An IMS testbed for SIP applications

Caba, Cosmin Marius; Soler, José

Published in:
Proceedings of IIT Real-Time Communications Conference

Publication date:
2013

An IMS testbed for SIP applications

Cosmin Caba, José Soler
DTU Fotonik – Networks Technology & Service Platforms Group
Lyngby, Denmark
+45 4525 3217
{cosm, joss}@fotonik.dtu.dk

ABSTRACT
The paper presents the design and implementation of an emulation platform for the IP Multimedia Subsystem. The SIP Servlet API v1.1 has been used to implement the final system. The purpose of the emulation is to offer to IMS service developers an environment where they can integrate development, deployment and testing of their SIP applications in a single platform, easy to setup and maintain. The emulation platform offers the possibility of configuring network triggers (e.g. initial Filter Criteria and Service Point Triggers) through a web interface, and executing complex test scenarios according to the configuration. The result is ideal to introduce telecommunication students to service development, testing and deployment in a user friendly environment.

Categories and Subject Descriptors
C.2.0 [**Computer-Communication Networks**]: Network Architecture and Design - *Packet-switching network*

General Terms
Design, Experimentation, Verification

Keywords
IP Multimedia Subsystem, SIP servlet, SailFin, test, emulation.

1. INTRODUCTION
One of the key components of a converged IP Multimedia Subsystem (IMS) network is the service layer. The service layer is comprised of application servers and gateways towards third party servers. The applications servers run the software applications that represent the services for the end-users.

It is important to educate developers into programming applications for the IMS network, to provide innovative and appealing services to customers. The platform implemented in this project is targeted towards students who start in the field of programming SIP (Session Initiation Protocol) based services for the IMS network.

While there are free tools and application servers that developers can use to program IMS services, there is no integrated environment offering capabilities similar to the IMS network along the regular development capabilities. The present project aims to cover this niche. The solution we implemented represents a simple interface for students and developers dealing with the topic of IMS service development, to have a single platform where they can create, deploy and test SIP-based applications.

To test a service from a system's perspective [1] one can use a production network or a small scale system built specially for testing purposes. While very useful and precise, this method has the disadvantage of being costly. Another method for testing IMS services is to use an emulation platform in place of the real network elements. On one hand, this method leverages the properties of a real network, offering the possibility to test an ecosystem of services and observe the interactions among them. On the other hand, it is more cost effective, faster and easier to use than a real system. The approach we take towards testing IMS services is to emulate a part of the IMS network. It is important to mention that this paper only discusses the aspect of functional testing [1]. Other types of testing are left for more complex platforms. A brief comparison with other similar systems will be given in a later section.

Our solution is essentially an emulation of the core entity of the IMS network (i.e. Serving – Call Session Control Function). Using the solution proposed in this paper, developers can create realistic IMS network conditions (e.g. service execution, network triggers) within the same machine they use for the development, with a minimum amount of effort. The IMS functionality captured in the emulation is service composition and execution according to network triggers (i.e. initial filter criteria, service point triggers). By using our solution, the developers can verify that an application meets the functional requirements it was designed to fulfill hence perform functional testing.

IMS services may be split in two categories:

- Legacy services: traditional circuit switched services that are enabled through special gateways to translate between the protocols used in IMS and the legacy network [2].
- SIP based services: services that are executed using SIP messages.

The focus of the present paper is on SIP based services created using the SIP Servlet Application Programming Interface (SSAPI) [3], although, it will become clearer later that any SIP services can be tested under specific configuration of the emulation platform.

2. BACKGROUND

2.1 Service triggering

Service triggering relates to the Distributed Feature Composition (DFC) concept [4]. The DFC is a logical architecture for structured feature chaining within a multimedia call. A feature is, in this context, a service (be it SIP based or legacy) that adds value to the call. An established call is composed of a source, a destination and a set of feature boxes (FBs) in between (Figure 1).

The entity that orchestrates the insertion of the feature boxes along the call path is called DFC router. The DFC router coordinates the feature boxes and routes the call setup message towards the correct boxes based on an internal algorithm. The algorithm may consist of precedence rules for the features, user subscriptions or any other mechanisms.

