RC24346 (C0709-003) September 10, 2007
Electrical Engineering

IBM Research Report

SIP Application Composition Framework Based on Business Rules

Lina Ren, Jia Jia Wen, Qi Yu
IBM Research Division
China Research Laboratory
Building 19, Zhouguancun Software Park
8 Dongbeiwang West Road, Haidian District
Beijing, 100094
P.R.C.

Lee Longmore
IBM S&D Communication Sector

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

SIP Application Composition Framework Based on Business Rules

Lina Ren, Jia Jia Wen, Qi Yu
IBM China Research Lab
E-mail: {renlina, wenjj, yuqi}@cn.ibm.com

Lee Longmore
IBM S&D Communication Sector
lee.longmore@uk.ibm.com

***Abstract*- This paper presents a new SIP application composition framework, which can enable service operators to manage SIP application compositions using intuitive business rules. The framework supports complex conditions and feature interaction management when making application composition decisions. The requirements of SIP application composition is studied from service operator's point of view, and possible extensions to current JSR 289 SIP application composition mechanism are discussed. We give example implementations to illustrate how the rule-based framework can be integrated to SIP Servlet environment.**

Keywords-Telecommunication System Software, Voice over IP Services, Application Composition, Rule Engine Technologies

I. INTRODUCTION

Session Initiation Protocol (SIP) [1] is an application-layer control (signaling) protocol designed for creating, modifying, and terminating communication sessions with one or more participants. SIP has been chosen as the core signaling protocol in IMS [2] network for its simplicity and extensibility. The SIP Servlet standard is a popular Java application programming interface (API) for developing and deploying SIP applications in Voice over IP (VoIP) environments. It is built on the base of the programming model of HTTP Servlets, and is well supported in industry implementations. A number of commercial container implementations conforming to the specification are available.

Generally speaking, application composition in telecommunications means the situation that several software applications are actively serving one or more participants involved in a communication session or related sessions. While the applications are independent of each other, providing their features separately, their functionalities are actually combined during the communication, and are experienced as a whole by service users. For example, an Outgoing Call Barring application and Speed Dial application can be both triggered to serve the same user in a voice call, and thus the two applications form a composition.

Application composition is inevitable in present and future telecom networks, for it is mainstream trend that telecom network opens its functionality and telecom applications are created by different parties. The "separation-of-concerns" design of IMS service layer provides the complete framework for developers to plug various telecom applications serving end users, and it is sure that those applications would form application compositions easily in most of communication sessions.

It is well known that application composition sometimes is sometimes a trouble to users and telecom operators. Unwanted feature interaction [3] between different applications is an outstanding problem, which has no perfect solutions.

It is also noted that the benefits of application composition are obvious. Through the well-coordinated cooperation of multiple applications, end users can get better experiences. To operators, appropriate application composition helps them to provide new set of features with low costs, by just reusing their existing "old" services and integrating them in new ways. Thus, a service in operation is not only a service it is designed for, but a functional component that can be weaved into other services. Moreover, if telecom operators give end users such flexibility to customize their own application compositions, service quality would be greatly improved.

As a result, the ability to manage application composition in telecom network is of growing importance to today's telecom service providers.

However, current SIP Servlet environment has not provided enough support in this area yet. SIP Servlet API 1.0 specification JSR116 [4] stated the goal that several applications can possibly execute on the same request or response independently in a "well-defined and orderly" fashion. But the specification didn't standardize such approach in detail. Since there is the need for enhancement, people are working around the specification of JSR 289 [5] since 2006 to author a new SIP Servlet API 1.1 specification, mainly to address the application composition support. In the new specification draft, a new component called *application router* is proposed to control the application selection/compositions inside one application server as well as among different application servers. This application router should be controlled by application deployer, so application developers can focus on implementing individual applications. Unfortunately, although the new specification presented the entity of application router,

it does not specify how application router decide the application selection and composition process, only indicates that this is left to application deployers.

