

# An Application Router for SIP Servlet Application Composition

Eric Cheung and K. Hal Purdy, *Members, IEEE*

**Abstract**—The SIP servlet standard offers a familiar Java application programming interface (API) for developing Session Initiation Protocol (SIP) applications, and the upcoming version 1.1 specification standardizes how multiple SIP servlet applications are composed and invoked to service a communication episode. Such disciplined composition promotes software modularity and reuse, and also helps manage feature interactions. At the core of the application composition framework in the SIP servlet API 1.1 is the Application Router that performs the task of selecting an application to service a SIP request. In this paper, the authors describe an application router implementation based on the Distributed Feature Composition (DFC) architecture routing algorithm (DFC-AR). While the DFC-AR is designed to be simple so it can serve as a sample implementation, it is also sufficiently powerful for real-world VoIP service deployments. The DFC-AR has been proposed to be the default AR in the reference implementation of the 1.1 specification.

**Index Terms**—Communication system software, Feature Interaction, Programming Environment, Voice over IP, IMS

## I. INTRODUCTION

The SIP servlet standard is defined in the Java Community Process (JCP)[1]. It defines a container-based model for building and deploying Session Initiation Protocol (SIP)[2] applications. In a SIP network, a SIP servlet container hosting one or more applications acts as an application server. For example, 3GPP defines SIP servlet containers as one of the standard application server types in the IP Multimedia Subsystem (IMS) architecture[3].

Application developers use the Java SIP servlet API (SSAPI) to interface with the container. This API builds on the well-established Servlet API used for programming HTTP servlets; thus it offers a programming model that is familiar to a large developer community. SSAPI provides a level of abstraction that is roughly between the SIP transaction layer and transaction users. While it affords the programmers access and control over protocol details such as SIP messages headers and contents, the container is responsible for handling details such as message formatting, retransmission, messages correlation, and protocol timeout. The container also provides additional facilities for proxy, user agent, and back-to-back user agent (B2BUA) applications, further simplifying the task of the programmers. The container manages application state for servlets, and distributed containers also provide state

replication, load balancing and automatic fail-over capabilities.

In March 2003, the JCP released the 1.0 specification (SSAPI-1.0). At the time of writing, a new version (SSAPI-1.1) [4] is under development and is planned to be released in 2008. Besides additional support for application composition, the other major enhancements include support for converged applications that handle multiple protocols such as HTTP and instant messaging. Currently there are at least five commercial and one open source SIP servlet containers, making it arguably the best supported SIP application environment.

### A. Composition of SIP Servlet Applications

A SIP servlet application provides features to its subscriber when people engage in communication activities. Application composition occurs when two or more applications are combined to provide a more complete and complex service to the users. In an environment that supports application composition, it is desirable that each application involved in providing such features be able to do so independently of and without interfering with other applications. This feature modularity helps with specification, design, implementation and testing of applications, and encourages re-use. It also allows multiple applications developed by unrelated parties to be composed in a well-behaved manner.

Separate applications active on the same communication episode need to be able to combine their functionality in a well understood, controlled manner. This is also important to facilitate the analysis and management of feature interactions.

SSAPI-1.0 supports application composition at the SIP signaling level. When the container receives an initial SIP request, it selects an application to invoke. As each invoked application relays the request (as a proxy or B2BUA), another application may be invoked, thus forming a chain of applications. Each invoked application operates in an environment as if it were the only application and can send and receive SIP requests and responses, although in reality the message exchanges may be with the adjacent applications (upstream and downstream) in the chain. Figure 1(a) illustrates application composition in SIP servlet containers.

SSAPI-1.1 further standardizes the application composition process, and introduces a new component called the *application router* (AR) that is responsible for application selection, and new concepts including routing regions and routing directives. Much of the SSAPI-1.1 work on application composition is based on the theoretical architecture of Distributed Feature Composition (DFC) [5]. The next section introduces the DFC architecture.

Revised 16 January, 2008.

The authors are with AT&T Labs Research, 180 Park Avenue, Florham Park, NJ 07932 USA (cheung@research.att.com, khp@research.att.com).

This work has been submitted to the IEEE for possible publication. Copyright may be transferred without notice, after which this version may no longer be accessible.