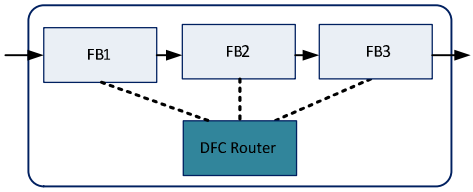

Figure 1. Distributed Feature Composition

The SSAPI can be used to create SIP based services, which are also called SIP applications. In version 1.1 of SSAPI, the DFC concept has been included under the name application selection, and it denotes the process of selecting a set of SIP applications from the applications deployed in a given SIP container. The Application Router (AR) is the entity performing the tasks of the DFC router in the SSAPI. It must be emphasized that the application selection mechanism enables service chaining only within a single SIP container.

In IMS, the service triggering at the network level involves a slightly different mechanism (Figure 2) [5]. The main entity responsible for the service composition is the Serving-Call Session Control Function (S-CSCF). The S-CSCF is aware of all the services available in the system and the subscriber data, which consists of subscription information, different rules depending on the terminal type and capabilities, etc. This information resides in the Home Subscriber Server (HSS) in the form of initial Filter Criteria (iFC) [6]. Unlike a DFC router, the S-CSCF processes every call setup message, therefore it is on the path of the multimedia call. Upon receiving a call setup message (i.e. SIP INVITE) the S-CSCF analyzes the iFC set for both the caller and the callee and creates a chain of services, following that the SIP message is routed to each service in turn, for execution.

An application server may contain multiple SIP applications. The S-CSCF uses SIP routing mechanisms to route the call to an application server but the server must manage internally the selection of the application to be executed for that SIP call. The application server may comprise of several containers, each of them managing one type of application. The SIP Servlet container manages applications build using the SSAPI. Inside the SIP Servlet container, the application selection process follows the DFC model, previously described for the SSAPI. Therefore, both triggering methods are used, one at the IMS network level and the other (i.e. DFC) inside the SIP servlet container.

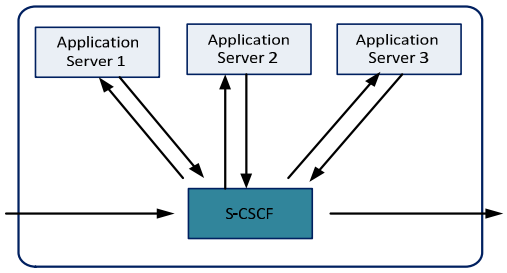

Figure 2. Service triggering in IMS

2.2 The emulation

The paper presents the design and implementation of an IMS emulation platform. The implemented system is named intuitively **IMS Core Emulation** and its main purpose is to provide to service developers an emulation environment for the IMS network, where SIP applications can be deployed and have their behavior assessed. To this end, the emulation provides a graphical interface for configuration of user information (i.e. SIP address, iFC) and executes the service logic encapsulated in the SIP applications according to the configuration. In this context, a SIP application represents the System Under Test (SUT) and the SIP application developer is the tester [1].

Moreover, the emulation platform is integrated with the application server used to run the SIP applications, such that the students or developers can benefit from a unified environment to program, deploy and test the applications' behavior. This represents the main strength and the driver for the IMS Core Emulation platform.

The technology used to implement the emulation platform is the SSAPI, which is also used to create SIP services. The implementation of the SSAPI that we use for the emulation is SailFin [7]. SailFin provides other features apart from the SIP container implementation (e.g. a web container) offering the possibility to implement rich SIP based services that interact with web technologies (e.g. web applications and services). In essence, the IMS Core Emulation can be viewed as a SIP application which behaves similar to the S-CSCF with respect to other SIP applications.

We are striving for simplicity, therefore the emulation captures a thin layer of functionality from IMS, just enough to be able to test the SIP application logic and service interaction.

The features the IMS Core Emulation provides are:

- A graphical user interface (GUI) to configure the test. Through the GUI the developer can set up the network triggers (i.e. iFC, SPTs), user related data (i.e. SIP addresses) and application specific data (i.e. socket, name, etc.).
- The execution of the service logic according to the configuration. In other words, the applications to be tested are executed as dictated by the network triggers.