These considerations motivated our work in this paper. We believe that it is crucial to think through the requirements of composing the applications in their networks, the required flexibilities, influencing factors, changing conditions, and dynamic monitoring of application composition. These issues should impact application composition collectively. We also believe that providing a mechanism to cover those requirements is important, for application composition is a very practical problem that service operators will face when they try to provide flexible and componentized services. So we discussed these issues and proposed a solution in this paper.

This paper is organized as follows. Section II overviews the operational requirements of SIP application composition and presents the idea of rule-based application composition in SIP Servlet environment. Section III gives the example implementation of the proposed application composition framework and analyses the system from different perspectives. Section VI reviews some related works. Finally, the paper concludes in Section V.

II. RULE-BASED APPLICATION COMPOSITION

In this section, we introduce our rule-based approach to instruct JSR 289 application composition. First, let us look at the design of JSR289 application composition, as the basis for our discussion.

A. *JSR 289 Application Composition*

JSR289 specification defines version 1.1 of the SIP Servlet API. It enhances version 1.0(JSR116) especially in the area of application selection and composition. The enhancement can be summarized as follows:

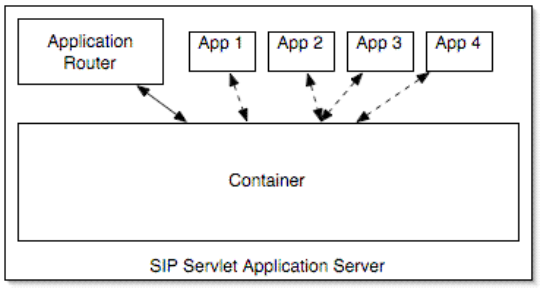

Figure 1. JSR289 Application Composition Architecture

1) Application Selection. JSR289 introduces the concept of application regions: originating, terminating and neutral. AR will select the originating applications for calling party, and the terminating applications for called party; neutral applications shall be invoked for both calling party and called party. JSR289 does not define how one should derive the session case from the request received from an application or the core network.

2) Application Composition. Application Router (AR) decides the order of application composition - it is noted that the specification outlines the high-level requirements of the AR only; the design and implementation is not specified. The Container invokes AR to select the next application name, and dispatches the request to the selected application.

3) Container and Application Router.

a) Container: The JSR289 Container is responsible for creating and maintaining application selection state associated with each initial request it receives externally or from applications. It also invokes the AR with the application selection state information to obtain the name of the application to service the initial request, selects the SIP Servlet within the application, and dispatches the request to that SIP Servlet. The JSR289 Container should also maintain application selection state and invoke the AR to obtain the next application name. The application selection state includes the following information: (1) the routing directive used to create this request; (2) region of invocation (originating or terminating); (3) the URI that the selected application is invoked to serve; (4) arbitrary application router state information.

b) Application Router: The AR plays a central role in the application selection process. It is logically separated from the Container and an API is defined for the communication between the Container and the AR. The Container calls upon the AR to choose which applications to invoke in response to an initial request. The AR decides the application composition order. It does not invoke the applications directly. The Container is responsible for invoking the AR and dispatching the request to the application. Although AR is a very important component in JSR289, the specification does not describe the architecture of AR in detail, and it is logically separate from the container. So the JSR289 AR can have many different implementations, as long as the interface with Container is supported.

The major AR functional responsibilities include: (1) Dynamically maintain the current list of applications deployed in the Container. This is achieved by getting names of deployed applications during initiation and getting notifications from Container afterwards. It should be noted that according to JSR289, it is the responsibility of Container to notify AR what applications has been deployed locally, but the specification does not specify explicitly how the AR should be informed about external applications. (2) By utilizing a range of information sources - user and service profile, received initial Servlet request, routing region, routing directive, state information – the AR selects the next application (local or external) to invoke, and provides information in support of application selection to the container (for local application: the application name, subscriber URI, and state information; for external application: the external route). JSR289 does not specify in detail how applications should be selected and composed.