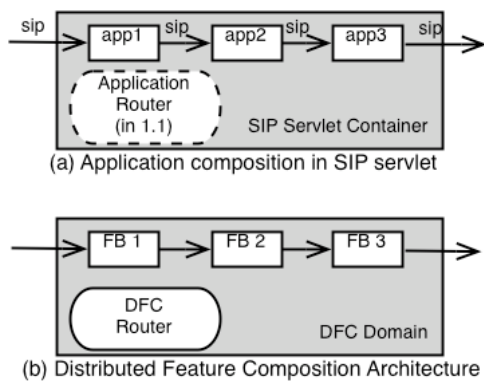


Figure 1. Application composition in SSAPI and DFC

#### A. Distributed Feature Composition Architecture

DFC is a virtual architecture designed for structured feature composition and the analysis and management of feature interaction. It is a pipe-and-filter software architecture in which multiple *feature boxes* are linked by *internal featureless calls*, thus forming a (not necessarily linear) chain of feature boxes. By nature, a communication episode has a caller and a callee, thus DFC defines the *source* and *target regions*, and each call is set up with a *source address* and a *target address*. Feature subscription is based entirely on these addresses, and the feature boxes subscribed to by the source address are invoked first, followed by those subscribed to by the target address. The ordering of the features within each region is extremely important, as the feature closer to its subscriber has higher precedence. A DFC system is provisioned with (a) the list of addresses, (b) the list of feature boxes, (c) the precedence order of the feature boxes in each region, and (d) the list of feature boxes each address subscribes to in each region. A logical entity, the *router*, follows a well-defined routing algorithm to select the next feature box to invoke based on the source and target address of the call setup and the system provisioning. Figure 1(b) illustrates a typical arrangement of feature boxes.

Comparing Figures 1(a) and 1(b), it is easy to see the similarities between application composition in SIP servlet environment and the DFC architecture. For this reason, and also based on our extensive and positive experience with using DFC in real-world telecommunication services, the authors have introduced most of the DFC concepts to SSAPI-1.1 and are the main contributors in the expert group. We presented this work in an earlier paper[6]. SSAPI-1.1 is general and flexible and does not dictate the use of the DFC routing algorithm, and deployers of SIP servlet applications may choose any routing algorithm or strategy by providing their own application router implementation. At the time of writing the authors are aware of several groups developing application router implementations.

In this paper, we present a simple but powerful SIP servlet application router that implements the DFC routing algorithm. This application router, the DFC-AR, has been used with SSAPI-1.0 containers successfully. The next section discusses the considerations in its design and adaptation to the SIP servlet environment. Section III describes the implementation of the DFC-AR, followed by section IV with

some examples. Section V discusses future areas of work and improvements.

### I. DESIGN CONSIDERATIONS

Several criteria were of importance when designing the DFC-AR.

#### A. Simplicity

Simplicity was a major design goal for the DFC-AR. This implementation has been proposed as the default application router implementation for SSAPI-1.1. As such, it is important that the design be simple and easily implemented on the variety of platforms that support SIP servlet containers. Further, it is hoped that this implementation shall be suitable for application developers who are experimenting with composing SIP servlet applications. To ease understanding SIP servlet application composition, it is desirable that the application router implementation be easy for a developer to understand.

The design is simplified by having the AR configuration be static and entirely contained within a single XML file. There are no dependencies on external databases or other data stores. The static nature of the DFC-AR configuration means that it need not be complicated with considerations of exercising its routing algorithm in the presence of a dynamically changing configuration. Simplicity is further achieved by “hard coding” the addressing information in an initial SIP request. A more flexible yet complicated design would allow for dynamic configuration of what information in SIP messages would be used by the DFC-AR for matching subscriptions. Further, by fixing the address information to be only URIs, it becomes much easier to compare two addresses for inequality, a capability that the DFC-AR is required to have for its algorithm to work properly. As a design goal, simplicity is chosen even over compliance with the original DFC router specification, as will be seen in the description of how the DFC-AR handles the *REVERSE* directive.

#### A. Flexibility

A design goal second only to that of simplicity is flexibility. As noted, it is hoped that developers can use the DFC-AR while experimenting and learning SIP servlet application composition. The goal of encouraging good software design through composition is not achieved if developers and deployers are unable to achieve flexible configurations involving multiple SIP servlet applications.

Flexibility in the DFC-AR design is achieved mostly by leveraging the power of regular expressions as a mechanism for defining application subscriptions. As users of DFC routing based on regular expressions for some time now, we have found that the use of regular expressions in evaluating the contents of SIP messages is a powerful mechanism for supporting the ability of an application router to choose which applications need to be invoked for a communication session.

### I. DFC APPLICATION ROUTER IMPLEMENTATION

Like the original DFC router specification, the DFC-AR

implementation uses precedence relationships between applications and subscriptions to determine the applications to invoke for a particular SIP initial request.

#### A. Configuration Data

The DFC-AR uses a static XML configuration file to populate the precedence relationships and subscriptions that together constitute the information it needs to route SIP requests to SIP servlet applications.