There are other platforms for IMS, freely available, that offer a larger set of features and a more complete implementation. One such example is the Open IMS Core [8], which captures the standards in great detail. Nevertheless, the advantage of using our solution, the IMS Core Emulation, is that the developer is not required to deploy and maintain a complete IMS system

(comprising of all the core entities). The emulation platform implemented in this project is targeted more towards software developers of SIP applications, and not for telecommunication engineers that have good knowledge about IMS. As we envision, the IMS Core emulation can be used by developers with minimal knowledge about IMS, but good understanding of SIP, in the context of rapid prototyping of SIP applications. The IMS Core Emulation platform basically combines the simplicity of SIP testing tools such as SIPp [9], with the service triggering and composition concepts from IMS. More thorough performance testing for the SIP applications will have to be performed in upcoming phases of the development (and/or deployment) process, using a complete IMS network, which may fall in the scope of the network engineer.

The rest of the paper is structured as follows: section 3 presents details about the implementation. Section 4 describes a typical test scenario and section 5 will give some general conclusion about the project.

3. IMPLEMENTATION

3.1 Design considerations

Before delving into details regarding the implementation of the IMS Core Emulation, it is necessary to show what are the exact entities and functions from IMS that need to be captured in the emulation. Figure 3 illustrates the part of the IMS architecture that is of interest for the project.

The SIP Application Server (SIP AS) is the entity hosting the services in the IMS network [2].

The Home Subscriber Server (HSS) is the user database. The Serving-Call Session Control Function (S-CSCF) is the core SIP server that handles multimedia calls and decides the set of services that must be executed for a particular call [10].

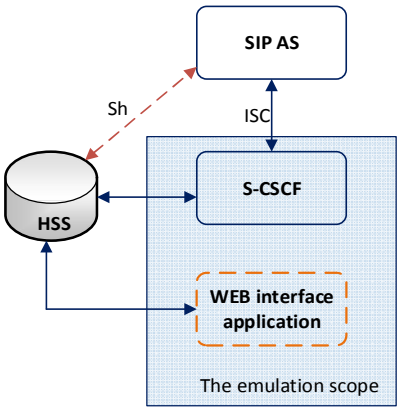

Figure 3. The scope of the IMS Core Emulation

Within the emulation platform, the HSS is a relational database, thus it does not have to be emulated in software.

To be able to offer a GUI to configure the test data, the final system must also include an entity that encapsulates the necessary logic. This entity is represented by the *WEB interface module*, illustrated by the dotted rectangle in Figure 3. Even though the web interface module is in the scope of the emulation platform provided in this project, it does not belong to the IMS architecture as defined in the standards.

Concluding, the entities that fall under the scope of the emulation platform and have been implemented in software are the S-CSCF and the web interface module. The SIP AS runs the applications under test therefore is within the scope of the tester.

As previously mentioned, the emulation is implemented using the SIP Servlet API; therefore it is a SIP application which must run in an application server. The difference between the emulation and any other SIP application lies in the way it processes the incoming SIP messages: it interprets a test configuration and executes the logic of the applications under test. Because both the applications under test and the emulation run in a SIP AS, it is possible to use a single application server to deploy all the software.

The SailFin AS provides the necessary infrastructure to run the emulated S-CSCF and the web interface application. Figure 4 captures a detailed view of the components that build up the IMS Core Emulation. In this schema, a single SailFin AS is used to run both the emulation software and the SIP application to be tested. Later, we will show another scenario distributed over two application servers.

