box program that is used, with parameters for regular expression to be matched and string to be substituted, to create any box type that modifies the addresses (telephone numbers) in the signals that initiate internal calls.

At the lowest level, our programming language for boxes [3] allows us to identify and package program fragments for reuse in other box programs. We have amassed a significant collection of such fragments.

3.3 Boxes as adaptors

The V+Plus application server operates in an environment with many other hardware components such as VoIP switches, gateways, routers, telephone adaptors, and media servers. Despite the fact that SIP is a standardized protocol, all VoIP technology is immature, and integration problems are commonplace.

Given the fact that a DFC box is a filter in a pipes-and-filters architecture, it is not surprising that we find them useful as adaptors. The most important adaptors are *interface* box types. Interface boxes form the periphery of DFC graphs. A *SIP interface* box translates between SIP and the DFC protocol, which is better suited for component composition than SIP. An *iStudio interface* box allows Web services to launch telecommunication activities (see Sections 3.4 and 4). A *VoiceXML interface* allows feature boxes to place calls to media servers capable of running VoiceXML scripts that specify interactive voice-response dialogues.

We put code in interface boxes and other adaptors to solve integration problems as they arise. Modularity is particularly beneficial in this context, because the adaptors represent short-term or localized decisions that we would not wish to embed deeply in feature code.

For example, the *identification* and *de-identification* box types bridge a conceptual gap between DFC and the architecture of the service as a whole. In DFC routing decisions are based strictly on addresses. In the architecture of the service as a whole, routing decisions can also depend on which hardware component originates the routing request. Fortunately, well-placed adaptors can convert from one kind of state to the other.

A *tone generator* box is another kind of adaptor. For the most part, generation of "progress" tones such as busytone and ringback is the responsibility of hardware components independent of V+Plus. However, deficiencies in SIP

prevent these components from getting the necessary signals under all circumstances. When SIP cannot carry the necessary signals, a *tone generator* box (with the help of a media server) generates the tone and inserts it into the voice channel.

3.4 Boxes as interfaces to Web services

Persistent data is the interface between telecommunication and Web services, as it can be read and written by both. For example, *call log* boxes record history that can be accessed by subscribers via the Web. A *phonebook name* box looks up the name corresponding to a calling telephone number in the callee's phonebook, and substitutes the name for the number in Caller Identification.

Sometimes the interaction between the two aspects of the service is more active. The Click-to-Dial feature is activated by a Web service when a subscriber clicks on a telephone number. The Record and Send feature calls a list of telephone numbers, delivering a prerecorded message to each; it is activated from the Web and implemented by repeated activations of Click-to-Dial.

4 Component-based development of data views in the service

Both Web browsers and telephones are end-user interfaces for AT&T CallVantage^{SM} service. Through them, users can enable or disable features, change feature settings, or access personal content such as voicemail messages. The iStudio [7] architecture offers a component-based implementation of these interfaces that is complementary to boxes in the DFC architecture.

Similar to Apache Struts [1] in philosophy, iStudio provides us with a mechanism for supporting software objects that compartmentalize the data for features such as Call Log, Voice Mail, and Locate Me. Each component manages the database tables and operations for its own feature.

iStudio accesses the database on behalf of all other software in V+Plus. It produces HTML for visual Web pages, VoiceXML scripts for interactive voice-response dialogues, and data values for use by DFC boxes. The generators of these data views all share the feature-specific software objects mentioned above.

The Click-to-Dial and Record and Send features are activated by Web services. A user request for one of these features is delivered from the Web application to an *iStudio interface* box that places a DFC internal call to begin assembly of a graph of feature boxes.

5 History and evaluation of software development with V+Plus

We delivered the first 11 features to a test organization two months from the inception of the project. It was possible to fit requirements specification, design, and implementation into this extremely short period only because of much reuse.

Reuse of code, as described in Section 3.2, is obviously important. Equally important is the reuse of domain knowledge based on the DFC architecture.

The DFC architecture constrains how features can interact, and is therefore a foundation for theories of feature interaction. Such theories predict how features *can* interact, justify how they *should* interact, and provide design constraints proven to satisfy correctness in these terms.