##### 1) Precedence

For each routing region, originating or terminating, the configuration file allows the deployer to establish a partial order among the applications. Within an ordering element in the XML file, the deployer lists application names. The higher an application name appears in the list, the higher precedence it has. Because proximity to subscribers confers priority, a higher precedence application appears closer to the endpoints when invoked. That is, a higher priority originating region application is invoked before a lesser priority application while a higher priority terminating region application is invoked after a lower priority application.

Not all deployed applications must be named in an ordering list within the precedence section of the XML configuration file. This is because it is perfectly acceptable that some applications do not have any precedence relationship with other applications. For example, a call logging application simply needs to be invoked somewhere among all the invoked applications. Therefore the precedence order among applications in a region is a partial not total ordering.

##### 1) Subscriptions

DFC-AR subscriptions are described in the same XML configuration file in one or more mapping sections. A subscription consists of one or more Java regular expressions combined with one or more application names. The regular expressions in a subscription are evaluated by the DFC-AR against the address fields extracted from the initial SIP requests. The originating address is the From header value including the display-name portion of the value. The terminating address is the Request-URI of the initial SIP request.

The choice of applications to be invoked to handle a given SIP request is governed by the subscriptions that match the SIP initial request. Because multiple subscriptions may be listed in the XML configuration file and because multiple application names may be associated with one regular expression in a mapping element, it is clear that when subscription regular expressions match an address in an initial SIP request, the result may be multiple applications that are to be invoked to handle the request. The fact that multiple applications may be invoked to handle a request is central to the support for application composition in SSAPI-1.1 and therefore the subscription mechanism in the DFC-AR is one of its most important aspects.

#### A. Stored State Information

The DFC-AR needs to carry certain state information from one invocation to the next invocation for the same episode. It keeps it in a character string and delegates its persistence to

the SIP servlet container. The state information comprises:

1. the address that caused the last application to be invoked. This is needed so that the DFC-AR can determine if the application has changed the address when relaying the request thereby requiring computation of a new route list.
1. the route list. This is the remaining ordered list of applications to be invoked *if* the relevant address in the SIP request has not been changed. The first application in the route list is the one that was most recently invoked to handle the initial SIP request. This information is required when an application issues a *REVERSE* directive and the remaining applications to be invoked to handle the reversed SIP request must be calculated. Note also that the inclusion of the remaining route list in the state information is an optimization. Given knowledge of the last application invoked, the route list could be calculated anew on each invocation of the DFC-AR. In this case, the AR would simply calculate the whole route list, find the last invoked application in it, and invoke the next application named in the route list. For performance and robustness reasons in the face of dynamically changing subscriptions, this optimization is warranted.

#### A. The Workings of the DFC Application Router

When the SIP servlet container receives an initial request, it calls the AR providing it with the request, the routing directive, the routing region, and any stored state information from the previous invocation. After the AR performs the application selection, it returns to the container the next application's name, the updated routing region, the subscriber address and the updated state information. This function of the DFC-AR is described below.

##### 1) Determining the Routing Region

Since subscriptions are defined by routing region, either originating or terminating, the DFC-AR must first determine in which routing region it is currently operating. If the initial request has not been seen before by the DFC-AR, then the DFC-AR commences in the originating region as recommended by SSAPI-1.1. If the routing region parameter passed to the DFC-AR from the container is set to originating but the DFC-AR determines that all originating region applications have already been invoked, then the DFC-AR advances to the terminating region. The DFC-AR does not currently make use of the neutral routing region.

##### 1) Examining the Routing Directive

Once the region has been determined the DFC-AR examines the routing directive indicated by the last SIP servlet application invoked. This directive may have one of three values: {*NEW*, *CONTINUE*, *REVERSE*}. When the directive is *NEW*, the DFC-AR must perform the core subscription matching algorithm to build a new route list. If the resulting route list is not empty, the first application name found in the route list is returned to the container. If the route list is empty then either the DFC-AR advances to the terminating region (when in the originating region) or it simply returns a null application name value to the container signaling that there are no further applications to be invoked for this request (when in the terminating region).

If the routing directive is *CONTINUE*, the last application is indicating that it wishes the application routing to proceed within the current routing region. For this case, the DFC-AR must determine whether a new route list must be constructed. If the address in the SIP request was *not* changed by the last application, then the previously computed route list is still valid and the next element specifies the application to be invoked. If the address in the SIP request was changed by the last application then the DFC-AR must build a new route list using the new address.