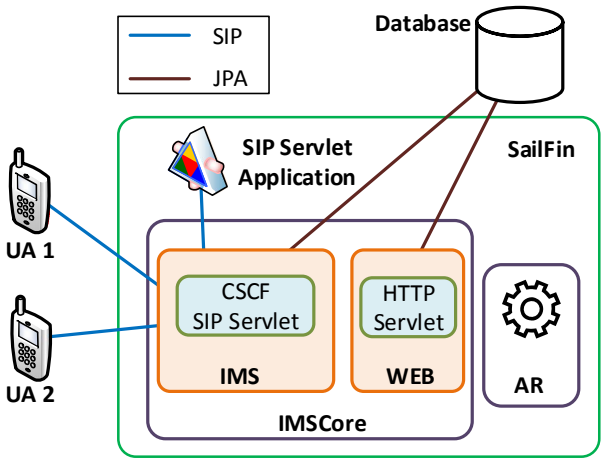

Figure 4. IMS Core Emulation architecture

3.2 Emulation components

As it can be noticed in Figure 4, the final implementation is comprised of two software packages deployed in the same instance of SailFin. The first package, IMSCore, is also the most important because it contains the logic for the emulation platform. The second package, AR, is the SSAPI application router, which contains extra logic to perform the application selection process.

We split the logic of the emulation, encapsulated in the IMSCore software package, in two conceptual modules which we call subsystems: The IMS subsystem and the WEB subsystem. This will make it easier to refer to distinct parts of the implementation.

The WEB subsystem could have been implemented and deployed separately from the IMS subsystem, but we decided to reduce at minimum the number of packages in order to increase the usability, hence they are bundled together. However, the existence of the AR module is imposed by the SSAPI programming model, therefore, it was not possible to discard it or combine it with the rest of the platform.

The database module in Figure 4 represents the HSS from IMS. All the data required for the test is persisted in the database using

the Java Persistence API (JPA) framework. To be able to execute SIP tests, the developer must use a SIP test tool to send SIP requests towards the IMS Core Emulation system. When a test is initiated, a SIP message is send from one User Agent (UA) destined for the second UA. The message is processed by the emulation platform and the target SIP applications are executed according to the test configuration. In this way, the developer can deploy multiple services and test their behavior and the interactions between the services.

3.3 Data persistence

A test configuration consists of a set of initial Filter Criteria (iFC) and Service Point Triggers (SPT) that dictate the order in which the services must be executed and whether or not a specific service needs to be executed. The test configuration is persisted in the database so that it can be reused even if the server is restarted.

The emulation platform must also store the state of the service chain between consecutive invocations of the SIP applications. This information is persisted in a session object running in the server therefore it is not maintained if the application server is restarted during a call setup. We took the decision to store the state of the service chain in the server session in order to decrease the call setup time. Accessing the database several times within a call setup would have been slower than the chosen solution.

3.4 The testing process

In general, the service testing process with IMS Core platform implies the following steps:

- Deployment of the SIP applications to be tested in a SIP Servlet container (can be the same SailFin instance running the emulation or other SIP Servlet container).
- Creation of the test configuration. In this step the developer must configure the following data through the web interface of the emulation platform: the UA SIP addresses, the application specific data (i.e. location where the application can be reached), and the network triggers (iFC and SPT).
- Test execution. A SIP request is send towards the IMS Core Emulation to trigger the service execution, according to the test configuration from the previous step.
- After the test execution is finished, the developer investigates the SIP messages at the originating and receiving UAs, and the log messages at the application server, to decide whether the test has completed successfully or not. The log messages at the application server may contain any of the parameters o the SIP messages processed by the CSCF servlet object, or various indications about the status of the SIP call. Figure 5 shows an example of the standard message given by the emulation platform in case one of the UAs (i.e. originating or receiving) is not registered through the web interface before the test is initiated. In the figure, it can be seen that the INVITE request is processes by the CSCF servlet. The service chain could not be created because the SIP addresses used in the request were not added to the IMS domain hence the CSCF is not aware of those SIP addresses.

The emulated S-CSCF does not fulfill all the functions of a real S-CSCF. Its purpose is limited to interpreting the test configuration and routing the SIP messages towards the applications under test to execute the service logic. Moreover, the emulated S-CSCF can act as a registrar in case the developer activates this feature for a test scenario. More details regarding the IMS and the WEB subsystems will be offered in the following two subsections.

