

# Service composition in IMS using Java EE SIP servlet containers

Torsten Dinsing, Göran AP Eriksson, Ioannis Fikouras, Kristoffer Gronowski, Roman Levenshteyn, Per Pettersson and Patrik Wiss

The IP Multimedia Subsystem (IMS) gives operators, service providers and developers the ability to create new services using standard session initiation protocol (SIP) routing. To further enhance and complement IMS service creation, Ericsson Research has developed a composition engine that uses core IMS mechanisms, Java EE SIP servlet containers, and service composition technologies.

The authors describe an approach for composing new services using the composition engine and functionality exposed by, for example, SIP applications in a Java EE application server, a service delivery platform (SDP), or standardized IMS enablers.

## Introduction

In consumer-driven markets, stiff competition forces operators to complement standard mass-market services with personalized consumer offerings. And because consumer interest is always shifting to new areas, operators should consider increasing their capability to offer a broad range of personalized services rather than invest heavily in one specific service. Therefore efficient service creation becomes a critical factor for system-integration projects that implement individualized consumer offerings. Time to market (TTM) is also increasingly important.

## SIP servlet container in IMS

The SIP servlet container, defined in a draft version of JSR289, contains and manages SIP applications and provides access to session initiation protocol (SIP) mechanisms via a Java API.

Java Platform Enterprise Edition (Java EE) is a scalable middleware platform used in the telecom domain. A Java EE application server (AS) is the platform on which the SIP servlet container is deployed. The AS provides network services over which SIP requests and responses are sent and received.

An AS in IMS is connected to the serving call session control function (CSCF) via the IMS server control (ISC) interface. Initiating SIP requests, received by the AS from the CSCF, are passed on to the container. The container identifies the SIP application in question by querying an entity called the application router (AR). The container then dispatches the request to the selected SIP application. Provided the SIP application does not terminate the request, the container queries the AR again for the next SIP application to invoke (Figure 1).

By pushing a route onto the SIP route header (in much the same way as the CSCF routes to an IMS application server), the application router may also instruct the container to route the request to a SIP application deployed on another server.

JSR 289 explicitly avoids specifying the inner working of an application router (AR). In the sections below, we discuss a flexible and dynamic composition engine implementation.

## Composition engine for SIP services

Researchers at Ericsson have defined and implemented a service composition engine that uses the AR interface to provide the con-

tainer with SIP routing decisions at runtime – dynamic SIP routing. The engine takes into account data derived from

- the state maintained by the composition engine;
- SIP signaling;
- formal descriptions of SIP services exposed by SIP entities;
- data from external entities queried at runtime via SIP or other technologies such as web services; and
- Java virtual machine (JVM) data, such as time and current load.

Ericsson's composition engine may be seen as a programmable state machine that aggregates constituent SIP services (for instance, SIP applications in the container) to create rich, new composite services.

The implemented algorithm is event-based and data-driven as opposed to the process-driven approach used in WB-BPEL where process activities are primary language constructs and events are supported only implicitly. An event-based model that supports sessions correlates directly to signaling in call control and is more natural and flexible for composing real-time communications.<sup>5</sup> Therefore, existing process-driven and web-services-specific technologies, such as WS-BPEL, are not suitable for composing SIP services.