These theories are still immature (see [9, 8, 10] for examples). A rudimentary understanding is better than none at all, however, and was extremely helpful to us in predicting feature interactions and in deciding how to manage them. For example, all of Locate Me, Do Not Disturb, Voice Mail, Call Blocking, Send to Voicemail, and Safe Forwarding Number make decisions concerning the disposition of incoming calls. They must interact so that exactly the right features, in the right order, are activated in each situation.

A few feature interactions compromise modularity to the extent that one feature must be programmed with another feature in mind. For example, a *voice mail* box generates a special signal so that a *call log* box knows whether or not a caller recorded a message. This is necessary because the basic DFC protocol does not distinguish these cases. It is always possible, however, to program cooperating feature boxes so that neither breaks if the other is absent.

The implementation of AT&T CallVantage^{SM} features is not a trivial use of components. In the first release of the service a connection path between two subscribers could contain 20 DFC boxes, even without any forwarding (forwarding to other subscribers would increase the number of boxes by seven per forward).

The first release of the service was deployed in a consumer trial which began October 2003. In preparation for the first generally available release in March 2004, we removed a few features from the trial version and made a major change in the media handling. Feature removal was easy due to feature modularity in both DFC and iStudio. The need to change media handling is typical of a rapidly evolving technology, in which the available resources and capabilities can change frequently. The software modification was accomplished quickly, in part because the DFC architecture maintains a separation of concerns between the service layer (features) and the network layer (resources).

Subsequent software development has entailed new feature development, maintenance, and performance optimization. As expected, adding new features to the service is easy. There have been relatively few bugs in feature code.

Most maintenance issues are system-integration problems, arising from the immaturity of VoIP technology. Unfortunately they have arisen frequently and will continue to arise for some time to come; at this point almost every new function added to the service exposes new incompatibilities among the hardware components of the service architecture.

It is inevitable that the modularity of DFC will exact a performance penalty. Our measurements indicate that the penalty is small compared to VoIP performance issues that are independent of feature modularity.

We are working toward accurate performance assessments. Meaningful comparisons between implementation alternatives are difficult to obtain, however,

because they require multiple implementations of equivalent feature sets, not to mention adequate time in a laboratory full of expensive test equipment.

6 Conclusion

V+Plus was originally built as a research prototype. Nevertheless, it continues to provide the advanced features of a nationwide consumer telecommunication service built on rapidly evolving technology.

Our experience demonstrates the feasibility and value of a component-based architecture in the area of telecommunications. The experience is particularly interesting because the component model is based on pipes and filters rather than the more common object-oriented programming. Object-oriented programming is also present—most of our infrastructure code is written in Java—but at a lower level of abstraction than the components discussed here.

In the community of researchers concerned with feature interactions and telecommunication software, the DFC component model has been considered interesting but radical and impractical. Our experience demonstrates that it is adoptable and completely practical. None of us would dare to work with a technology as complex and volatile as VoIP without this kind of support for evolution and adaptation.

References

1. Apache Struts. `http://struts.apache.org`.
2. G. W. Bond, E. Cheung, K. H. Purdy, P. Zave, and J. C. Ramming. An open architecture for next-generation telecommunication services. *ACM Transactions on Internet Technology*, 4(1):83–123, February 2004.
3. G. W. Bond and H. H. Goguen. ECharts: Balancing design and implementation. In *Proceedings of the Sixth IASTED International Conference on Software Engineering and Applications*, pages 149–155. ACTA Press, 2002.
4. M. Jackson and P. Zave. Distributed Feature Composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering*, 24(10):831–847, October 1998.
5. 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.
6. M. Shaw and D. Garlan. *Software Architecture*. Prentice-Hall, 1996.
7. A. H. Skarra, K. J. Hanson, G. M. Karam, and J. S. Elliott. The iStudio environment: An experience report. In *Proceedings of the International Workshop on XML Technologies and Software Engineering*, May 2001.
8. P. Zave. An experiment in feature engineering. In A. McIver and C. Morgan, editors, *Programming Methodology*, pages 353–377. Springer-Verlag, 2003.
9. P. Zave. Address translation in telecommunication features. *ACM Transactions on Software Engineering and Methodology*, 13(1):1–36, January 2004.
10. P. Zave, H. H. Goguen, and T. M. Smith. Component coordination: A telecommunication case study. *Computer Networks*, 45(5):645–664, August 2004.
11. P. Zave and M. Jackson. *The DFC Manual*. AT&T, 2001. Updated as needed. Available from `http://www.research.att.com/projects/dfc`.