A Standards Based Software Environment for Providing SIP Application Composition

K. Hal Purdy
khp@research.att.com
Eric Cheung
cheung@research.att.com
What are applications?

- Incremental or additional functionality
  - Interjected between participants (humans) engaged in real-time communication
    - The focus is not on transactional human-machine interactions
    - Applications *mediate* communication between people and are not “sources” or “sinks” of communication themselves
  - Core, Run-time logic
    - Although important, we are not considering here the OSS/BSS, billing, provisioning, work-flow aspects of an application
  - Not basic connectivity, transport or dialtone services
  - Roughly synonymous with ‘features’ in PSTN world or ‘advanced features’

- Emphasis on Network Resident, Application Servers

- Real-time voice component
  - e.g. telecommunications oriented applications
  - video and text are considered in context of multimedia

- Examples:
  - Call Forwarding, Call Screening, Do Not Disturb
  - Call Waiting, Mid-Call Move
  - Record Voice Mail, Conferencing

- Independent, self-contained communicating only with SIP messages
  - e.g., applications developed with ECharts for SIP Servlets (www.echarts.org)
Composition

- Applications are simultaneously active during the same communications episode
- Emphasis on run-time composition
  - Invoked services may change on call by call basis
- Because applications are independent and self-contained,
  - Composition results in overall system behavior arising from cooperation of peers
  - Example:
    - Application 1: Call Forwarding on Unavailable
    - Application 2: Do Not Disturb
  - Both applications may run singly. When run together, overall behavior is that of the composition of the two applications
- Different from composition where a higher level application makes use of a lower level enabling service
  - Example:
    - Application: Call Forwarding
      - Enabling “Service”: Presence Engine
  - Call Forwarding builds upon service provided by Presence Engine and is said to be “composed” with it
Need for Application Composition

- Real-world VoIP services will be made up of multiple applications
  - Applications come from different development teams or third parties
    - e.g. combining Consumer VoIP and Business VoIP features into a single offer
  - Off-the-shelf applications e.g. Voice Mail
  - Each application deployed, upgraded and retired at different time

- Individual applications should be modularized
  - Aids understandability, encourages re-use

- Industry Recognition of Need
  - From vendor’s IMS SCIM Presentation:
    - Enabling coherent combination of applications (within each Service Delivery Platform and across Service Delivery Platforms) without having to change them is the primary goal for providing a targeted, personalized, and end-user-efficient communication experience.

- Unanimous support for application composition in new Java Community Process (JCP) SIP Servlet 1.1 standard
Architectural Underpinnings - Distributed Feature Composition (DFC)

- Pipe and filter software architecture applied to telecommunications to describe a virtual PSTN telecommunications switch
  - Feature boxes are independent software entities “connected” by pipes that provide communications channels between them
    - Across channels travel a restricted set of well understood signals
  - Order in which features appear defined by partial precedence rules

- Addresses and Subscriptions
  - Subscription: Mapping from subscriber address to features enabling DFC system to invoke correct set of features for a subscriber
  - Addresses
    - simple unstructured strings
    - come in many flavors: identifiers of devices, roles, groups, individuals

- Regions
  - Logical partitioning of features into two regions representing callers (source) and callees (target)
  - Common division
    - In IMS, these ‘regions’ are called ‘originating’ and ‘terminating’
    - AIN in PSTN has similar notion
DFC Overview

Three main aspects of DFC architecture:

1. DFC Router – Invokes features in chains to serve subscribers

2. DFC Protocol – Definition of signals and rules to determine what may pass between feature boxes.

3. DFC Data Model – Data partitioning scheme to address the effect of data sharing on feature interaction
Applicability of DFC to VoIP (SIP, SIP Servlet Environment)

- DFC Protocol supplanted with SIP messages/protocol
  - At first, we translated between SIP and DFC protocol
  - Converting to “SIP only” signaling has required hard work
    - Mostly successful in accruing the feature interaction advantages of the DFC protocol within confines of SIP
  - Not the focus of this tutorial