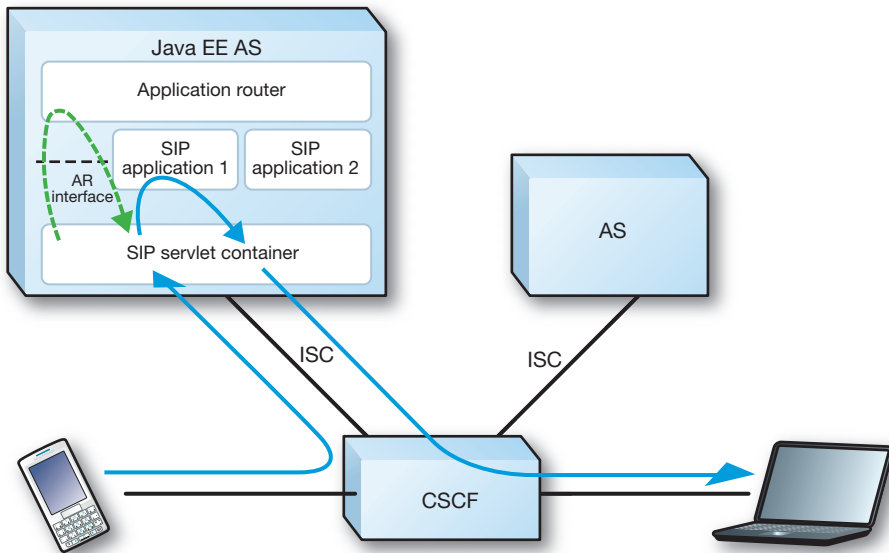
The algorithm, which focuses on service properties and the constraints that govern them, uses data to dynamically set up and adapt sessions according to events and the state of the execution environment. Constraints describe service interactions and dependencies.

The descriptions of constituent services are stored in a service database that exposes them to the composition engine. Using composition templates and constraints, the composition algorithm searches the database for the next constituent service to be added to a composite service. This approach, in which a specific constituent service is selected at runtime, is called late binding. The method supports loose coupling, which makes composite services more adaptable to changes of constituent services.

The algorithm enables the construction of a composite service by adding constituent services, piece by piece (where each piece satisfies all constraints), to a session that is being established. This approach makes it possible to manage feature interaction, provided the features in question have been foreseen in the model of the SIP services and their relationships.

## TERMS AND ABBREVIATIONS

API	Application programming interface	Java EE	Java Platform Enterprise Edition
AR	Application router	JSR	Java specification request
AS	Application server	JVM	Java virtual machine
CSCF	Call session control function	OMA	Open Mobile Alliance
e2e	End to end	PNA	Presence network agent
IDE	Integrated development environment	SDP	Service delivery platform
iFC	Initial filter criteria	SIP	Session initiation protocol
IMS	IP Multimedia Subsystem	TTM	Time to market
ISC	IMS server control	WS-BPEL	Web services - business process execution language



**Figure 1**  
SIP servlet container.

The core composition logic applies to a multitude of technologies and protocols. The composition engine is thus not restricted to SIP services. Web service invocation, for example, may be used to query external entities in order to make routing decisions (in Figure 2, for example, to link SIP Application 2) or to execute SDP business processes, such as charging or to collect user statistics.

Furthermore, the SIP applications that expose SIP services may also be deployed on different application services. In this case, the composition engine can use the AR interface to push a SIP route that points to the other application server (Figure 2). If the composition engine requires the SIP request to be returned to the current application server after the other application server has finished processing, it may also push a route in the SIP message pointing to itself. If necessary, the other application server may remain on the SIP path using standard SIP techniques.

### Service creation environment for service composition

The prototype includes a graphical service creation environment that is based on Eclipse, an open-source integrated development environment (IDE). The environment supports service composers by providing views for defining service descriptions and constraints, and a graphical editor for com-

binning constituent services into composition templates that can be deployed and executed by the composition engine.

The environment also permits its users to monitor running composite services and to inspect all state information stored in the composition engine. Finally, it may be used to set breakpoints and to execute a composite service in debug mode.