Determining whether the address has changed or not is accomplished by the DFC-AR by comparing the current address in the SIP request with the SIP address stored in the state information object from the last DFC-AR invocation. For the purposes of comparing addresses in this way, the DFC-AR is currently written to ignore certain SIP URI parameters such as “transport” and “tag”. A nice feature of a future version of the DFC-AR or of other similar AR implementations would be to allow the set of URI parameters excluded when comparing URIs to be a configuration setting.

The last directive, *REVERSE*, signals the DFC-AR that it must reverse the direction of the call in an application chain that extends from and includes the last invoked application. To reverse a request, the DFC-AR first switches the routing region (i.e. from originating to terminating or vice versa). Using the appropriate address for the new routing region, the DFC-AR then builds a new route list using the core algorithm. However, the semantics of the *REVERSE* directive dictate that only those applications appearing **after** the last invoked application should be invoked to handle the reversed request. Thus, the DFC-AR computes the suffix of the route list that includes only those application names appearing in the route list after the last invoked application (the last invoked application is the first entry in the stored route list in the state information.) While the original DFC specification of *REVERSE* handling dictated some additional constraints on subscriptions to guarantee that the last invoked application name would appear in the route list, the DFC-AR implementation is simplified and does not impose these additional constraints. Hence, it is possible that the name of the last invoked application does **not** appear in the computed route list. The DFC-AR handles this case by simply not computing the aforementioned route list suffix and using the computed route list as is.

#### 1) The Core Algorithm: Building a Route List

When the DFC-AR recognizes that it must build a route list that gives the list of applications to be invoked for a particular initial request it proceeds as follows:

1. Based on the routing region, extract the address from the SIP request.
1. Examine sequentially each subscription for the current routing region. If the extracted address from the SIP request matches the regular expression in the subscription, add all the application names in the subscription to the route set of candidate application names.
1. Eliminate duplicates. If any application name appears multiple times in the route set, eliminate the duplicates.

1. Sort the route set. Using the resulting route set, use the partial ordering established in the precedence section of the XML file configuration to sort the route set into an ordered route list compatible with the partial ordering.

The resultant route list specifies the complete list of applications to be invoked for the address extracted in step 1. The first element of this route list specifies the next application to be invoked to handle the SIP request.

## I. EXAMPLES

This section presents three example to illustrate the operation of the DFC-AR. Consider a system in which all addresses subscribe to Outbound Call Screening (OCS) in the originating region, Call Forwarding (CF) in the terminating region, and Call Waiting (CW), 3-way Call (3WC) in both regions. The precedence among these applications is fully specified. The XML configuration file will contain:

Precedence:

Originating: (highest) CW, 3WC, OCS (lowest)

Terminating: (highest) CW, 3WC, CF (lowest)

Subscriptions:

Originating: .\*sip:.\* subscribes to OCS, CW, 3WC

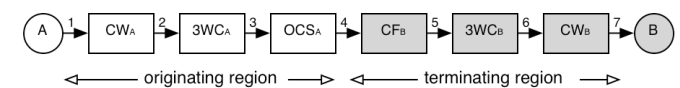
Terminating: sip:.\* subscribes to CW, 3WC, CF

### A. No Address Change

In this example, A calls B and all applications simply relay the request without changing the From header or the Request-URI. The SIP INVITE from A to the container, the INVITE requests between applications, and the outgoing INVITE requests all have the form:

```
INVITE sip:B@example2.com
From: "Alice" sip:A@example1.com
```

This results in the application chain below:



In step 1, container calls the AR with the INVITE request and NEW directive. AR sets originating region, sets subscriber address to the From header (i.e. “Alice” sip:A@example1.com), and finds matching subscription and sets route list to CW,3WC,OCS. It then returns CW as the next application to the container. In step 2, the container calls AR with the request that CW<sub>A</sub> sends, *CONTINUE* directive, originating region, and the previous state information. Comparing the subscriber address in the stored state and the request, AR determines there is no change in originating address and simply obtains the route list from the stored state and removes the first entry, and returns 3WC to the container. Step 3 is similar. In step 4, AR determines no change in originating address, and after removing OCS the route list is empty. It advances to terminating region, sets subscriber address to the Request-URI (i.e. sip:B@example2.com) and finds matching subscriptions, sets route list to CF,3WC,CW and returns CF to container. Steps 5 and 6 are similar. In step 7, after removing CW the route list is empty. The AR then returns no application to container, and container sends the request externally to B.

### B. Address Change

In this example, B has configured his CF app to forward all calls to C. Therefore when A calls B, CF<sub>B</sub> changes Request-URI to forward the call to C. This forms the chain:



The difference lies in step 5. AR upon examining the request detects that the terminating address has changed from B in the stored state to C. AR finds applications for C, sets route list to CF,3WC,CW and returns CF to container.