4) Application Composition Process: On receiving an initial request (from the external network or an application hosted by the Container), the Container invokes the AR to obtain the next application to invoke.

If the next application is local to the Container, the AR returns the application name, subscriber URI, and some state

information to the Container, which in turn dispatches the request to specified SIP Servlet of the application;

If the next application is external to the Container, the AR returns the external route which points to the location of the next application to the Container, which in turn adds the external route to the top Route header and sends the request out.

It should be noted that JSR289 does not specify in detail the requirements around inter-Container application routing; for example, the manner in which an AR should select external applications is not detailed.

B. Extending Application Composition with Rules

As discussed in JSR 289 application composition introduction, the problem of how to instruct application selection and composition in an flexible way is not addressed, while in practice such support is vital for operators to manage their services. We introduce the use of intuitive rules to aid operators at this point.

Rule engine technologies are evolved from rule-based expert system. They represent knowledge with rules, and use its set of rules to reason and reach a conclusion (often take some actions). A rule engine may be viewed as a sophisticated if/then statement interpreter. The if/then statements that are interpreted are called rules. A rule is typically composed of two parts: a condition and an action. Simply put, when the condition is met, the action is executed.

Rule engines have been widely used as embeddable components in software systems in order to extract business policies out from codes, so non-technical people like business analysts can access, understand, and manipulate business policies of the system with little effort and quick response time. People can use pre-defined semantics to write business policies, the engine accepts data input, reasons through its rules, and make comprehensive decisions.

As we have discussed in Section II, the application selection/composition decision also falls into the category of business policies, and those decisions may need to be changed under different situations, according to external information. It can be achieved by applying rule engine technologies.

We would first discuss the detail requirements of flexible business policies in application composition with examples. Then we would explain the rule-based approach to satisfy those requirements.

For any initial request received, the goal of application composition is first to derive a set of ordered applications to invoke, then to decide in what order should the applications be invoked, also consider the need to cut some application in the chain or append some more applications if necessary, according to subscriber preferences and operator needs.

We base our discussion on the assumption that the initial set of applications that should be triggered is known to application composer. This is because current service triggering mechanisms (S-CSCF, SIP Application Server, etc.) provide a lot of means to determine an initial set of services for a user, for example, using user service profile, using protocol headers in the initial message. This paper focuses on adjusting the invocation order and application removal/appending based on business conditions, as discussed below:

1. Management of invocation order of selected applications.

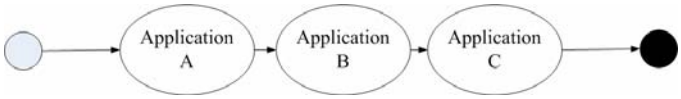

Figure 2. Order of Applications in Composition

When the list of selected applications is known, the order to invoke them is very important. Inappropriate order between multiple invoked applications may result in malfunctioning of applications and bad user experience.

For example, if a subscriber has two services which are SpeedDial and OutgoingCallBarring, when the subscriber dial an abbreviated number, different order of the two applications may bring different execution result. If SpeedDial Service is triggered before OutgoingCallBarring, it can replace the callee number with real callee number, so when afterwards OutgoingCallBarring service is triggered, the service can function as expected. However, if OutgoingCallBarring is triggered before SpeedDial, then OutgoingCallBarring service would have no way to decide whether to let the call passes.

Another example would be the interaction between CallForwarding service and OutgoingCallBarring service. Obviously, if CallForwarding service is triggered after OutgoingCallBarring service, it could redirect the call to a callee that OutgoingCallBarring wants to block, while OutgoingCallBarring service would have no chance to stop the call.

The problem can be solved by categorizing applications into different groups. For example, one group of applications would change the callee number, and another group of applications would be affected by any changes to callee number. So when selecting applications, add a rule that put the first group of applications ahead of second one can help a lot.