- DFC Data Model
  - Least developed part of DFC
  - Hard to apply with real world application deployments
    - Where data models are often already largely determined

- DFC Router
  - Useful mechanism for building application (feature) chains
    - Clear distinction in DFC between feature capability and router capability
    - Router algorithm stateless and largely independent of message protocol used between features
    - Our main contribution to the SIP Servlet environment
SIP Servlet API

- Programming SIP applications using servlet model
  - Familiar to HTTP servlet programmers
  - Container provides value-adds such as protocol stack, state management, high availability, load balancing, etc.
- Java Community Process (JCP) standard
  - Version 1.0 (JSR116) released in March 2003
  - As of 7/2007 at least five commercial and one open source implementations
- Native SIP application server in 3GPP IP Multimedia Subsystem (IMS)
- Environment matches well with DFC
  - Employs pipe and filter architecture (Cascaded Services Model)
    - Features (filters) == SIP Servlet Applications
    - Featureless Calls (pipes) == SIP signaling paths
- Increasingly popular SIP application environment
SIP Servlet 1.0 Applications

- A SIP servlet application is made up of
  - Servlets
  - Deployment descriptor (sip.xml) contains
    - Triggering rules that are predicates over SIP requests
    - Triggering rules indicate servlets to be invoked
SIP Servlet 1.0 – Application Triggering

- Application selected by container
  - Determined by matching rule in deployment descriptor (DD)
  - What happens when rules match from multiple applications?
  - Which applications to pick, how many, and in what order?

Rules from both DDs match. What to do?

- DD
  - App 1
- DD
  - App 2
SIP Servlet 1.0 and Application Composition

- SIP Servlet API 1.0 spec (JSR116) states that application composition is:
  - Desirable
  - Optional
    - An implementation is compliant, even if it only ever invokes one application
- No standard composition mechanism defined
  - Left entirely to container implementation
- Leads to interoperability issues
  - Not just in theory
  - Three container implementations resulted in three interpretations of how composition should be supported
SIP Servlet 1.0 - JSR 116 – Further Issues

• “Deployer” of services most needs control over application invocation
  • Role mentioned in JSR116 but no formal mechanism of control
  • Application packaging (sar files) makes this problem worse
    - Deployer cannot easily change triggering rules

• SIP Servlet applications need (limited) influence over application selection
  • No clear mechanism available in JSR116

• No distinction between callee and caller services
  • DFC has source and target regions
  • IMS has originating and terminating ‘session case’
  • Without this, reuse is discouraged
SIP Servlet 1.0 - JSR 116 – Further Issues

• Applications need to know the subscriber on whose behalf they are invoked
  • Due to variety and structure of SIP headers in a message and different interpretations of those headers in deployment architectures (e.g. IMS), such subscriber information
    - is not always clear
    - may change from deployment to deployment
  • Example: originating subscriber can be identified by From header value or by P-Asserted-Identity or P-Preferred-Identity header values or some combination
SIP Servlet 1.0 - JSR 116 – Further Issues

• Applications need to know whether invoked for caller or callee
  • Extension of idea that distinction between the two needs to be made
  • Example:
    - Originating Call Screening (OCS) application
      • Used as an example application in SIP Servlet API 1.0 specification
    - Could be easily generalized to be a general call blocking/screening (CBS) application
      • On originating side, it blocks calls that caller is not allowed to make
      • On terminating side, it blocks or redirects undesirable calls to the callee
Call Screening/Blocking – Need for application knowledge of region

CBS logic can be used for both callers and callees if it can simply determine which side of the call it’s on.

- CBS logic can be used for both callers and callees if it can simply determine which side of the call it’s on.