Experience from using the service creation environment in a number of use cases shows great potential for reducing complexity. Indeed, the higher level of abstraction enabled by the graphical representation of composite services and the modular approach used

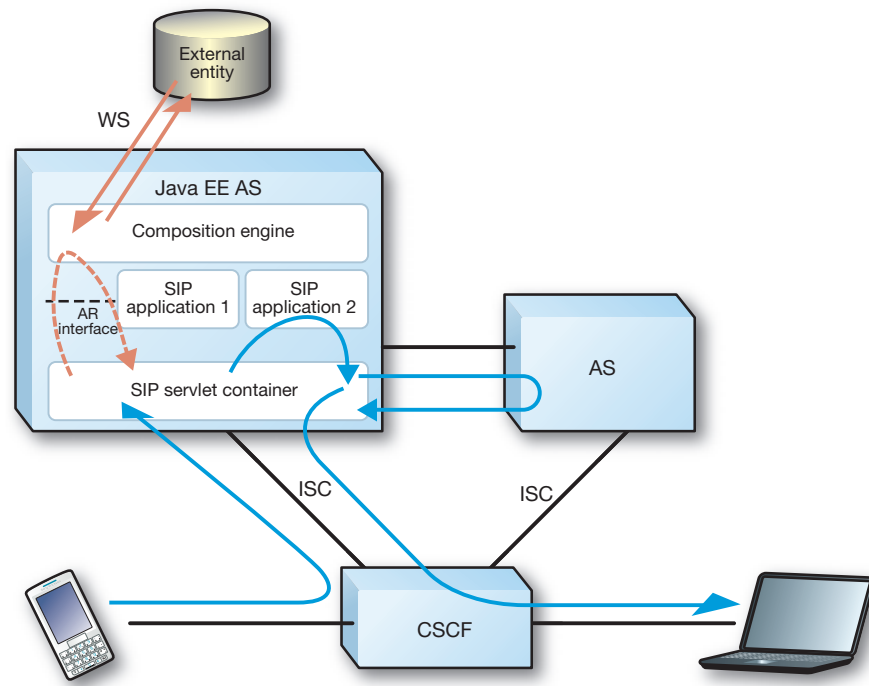
to aggregate constituent services considerably shortens development time. On the one hand, the work of the service composer is greatly simplified, because working with models as opposed to source code gives an overview and allows composers to focus on application logic instead of the details of the source code. And on the other hand, the work of application designers who develop constituent services is simplified through clean encapsulation of functionality. Service-composition principles allow for a separation of concerns and competences (designing SIP applications is one task, composing SIP services is another).

### BOX A, SIP REQUEST ROUTING IN IMS

The IMS call session control function (CSCF) uses the IMS server control (ISC) interface to route SIP messages to IMS application servers (AS). Trigger points in initial filter criteria (iFC) are matched with information in the initial SIP request – for instance, a specific SIP header, to determine which application servers are to be included (and in which order) in the SIP chain.

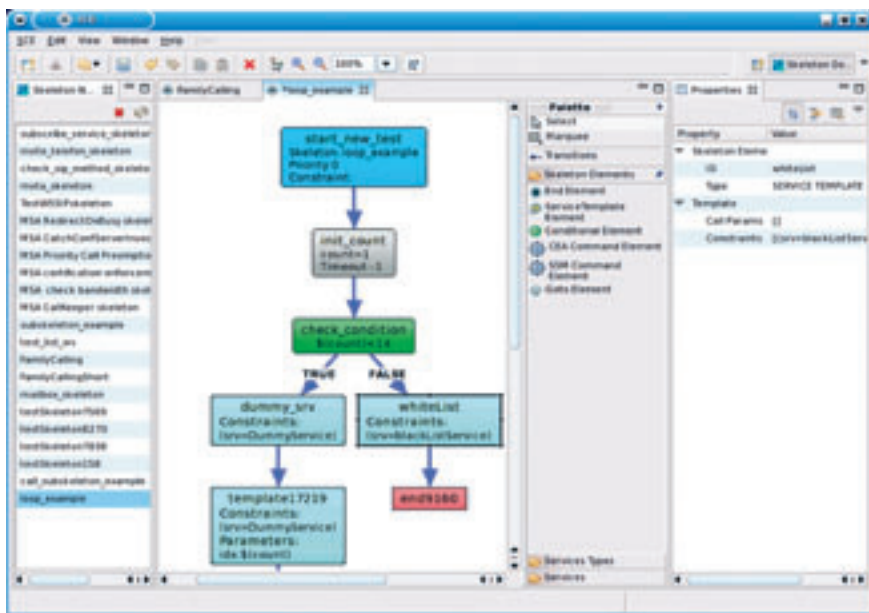
The CSCF might push a route that points back to itself onto the initial SIP request. It then sends the request to the application server associated with the iFC that matched the SIP request. Application servers route requests back to the CSCF after processing them (provided an application in the AS does not terminate the request and the CSCF has pushed the route pointing back to itself).