2. Insertion / removal of applications under certain conditions

Application composition may not always be the same for a subscriber request, and it can be influenced by many external conditions. The composition entity should enable subscriber or operator to set their conditions of triggering applications.

From subscriber's point of view, it can be customizing the way how to compose his/her own set of services under different scenarios. Figure 3 is an example of such customization.

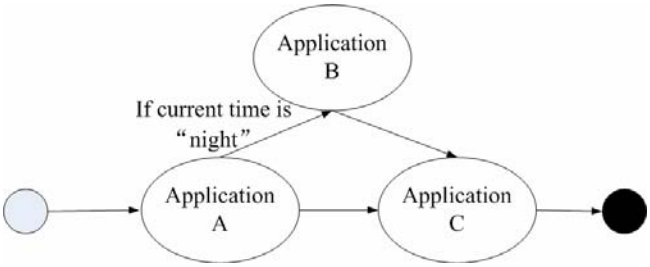

Figure 3. Subscriber Customization of Applications Composition

If a subscriber has 3 services (which is A, B and C), he/she might want to set some rules/policies to control the condition of service triggering. In this example, the subscriber has a customization rule that application B would only be triggered when the current time is night.

When subscribers have many subscribed services to customize, those services may come from different application developers, and when subscribers might need to change their preferences on application composition frequently, it is not easy for subscriber to access every different service application provider's portal or other channels to customize his/her service chain. A better choice would be to access telecom operator's unified interface and to customize subscribed services in a unified rule-based way.

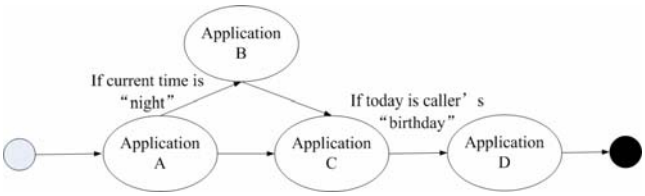

Figure 4. Operator Appending Application Applications Composition

Another situation that application composition should be swayed by various conditions is when telecom operators want to add some logic to enhance or transform subscriber's service chain, for better user experience. They can also do it by configuring rules. For example, in Figure 4, based on Figure 3's customized service chain, the operator may want to add a ring back tone application to the subscriber's application chain so as to play a "happy birthday" song as ring back tone for the caller, only when today is caller's birthday. It is like a little present from telecom operator to caller, and supposed to bring better user experience.

Now after examining the requirements of business policies in application composition, it is clear that certain mechanism is needed to support the fast-changing rules that may affect application composition. Figure 5 shows the proposed rule-based application composition framework.

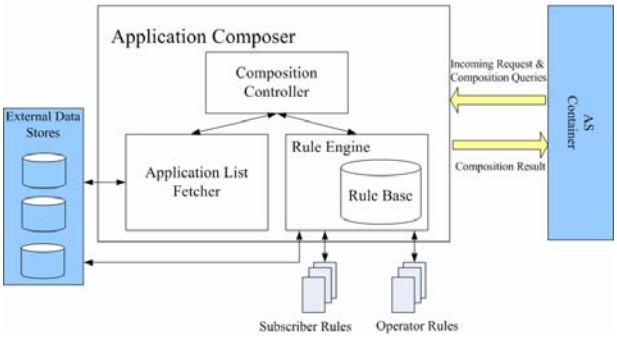

Figure 5. Rule-based Application Composition Framework

In Figure 5, there are 3 categories of entities that form the external environment of Application Composer:

a) AS Container. This is the container where Application Composer resides in. According to JSR289 application composition paradigm, the container queries Application Composer on a interface to decide how to compose applications. On this interface, the received (or processed by some applications) SIP request is sent to Application Composer, with some context information that would help Application Composer to do its job, for example, the role of subscriber as a caller or callee in this session, and the composition result (what is the next application name to trigger) is returned back to Container.

b) Rules from subscribers and operators. These are rules that defines the preferences/requirements of application composition from subscribers who manage their own service chains or from operators who want to control their service offerings. These rules and update of latest rules may come at any time, and Application Composer should hot-deploy and update its rules repository to make them effective almost instantly.

c) External Data Stores. These data stores can be any meaningful data store that is of interest to the composition process, and can be accessed by Application Composer. For example, the data store of user profile, used to decide the initial set of services that should be triggered. Other examples would be user preferences, application server status (high/low load), etc.