**Diagram:**

- **Originating Region**
  - **Logic:** “I’m on the calling side so I examine incoming INVITE for allowable Request-URI”

- **Terminating Region**
  - **Logic:** “I’m on the called side so I examine incoming INVITE for allowable From header value”
Summary and Chronology

• Mid-2004
  • Increasingly clear to us that
    - DFC technology is broadly applicable in VoIP realm to address application composition
      - We’ve done it in production with AT&T CallVantage service
    - However, to have maximum impact, the DFC ideas must be standardized
    - SIP Servlet 1.0 is:
      • Popular programming model
      • Is very much in need of an application composition solution
      • The JCP is the right standards body for a codification of these ideas

• Fall 2004
  • We start making “pitches” to vendors looking for support for a revised SIP Servlet specification
    - In particular, a JCP member who could serve as the specification lead
    - A process commences where we present our ideas to SIP Servlet container vendors (and anyone else who will listen)
Summary and Chronology

• Early 2005
  • Recognition that BEA shares AT&T’s goals for support of application composition in SIP Servlet environment

• Mid 2005
  • BEA agrees in principle to be the specification lead on a new JSR expert group to revise the 1.0 specification

• Remainder of 2005
  • Politics, Legal Wrangling

• January 2006
  • JSR289 Expert Group (finally) formed (http://www.jcp.org/en/jsr/detail?id=289)

• 2006
  • Politics, Technical Wrangling

• January 2007
  • Early Draft Review version released
  • Download EDR draft from: http://jcp.org/aboutJava/communityprocess/edr/jsr289/index.html

• July 2007 – All comments public and private compiled together for expert group resolution
JCP JSR 289 Expert Group

• Large expert group - 34 members, representing 24 companies:
  • 8x8, Apache, Appium, AT&T, BayPackets, BEA, Cingular, Cisco, Ericsson, Fokus, Fujitsu, IBM, JBoss, Mailvision, Netcentrex, Nexcom, Oracle, Orange, Sun, T-Mobile, Telcordia, Telecom Italia, Ubiquity, Voztelecom
  • Main contributions from 8x8, AT&T, BEA, Ericsson, IBM, Ubiquity

• Main themes:
  • Application composition
  • Convergence (SIP, HTTP, other protocols)
  • Improvements, bug fixes
Overview of Solution

- Formalize role of deployer, separate concerns of deployer, application developer, and container developer
- Introduce application router
- Introduce subscriber identity and routing region
- Introduce routing directives
- External request routing
Separation of Concerns

- **Deployer (e.g. service providers such as AT&T)**
  - Responsible for overall service in a communication episode, possibly made up of multiple applications
  - Controls application selection and invocation order

- **Application Developer**
  - Implements individual application logic
  - Makes minimal assumption about the presence, location and behavior of other applications

- **Container Developer (e.g. Vendors such as Avaya, BEA, IBM)**
  - Manages lifecycle of applications
  - Provides SIP protocol stack
  - Provides OA&M, performance, reliability, high availability, etc
Solution Part 1 - Application Router

- In SIP Servlet 1.0, Request matching based on Deployment Descriptor (DD) packaged inside an application
Solution Part 1 - Application Router

- New logical component, **Application Router (AR)**, introduced in 1.1. AR performs application selection.
Solution Part 1 - Application Router

- AR performs application selection
  - API interface with container (no direct interface with applications)
  - Container queries AR to obtain the name of application to service an initial request

- AR is controlled by the deployer
  - How AR performs selection is up to deployer
    - Can make use of parts of the request, proprietary information such as user profile, time-of-day, network condition, etc
  - Different AR implementations may be used
Application Router does not implement application logic

- AR does not stay in signaling path
  - Only invoked on initial requests
  - Does not see responses or mid-dialog requests
- AR cannot modify the initial request
  - Exception: can affect Route headers indirectly
- AR cannot respond to the initial request
Solution Part 2 - Information Provided to Applications

- **Invocation regions**
  - An application operates in one of 3 region types: *originating*, *neutral*, *terminating*
  - Deployer can define new region, but must be one of the 3 types

- **Subscriber Identity**
  - An application invoked to serve a subscriber
  - Not always obvious from SIP request headers

- AR determines region and subscriber identity
- Information made available to invoked application.
Solution Part 3 - Application Intention

- **Routing directives** – for application to indicate its intention to AR explicitly

- **NEW:** “This initial request is unrelated to any previous requests.”