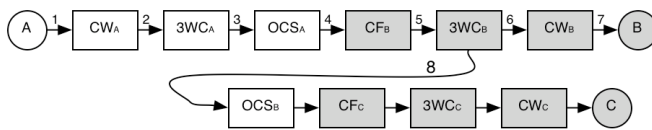
### C. Reverse Directive

Returning to example A, after the call from A to B is established, B activates 3WC<sub>B</sub> to add C to the conversation. In step 8, 3WC<sub>B</sub> sends request with *REVERSE* directive, and the request has the form:

```

INVITE sip:C@example2.com
From: "Bob" sip:B@example1.com
  
```

AR receives the request, *REVERSE* directive, and stored route list of 3WC,CW. Because of the *REVERSE* directive, AR switches from terminating to originating region. B's originating applications are CW,3WC,OCS. Using 3WC in stored route list to calculate the suffix, the new route list becomes OCS, and AR returns OCS to the container. This is followed by C's terminating applications, resulting in the following chain:



## II. FUTURE WORK AND CONCLUSION

The introduction of the application router component into SSAPI-1.1 has opened up new vistas for the deployer of advanced telecommunication services. For the first time in the SIP application server industry, there is a standardized mechanism by which the deployer, as opposed to the developer, can have flexible and powerful control over how and when multiple applications may be invoked to act on behalf of subscribers. For this concept to flourish, however, a community focused on application router technology must arise so that deployers understand what application routers are and have a broad selection of appropriate implementations to choose from. We have described here an application router implementation, the DFC application router, that has been submitted to the JCP expert group for inclusion in the SSAPI-1.1 Reference Implementation as the "default" application router. This implementation is simple enough to be easily understood and used by developers and deployers who are first experimenting with an application router and yet is flexible enough to support good application composition in a wide variety of application deployment scenarios. The implementation described here has been in use for over a year by means of an adaptation layer that allows its use on SSAPI-1.0. Furthermore, in hopes of fostering the necessary community around application router technology, this implementation is available as open source software[7].

While the DFC-AR is a good, simple application router implementation, there is a great deal more fruitful work to be done. First are enhancements that could be made to the DFC-AR. Production quality application router implementations will be obliged to handle dynamic, run-time changes in application subscriptions. In the face of such changes, the application router must nevertheless employ its core routing algorithm in a robust manner designed to give consistent results. Designing and implementing such a robust routing algorithm in a future version of the DFC-AR would further help the community interested in production grade application router technology.

Another enhancement that would provide additional flexibility at the price of a moderate increase in complexity would be allowing the choice of subscription addressing information to be a configuration setting rather than hard coded as it is in the current implementation. This enhancement could be particularly useful in an IMS environment where the P-Asserted-Identity header value would be a much better choice for determining subscriptions on an originating IMS application server.

Entirely different application router implementations are also an area of active research. For example, it is recognized that the SSAPI-1.1 application router component is functionally quite similar to the IMS Service Capability Manager (SCIM) defined in 3GPP specifications. Could a SSAPI-1.1 application router be built that instantiates a compliant SCIM? Would such an implementation be preferable to other SCIM implementations? Are there other application router implementations that would be better suited for use in an IMS environment? One example of such an implementation would be an application router that uses the Sh interface defined in 3GPP to access subscription information directly from the Home Subscriber Server (HSS) database. Along with the new concepts already embodied in SSAPI-1.1 and the work presented here, such future results should further ensure the success of the SIP servlet environment as the leading standards based environment for SIP application servers.

### ACKNOWLEDGMENT

The authors acknowledge our colleagues, Greg Bond, Tom Smith, Venkita Subramonian, and Pamela Zave for their invaluable ideas, comments and suggestions.

### REFERENCES

- [1] Anders Kristensen. SIP Servlet API Version 1.0. 2003. <http://jcp.org/en/jsr/detail?id=116>
- [2] J. Rosenberg et al, *SIP: Session Initiation Protocol*, IETF RFC3261, 2002.
- [3] 3GPP IP Multimedia Subsystem (IMS). <http://www.3gpp.org/>.
- [4] JSR 289. <http://jcp.org/en/jsr/detail?id=289>
- [5] M. Jackson, P. Zave, *Distributed feature composition: A virtual architecture for telecommunication services*, IEEE Trans. on Software Engineering XXIV(10):831-847, Oct 1998.
- [6] E. Cheung, K. H. Purdy, *Application Composition in the SIP Servlet Environment*, IEEE International Conference on Communications, Pages 1985-1990, 2007.
- [7] ECharts for SIP Servlets. <http://echarts.org/>.