The CSCF may also trigger on additional iFCs that match the request, causing them to forward the request to other application servers (which repeat the entire process).



**Figure 2**  
External data is used to route a SIP request.

**Figure 3**  
Graphical editor for creating composite services.



## Dynamic SIP service composition

The technologies described in this article collectively represent a SIP application framework. The CSCF identifies the sessions on which the composition engine should operate and links SIP entities according to initial filter criteria (iFC). Using internally stored or derived data, the composition engine uses SIP routing mechanisms to dynamically aggregate SIP entities (constituent services) during the initial SIP request routing. The composition engine thus complements the CSCF.

The SIP entities operate during session setup or during the session. From a composition engine perspective, the SIP entities expose SIP services that are used in a composite service.

## Deployment example

### IPTV deployment

Consider the hypothetical example of blending an IMS-based IPTV application with a presence-enabled chat application. While at home, a user, Shelley, may want to let her friends see which IPTV channel she is watching. A traditional approach to implementing this functionality would entail modifying the IPTV device to send a SIP PUBLISH message to a presence server, which would then notify Shelley's friends.

The effort involved in modifying IPTV devices can have a significant impact on time to market, and hence on capturing consumer interest and ultimately the business opportunity. Moreover, changing the IPTV controller that receives channel selection information increases the complexity of that entity.

However, by employing the approach described above, by proper matching of iFC, the CSCF links (during session establishment) the AS where the composition engine is deployed to the IPTV SIP session. The IPTV controller is then linked by the CSCF, again matching the iFC. When Shelley selects a channel, the composition engine intercepts the SIP message carrying information about the selected channel. The composition engine decides (according to an appropriate policy) whether or not to include the presence network agent (PNA) in the IPTV SIP session. If the PNA is to be included, the composition engine executes the PNA, which sends a SIP PUBLISH message containing

the selected channel to the presence server (Figures 6-7).

As shown above, the composition engine can make detailed, context-dependent decisions about including a constituent service on a per-session basis. The example shows that one may customize and extend the business logic of an IMS application by service composition without actually changing the product and its source code.

Different types of policies (for example, a location-based policy) may be defined for publishing presence information. Moreover, these policies are not hard-coded, but can be dynamically evaluated by the composition engine during IPTV session setup. The choice of policy can be configured per subscriber (different policies for different subscribers). And existing policies may be updated or new policies may be defined without having to modify the presence extension composition template. Subscribers (Shelley) may even change their policies via a portal that provides access to a policy database. All policy changes are automatically taken into account when the next session is set up.

The composition template aggregates a chain of constituent services for processing the initial SIP request; in this case, a SIP INVITE. From the subscriber profile, it fetches the name of the preferred policy for exposing the subscriber's presence information. Once it has obtained the name of the preferred policy, it runs the corresponding policy check. Figure 6 (right) shows a location-based policy that is solely satisfied when the subscriber is at home. After having successfully evaluated the policy, the presence network agent composition template puts the PNAHandler component on the SIP chain.