Inside Application Composer, there are 3 main parts: Application List Fetcher is responsible to provide the initial set of applications that should be triggered for an incoming request. It either talks to external data stores (to get user profile on subscribed services) or derive the application list from incoming request itself. Then Application List Fetcher supplies the set of applications as initial input to Rule Engine, which maintains and applies all related rules to adjust application invocation order, and to add/remove some applications if necessary, and finally to get an applicable application name list. The Controller manages the whole process, from accepting incoming request and context information to managing and returning application composition result to AS container.

III. EXAMPLE IMPLEMENTATION & ANALYSIS

In this section we illustrate an example implementation, based the rule engine of DROOLS [6], and a JSR289 SIP application server container. We also analyzed some related issues under this environment.

A. *Implementation of Rule-basd AC Framework*

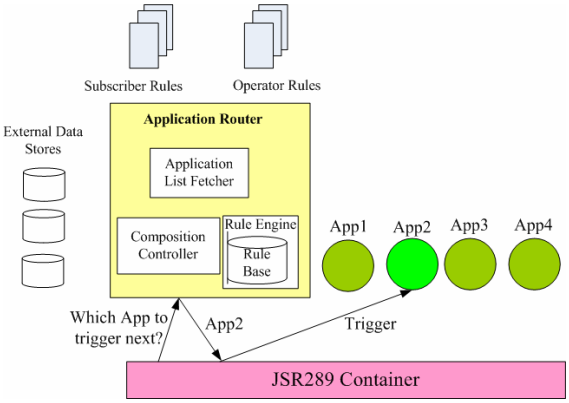

Figure 6. Implementation of JSR 289 Application Router

Figure 6 shows the example implementation of JSR 289 Application Router, which is the embodiment of rule-based application composition framework previously discussed. The interface between JSR289 container and application router is *getNextApplication* method defined in JSR 289 specification, which is called by container when a SIP Servlet sends or proxies an initial SIP request. The core to this Application Router is its Rule Engine component, which decides the application composition result.

We used the eclipse plug-in of DROOLS rule engine to design example rules. DROOLS provides the "Domain Specific Language" support to rule authors, so the business oriented rules can be expressed by natural language. The author just needs to define the set of sentences that can appear in a rule, and specify the corresponding DROOLS "native"rule language. Then the natural language rules can be used, even by business person who are familiar with business requirements but not so familiar with rule engine technologies. Following are some application composition rules written in natural domain specific language:

```
rule "<Operator> Invocation Order"

    when
        For any initial application name list
        If "Callee Sensitive" apps appear before "Callee Manipulative" apps
    then
        Put "Callee Manipulative" apps "ahead" of "Callee Sensitive" apps
end
```

Figure 7. (a) Operator rule of invocation order between 2 groups of apps

```
rule "<Subscribder> Composition Condition"

    when
        For any initial request from subscriber "Jack.Jones.85789263900153"
        If initial application name list includes app "Call Forwarding"
        If current time is "day"
    then
        Remove app "Call Forwarding" from application name list
end
```

Figure 7. (b) Subscriber rule of composition condiction of an app

```
rule "<Operator> Birthday Ringback Tone"

    when
        For any initial "call" request from caller subscriber
        If current date is "birthday of the caller subscriber"
        If current application name list has no app "Ringback Tone"
    then
        Add app "Birthday Ringback Tone" to current application name list
end
```

Figure 7. (c) Opeartor rule of adding birthday song to app list

In Figure 7, 3 example rules are listed. The rule in (a) is an operator rule that declares all "callee sensitive" applications, e.g. OutgoingCallBarring, should be moved to positions after "callee manipulative" applications, e.g. SpeedDial, in the application composition name list. The rule in (b) is from subscriber Jack Jones specifying his preference on CallForwarding service invocation. The rule in (c) is again an operator rule which inserts birthday song ring back tone to a subscriber's application list.