- **CONTINUE:** “I received a request, and now I am sending a request in the same direction.”

- **REVERSE:** “I received a request, and now I am sending a request in the opposite direction.”

- AR can use this information to influence its selection of the next application.
getNextApplication(SipServletRequest initialRequest,
                SipApplicationRoutingRegion region,
                SipApplicationRoutingDirective directive,
               Serializable stateInfo);

Returns:

String nextApplicationName
String subscriberURI
SipApplicationRoutingRegion region
Serializable stateInfo
Example 1

This example illustrates interaction between container, AR, and applications. It also illustrates NEW vs CONTINUE directives.

- Alice subscribes to speed dial. Alice dials a speed dial code to call Bob.
- Bob has location service that re-directs calls to his temporary location
- i.e. 1 originating application for Alice, 1 terminating application for Bob.
Example 1

Step 1: Container receives INVITE, queries AR

Initial INVITE

getNextApplication(req1, null, NEW, null)

Returns app="Speed Dial",
sub="sip:alice@…",
reg=ORIGINATING,
state="SD selected"

Application Router

req1

Container
Example 1

Step 2: Container invokes Speed Dial application

Initial INVITE req1

May call:
- SipSession.getSubscriberURI(),
- SipSession.getRegion()

Container
Example 1

Step 3: SD proxies request, container queries AR

Initial INVITE

req1

Speed Dial

proxyTo("sip:bob@…")

req2

getNextApplication (req2, ORIGINATING, CONTINUE, “SD selected”)

Application Router

Returns app="Location Service", sub="sip:bob@…", reg=TERMINATING, state="LS selected"

Container
Example 1

Step 4: Container invokes Location Service app

Initial INVITE

req1

Speed Dial

req2

proxyTo("sip:bob@

Location Service

Container
Example 1

Step 5: LS proxies request, container queries AR

- Initial INVITE
- req1
- Speed Dial
- req2
- Location Service
- proxyTo("sip:device1@