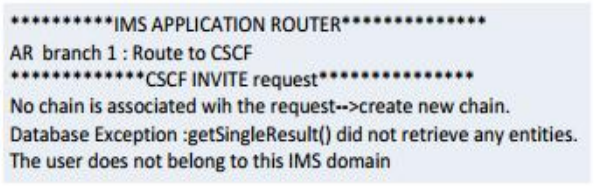

Figure 5. Log message at the application server

3.5 The IMS subsystem

This section further dissects the implementation of the IMS subsystem, therefore, the module responsible for the web interface is not included in the explanations or figures. The SSAPI offers two programming constructs to create SIP applications. The first construct is the servlet object, which defines callback methods where developers can write the code to process SIP messages. The emulated S-CSCF in the IMS Core Emulation is implemented using the servlet object, and it is represented in Figure 4 as the CSCF SIP Servlet entity belonging to the IMS subsystem. To be able to have increased flexibility in terms of managing the SIP sessions, the CSCF SIP Servlet is implemented using the Back-to-Back UA (B2BUA) object from the SSAPI.

The second programming construct that the SSAPI provides is the Application Router (AR) object. The AR has been introduced in v1.1 of the specification to cater for the need of performing complex application selection inside a SIP Servlet container. Every SIP Servlet container must have one AR deployed in order to be able to select a specific application. SailFin provides a default AR which selects the applications in alphabetical order. Since this selection method is not useful in our emulation, we have implemented another AR that replaces the default one provided by SailFin.

It may be the case that the developer would like to program its own application router to tailor a custom mechanism for the application selection process. This is not possible with the IMS Core Emulation since it is mandatory to deploy the provided AR. However, the emulation platform is itself a configurable system that is able to do application selection for any number of applications using flexible rules hence taking the role of the AR. Instead of programming an AR to select applications in a certain manner, the developer could configure the IMS Core Emulation through the GUI, to select the SIP applications to be executed. This is even more useful because, unlike the AR, the IMS Core Emulation is not limited to a single application server, but it can trigger SIP applications located at any network address.

The emulation can be seen as a SIP message router, where the routing logic is dictated by the test configuration. The routing of SIP messages is possible by using the *Route* header. As defined in the SIP specification [15], the *Route* header contains a SIP URI holding the destination address of the message. Therefore, by

manipulating *Route* headers the emulation is capable of instructing the SIP requests to reach specific destinations (which for our purpose are SIP applications). The headers are added to the message by the CSCF SIP Servlet object. However, the use of *Route* headers represents a coarse grained method to route SIP messages because it can only point to a SIP Servlet container. Consider for example two test target applications deployed in the same container. With the help of *Route* headers a SIP message can be routed to the container holding the applications but there is no implicit way to point to the exact application that must be executed, out of the two deployed in the respective container. The AR object complements the routing logic, by advising the container about which application should receive the SIP message, as explained in the DFC section.

It has been mentioned that a test scenario can be distributed over multiple application servers without losing the benefits of the IMS Core emulation. To better understand the mechanisms implemented in the IMS subsystem of the emulation, we show a detailed SIP request routing and service execution in Figure 6.

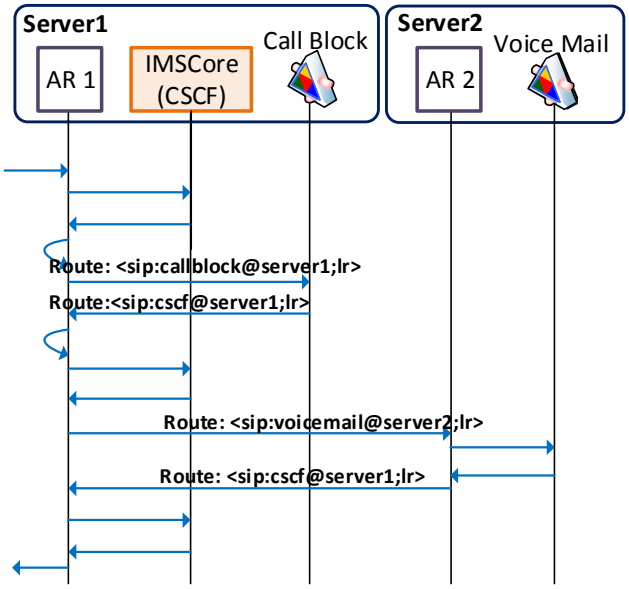

Figure 6. Message chart for the routing process

The scenario shows two application servers: server1 and server2. Server1 runs the IMS Core Emulation and the CallBlock application, while server2 runs the VoiceMail application. It is assumed that the incoming SIP request must trigger the execution of both the CallBlock and VoiceMail application is this exact order. AR1 and AR2 are custom application routers and they are provided together with the IMSCore package.