It is noted that all the rules can be reset online and be effective instantly.

With those rules, operators can transform their service portfolio to better serve subscribers, also avoid possible confliction between different services; subscribers can set their preferences, short term or long term, upon how all of their subscribed services behave.

B. *Some issues explained*

1) Inter-AS Composition: We have discussed around application composition in one SIP AS, where multiple SIP applications are deployed. However, in real network deployments, for example IMS network, it is common that many SIP application servers be triggered consequtively, each providing some services to a subscriber. Therefore, to achieve a unified application composition, inter-AS application composition is an important issue. The ideal mechanism seems to be let one AS to do the application composition in a logical domain consisted by multiple SIP application servers. The composing AS knows the application topology in this logical main and controls the application triggering process from beginning to end. Unfortunately, current SIP Servlet infrastructure still can't support such master-slave pattern of SIP AS control. For example, the JSR289 specification does not allow one AS to control over another AS's decision on next triggered application. So currently we focus the rule-based approach inside one SIP AS.

2) Transactional Application Composition: Transactional Application Composition may be the next level of composing different services. It means to dynamically adjust the next application based on the execution result of previous applications. For example, if application A is triggered, and executes successfully, then application B is triggered; but if application A is failed, then application C should be triggered. It can be imagined that such composition is more useful if more complex and fine-grained arrangement among applications is needed. However, no effective means is availible for application composer to sense the status of an application execution, unless through some native extension of

SIP headers. Meanwhile, how to judge the execution result of an application is still an open question. The simple "success/fail" information may not always be enough. Therefore, in this paper, we just focus on adjusting the order / elements of initial application name list, and assume that in following interactions within the same session the application list will be executed blindly, without beging changed anymore, unless some application terminates the session. This has leaves some uncertainties to the composed application execution.

3) Confliction Between Operators and Subscribers: There could be cases when operators and subscribers have different opinions on how applications should be composed, and this may reflect to rule conflicts in application composer. Coordination and priority mechanism is needed in this system.

IV. Related Work

Some pioneering work has been carried out in this area of SIP application composition framework. The expert group members of JSR 289 specification introduced the design pattern of JSR 289 application composition in detail in [7]. But they did not consider the requirement of flexible business policies with SIP application composition. With the use of SIP in large deployment of IMS, the importance of business flexibility for both subscribers and operators is enormous.

V. Conclusions

In this paper we analyzed the SIP application composition problem, especially on the need of flexible business policies. We found that current SIP application service framework has not introduced supporting mechanism for such important requirement. Based on rule engine technology and latest JSR 289 specification of SIP Servlet API, we proposed a new rule-based application composition framework and give an implementation, with example rules. Future works may include practical deployment of the framework, and further investigation on advanced issues around this area, like discussed in Section III.

References

[1] J. Rosenberg et al, "SIP: Session Initiation Protocol", IETF RFC3261, 2002.

[2] 3GPP IP Multimedia Subsystem (IMS). http://www.3gpp.org/.

[3] Dirk O. Keck and Paul J. Kuehn, "The Feature and Service Interaction Problem in Telecommunications Systems: A Survey", IEEE Transactions on Software Engineering, VOL. 24, NO. 10, OCTOBER 1998

[4] JSR 116, http://jcp.org/en/jsr/detail?id=116

[5] JSR 289. http://jcp.org/en/jsr/detail?id=289

[6] JBOSS Rules 3.0.4, http://labs.jboss.com/portal/jbossrules/docs

[7] E.Cheung and K.H.Purdy, "Application Composition in the SIP Servlet Environment", IEEE International Conference on Communications, June 2007