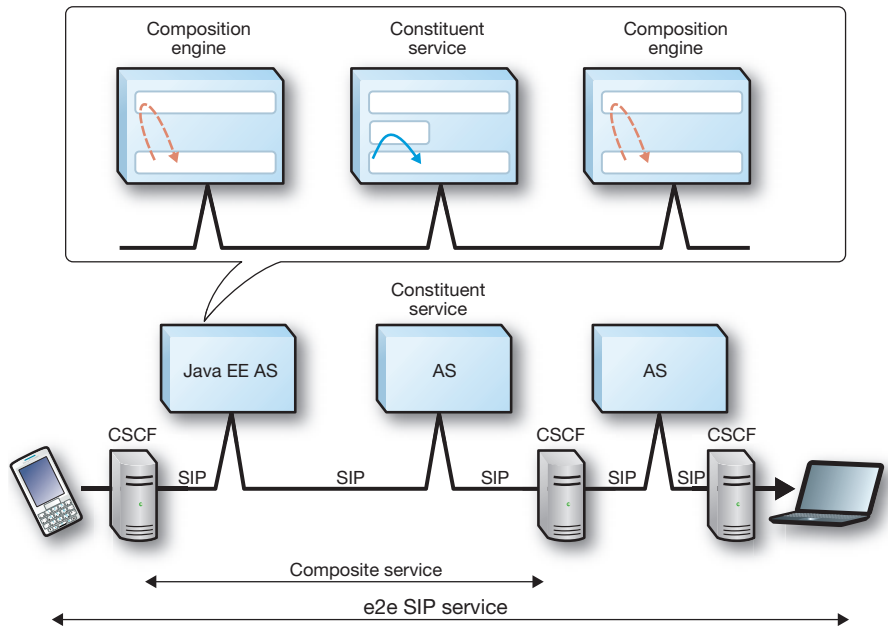
## Conclusion

Ericsson is part of the JSR 289 community that is standardizing an application router (AR) interface to the Java EE SIP servlet container, which controls the order in which SIP entities are linked to a SIP session.

Researchers at Ericsson have shown that an extensible composition engine offers a flexible way of implementing customized services.

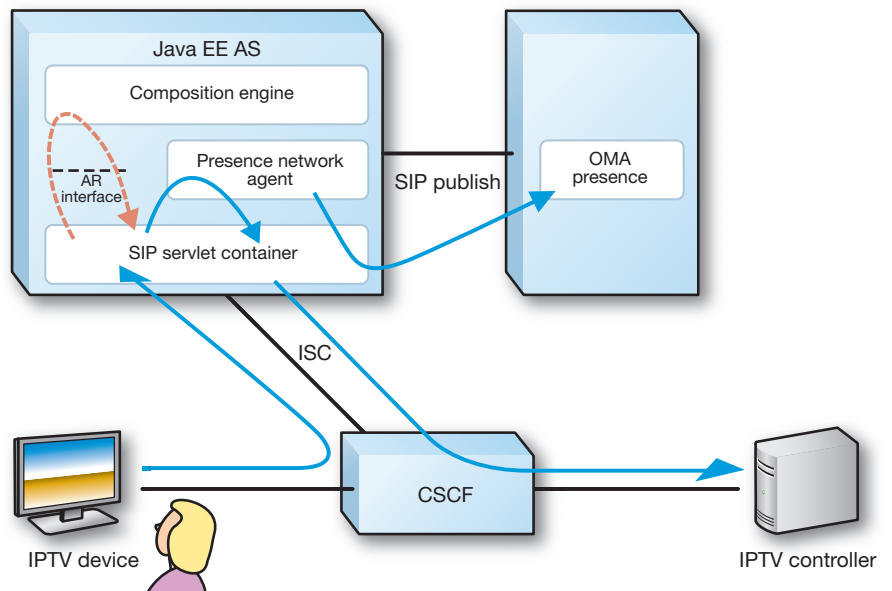
The composition engine flexibly links services offered by SIP entities – for example, SIP services exposed by IMS enablers can be composed into rich, new composite services.