Returns app=null

ggetNextApplication (req3, TERMINATING, CONTINUE, "LS selected")
Example 1

Step 6: Container sends request to Bob’s device

1. Initial INVITE
2. Speed Dial
3. Location Service
4. Application Router
5. Outgoing INVITE
6. Container

req1 → req2 → req3
DFC Application Router

- Example AR implementation, open source implementation. (www.echarts.org)
- Two regions - originating and terminating
- Application selection based entirely on subscription by addresses
- Subscriber addresses are derived from SIP headers:
  - Originating: From header
  - Terminating: Request-URI
- Subscription and precedence relationship are specified in XML file
DFC Application Router XML file

```xml
<sip-app-router>
  <application-mapping>
    <originating-region-mapping>
      <mapping>
        <address-pattern>(*sip:alice@.*)</address-pattern>
        <app-name>SpeedDial</app-name>
      </mapping>
    </originating-region-mapping>
    <terminating-region-mapping>
      <mapping>
        <address-pattern>sip:bob@.*</address-pattern>
        <app-name>LocationService</app-name>
      </mapping>
    </terminating-region-mapping>
  </application-mapping>
</sip-app-router>
```
Example 2 - Precedence

```xml
<sip-app-router>
  <application-mapping>
    <originating-region-mapping>
      <mapping>
        <address-pattern>*sip:*</address-pattern>
        <app-name>CW</app-name>
        <app-name>3WC</app-name>
      </mapping>
      <mapping>
        <address-pattern>*sip:alice@.*</address-pattern>
        <app-name>OCS</app-name>
      </mapping>
    </originating-region-mapping>
    <terminating-region-mapping>
      <mapping>
        <address-pattern>sip:*</address-pattern>
        <app-name>CF</app-name>
        <app-name>ICL</app-name>
        <app-name>CW</app-name>
        <app-name>3WC</app-name>
      </mapping>
      <terminating-region-mapping>
        </application-mapping>
  </application-mapping>
  <precedence>
    <originating-region>
      <ordering>
        <app-name>CW</app-name>
        <app-name>3WC</app-name>
        <app-name>OCS</app-name>
      </ordering>
      <terminating-region>
        <ordering>
          <app-name>CF</app-name>
          <app-name>ICL</app-name>
          <app-name>CW</app-name>
          <app-name>3WC</app-name>
        </ordering>
      </terminating-region>
    </precedence>
  </precedence>
</sip-app-router>
```
Example 2 - Precedence

```xml
<sip-app-router>
  <application-mapping>
    <originating-region-mapping>
      <mapping>
        <address-pattern>*sip:*</address-pattern>
        <app-name>CW</app-name>
        <app-name>3WC</app-name>
      </mapping>
      <mapping>
        <address-pattern>*sip:alice@*</address-pattern>
        <app-name>OCS</app-name>
      </mapping>
    </originating-region-mapping>
    <terminating-region-mapping>
      <mapping>
        <address-pattern>sip:*</address-pattern>
        <app-name>ICL</app-name>
        <app-name>CW</app-name>
        <app-name>3WC</app-name>
      </mapping>
      <mapping>
        <app-name>CF</app-name>
        <app-name>ICL</app-name>
      </mapping>
    </terminating-region-mapping>
  </application-mapping>
</sip-app-router>

INVITE ...
...
From: sip:alice@...

CW → 3WC → OCS

diagram
Example 2 - Precedence

```
<application-mapping>
  <originating-region-mapping>
    <terminating-region-mapping>
      <address-pattern> sip:* </address-pattern>
      <app-name> ICL </app-name>
      <app-name> CF </app-name>
      <app-name> 3WC </app-name>
      <app-name> CW </app-name>
    </terminating-region-mapping>
  </originating-region-mapping>
</application-mapping>
```

```xml
<terminating-region>
  <ordering>
    <app-name> CW </app-name>
    <app-name> 3WC </app-name>
    <app-name> OWS </app-name>
  </ordering>
</terminating-region>
```

```xml
<terminating-region>
  <ordering>
    <app-name> CF </app-name>
    <app-name> ICL </app-name>
  </ordering>
</terminating-region>
```