The composition engine implements a



**Figure 4**  
The process of creating a composite service by using SIP routing to aggregate constituent services.

**Figure 5**  
The composition engine inserts a presence network agent into the IPTV SIP dialog.



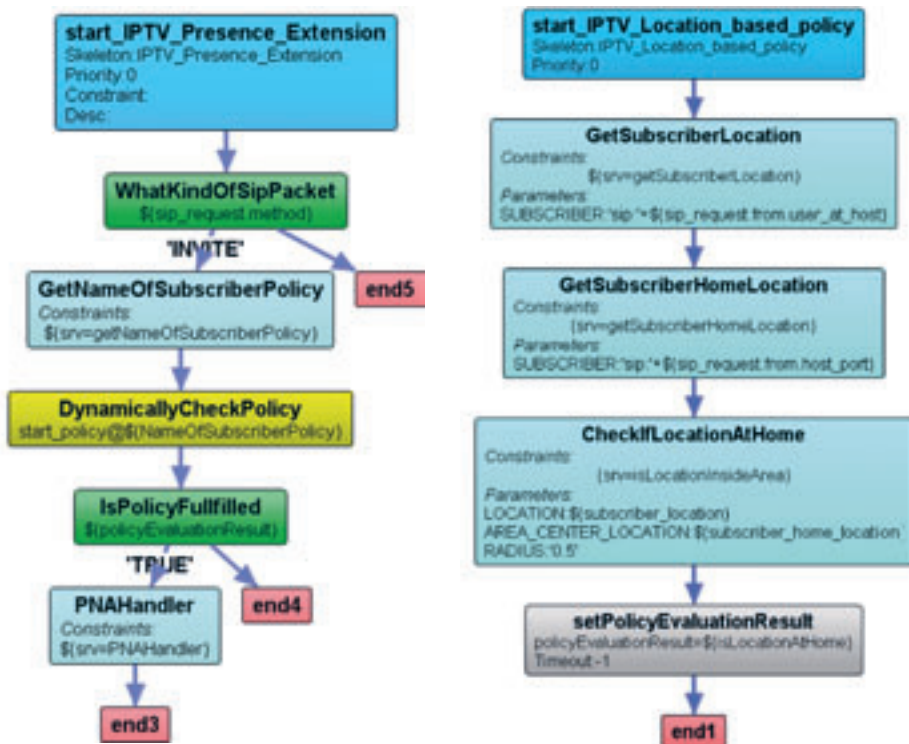


Figure 6  
 Left: Composition template for presence extension.  
 Right: Example of location-based policy.

data-driven algorithm that allows it to dynamically control SIP session establishment according to events. Apart from information in SIP signaling, the composition engine can use state, constraints, external data (for example, web services) and formal descriptions of SIP services to make context-specific decisions for SIP request routing. The described

mechanism also supports loose coupling, which makes composite services more adaptable compared to services developed in the traditional way.

Service composition principles allow for the separation of concerns and roles: designing SIP applications is one task, and composing SIP services is another. Experience from using the

service creation environment shows that the higher level of abstraction enabled by the GUI considerably shortens development time.

A critical factor for future success in this area is continued cooperation in the community toward a common framework that, for example, includes APIs, SIP extensions, and SIP service description formats.

## REFERENCES AND TRADEMARKS

1. Ericsson contributes to the JSR 289 standard together with other key players including SUN, BEA, and IBM. The Ericsson SIP servlet container is available to the open source community, see <https://sailfin.dev.java.net>
2. JSR 289 Java Specification Requests SIP Servlet v1.1, <http://jcp.org/en/jsr/overview>
3. Java Platform, Enterprise Edition (Java EE), <http://java.sun.com/javae/>
4. The 3G IP Multimedia Subsystem, Merging the Internet and the Cellular Worlds, Gonzalo Camarillo & Miguel A. Garcia-Martin, John Wiley & Sons Ltd, 2004.
5. Orchestration in Web Services and Real-Time Communication, Lin Lin and Ping Lin, IEEE Communication Magazine, July 2007, Vo. 45, No.7

- Eclipse is a trademark of Eclipse Foundation, Inc.
- Java and Java EE are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.