```
Example 2 - Precedence
```
Example 2 - Precedence

INVITE sip:bob@
...
From: sip:alice@
...

originating

terminating
Carol called Alice and in call.
Alice activates 3WC to add Bob to the call.
Example 3 – Reverse Directive

Carol Alice

INVITE sip:bob@
...
From: sip:alice@
...

?
Example 3 – Reverse Directive

INVITE sip:bob@

From: sip:alice@

... CW invoked. Will probably break.
Example 3 – Reverse Directive

Carol

INVITE sip:bob@…
…
From: sip:alice@…
…

Alice’s OCS missing.

CONTINUE
**Example 3 – Reverse Directive**

Carol calls Alice

INVITE sip:bob@...
... From: sip:alice@
...

Alice calls Bob
Example 3 – Reverse Directive

INVITE sip:tom@server.com
...
From: alice@server.com
...

Carol omega

Alice omega

Bob omega

Example 3 – Reverse Directive

INVITE sip:tom@server.com
...
From: alice@server.com
...

Carol omega

Alice omega

Bob omega
Solution Part 4 - External Routing

- Previous examples show AR returns name of locally deployed applications

- Alternatively, AR can return an external route:
  - Route to a SIP application server
  - If it is also a SIP servlet container
    - Can pass selection state information to the other AR

- Route back option

- In both cases, the container populates the Route header of the outgoing request.
Solution Part 4 - External Routing

INVITE sip:....
...
Route: sip:abc@example.com
...

Returns route="sip:appX@as2.example.com"

Container AS1

Application
Router

AS2
Solution Part 4 - External Routing

INVITE sip:....
... Route: sip:abc@example.com
...

Application Router

Container AS1

Returns route="sip:appX@as2.example.com"
ROUTE_BACK

INVITE sip:....
...
Route: sip:appX@as2.example.com
Route: sip:as1.example.com
Route: sip:abc@example.com
...

AS2
Supported Application Server Architectures

• With advent of SIP Servlet 1.1 API what application architectures might be deployed?
  • In IMS deployments
  • In enterprise or non-IMS deployments?

• This section is more speculative
  • So jump in and give us your ideas!
Architecture #1

• Simple Deployment
  • Single Application Server provides service to calling and called subscribers
  • Useful in some enterprise environments (IP-PBX)
Architecture #2

• Basic IMS Deployment
  • S-CSCF invokes Application Servers for originating and terminating regions using IMS mechanism (route headers)
  • ARs on Application Servers invoke appropriate applications using JSR289 mechanism
Architecture #3

- IMS Deployment with multiple Application Servers in a region
  - S-CSCF invokes multiple Application Servers for originating and terminating regions using IMS mechanism (route headers)
  - ARs on Application Servers invoke appropriate applications using JSR289 mechanism
  - Still standard IMS

![Diagram of Architecture #3]
Architecture #4

- IMS Deployment with chain of Application Servers
  - S-CSCF invokes a single Application Server for originating region using IMS mechanism (route headers)
  - AR on “Orig AS 1”:
    - Invokes appropriate applications using JSR289 mechanism
    - Uses external routing to delegate part of originating region to “Orig AS 2” (optionally passing selection state information to Orig AS 2)
- Orig AS 1 assumes some responsibility for AS routing
- Orig AS 1 and Orig AS2 may now cooperate more closely
Architecture #5

- IMS Deployment with hierarchy of Application Servers
  - S-CSCF invokes a single Application Server for originating region using IMS mechanism (route headers)
  - ARs on “Orig AS 1”:
    - Invokes its own internal applications
    - Uses ROUTE_BACK mechanism to visit Orig AS 2 and then get the request back
    - Does the same to visit Orig AS 3 and come back
    - Finally, request is returned to S-CSCF based on its pushed route header
  - Thus, Orig AS 1 plays coordinating role among other application servers (2 and 3)
IMS Service Capability Interaction Manager (SCIM)

• Orchestrates service delivery among application server platforms within the IMS architecture

• From IMS protocol perspective, appears as:
  • Application Server to S-CSCF
  • S-CSCF to Application Servers

• Very little text in the 3GPP specifications describing its actual function

• Intended as placeholder for vendor defined solution to problem of feature interaction management ("orchestration")
Architecture #6

- IMS Deployment with SCIM
  - S-CSCF invokes a single Application Server (SCIM) for originating region using IMS mechanism (route headers)
  - The SCIM:
    - Invokes its own internal applications
    - Pushes two route headers to visit Orig AS 2 and then get the request back
    - Pushes another two route headers to visit Orig AS 3 and come back
    - Finally, request is returned to S-CSCF based on its pushed route header
Architecture #7

- IMS Deployment with SIP Servlet container as SCIM
  - S-CSCF invokes a single Application Server (SCIM) for originating region using IMS mechanism (route headers)
  - The SCIM:
    - AR operates solely to coordinate other Application Servers
    - Pushes two route headers to visit Orig AS 2 and then get the request back
    - Pushes another two route headers to visit Orig AS 3 and come back
    - Finally, request is returned to S-CSCF based on its pushed route header
  - With Application Router technology, SCIM can be built using standard software environment: SIP Servlet API
Future Issues

• What application router implementations will prevail?
  • Will application routers be used to compose fine grained applications?
  • While S-CSCF/SCIM will be used for coarser grained composition of application suites?
  • Will SIP Servlet based SCIM implementations occur?

• What application server architectures will prevail?
  • Is there a need for an external SCIM component?
  • If so, will its functionality become better defined?