When the SIP request arrives at server1, the application router is the first entity to analyze it. The AR1 decides the request is initial (it has never been seen by this server), and instructs the SIP Servlet container to dispatch the request to the CSCF Servlet object. The CSCF has no information associated with the SIP request therefore it queries the database and evaluates the iFC and SPTs. The result of the evaluation is a chain of services that must be executed for this SIP request. The state of the chain, containing the CallBlock and VoiceMail applications, is stored in the server session object until all the applications in the chain are executed.

For each SIP application that must be executed for a SIP request, two Route headers are added to the respective SIP message: the first Route header contains the address of the CSCF Servlet object, and the second Route header contains the SIP URI of the application. The headers are processed in the reverse order, such that the request is routed to the SIP application in the first place, and after back to the CSCF for further processing.

In this particular case, the first application that must be executed is CallBlock, hence the CSCF servlet object pushes two routes in the SIP message: sip:cscf@server1 and sip:callblock@server1. The request is forwarded again to the AR1 for the next application selection. The AR1 decides that the request must visit the CallBlock service, and it sends the message to the network interface. However, because the host part of the top most Route header points to the same server (server1), the request is looped back on the network interface and dispatched again to the AR1. The AR1 analyzes the user part (callblock@server1) of the top most Route header and instructs the SIP Servlet container to dispatch the SIP request to the CallBlock application. The request reaches the CallBlock application, and the service logic is executed.

The top most Route header is removed, thus the request is further routed based on the next Route header in the stack (i.e. sip:cscf@server1). When the CSCF receives the SIP request the second time, it retrieves the associated service chain from the session object. The next application in the chain that must be executed is VoiceMail, hence the CSCF pushes the corresponding Route headers. The request is routed in a similar way to the VoiceMail application, except that instead of being looped back at the network interface of server1, the request is routed over the network to server2. In the end the SIP request returns to the CSCF, the service chain is removed from the server session because there are no more applications to be executed, and the request is forwarded to its intended destination.

Apart from the headers defined in the SIP base standard [11], the emulation uses the "P-Served-User" header defined in [12]. This header informs a SIP application on behalf of which subscriber the service logic must be executed.

3.6 The Web subsystem

As depicted in Figure 4, the WEB subsystem module is part of the IMSCore software package. This module supports the web based Graphical User Interface (GUI) through which the test can be configured.

The interface offers great flexibility in configuring tests and modifying already existing configurations. This supports the idea of rapid deployment of various test scenarios. The web interface consists of four pages, each of them providing means to visualize and modify specific information.

The first page of the GUI provides means for configuring the SIP applications under test. The second page contains configuration options for the subscriber data (e.g. SIP address, name). If the CSCF servlet object receives a request with unknown SIP addresses, it cannot retrieve any test configuration from the database hence it cannot create the service chain. On the third page the developer can add a set of iFC for each subscriber. An iFC represent an aggregation of SPTs. SPTs can be viewed as rules expressed as logic sentences. An iFC is deemed valid when the associated SPTs yield the logic value of true. The iFC are

assessed whenever the corresponding user is involved in a SIP session establishment. The fourth view shows the SPT set belonging to a single iFC. Figure 7 illustrates the user dialog for creating an SPT. The configuration of the SPTs is done according to the Disjunctive Normal Form as standardized by 3GPP [5]. The "Condition Negation" field indicates whether the actual SPT is negated or not. The "Group" field defines a subset of SPTs linked with the logical operator AND. Subsets of SPT belonging to different groups are linked between them with the logical operator OR. The rest of the fields define the body of the SPT rule. For a SIP request, an SPT is valid (logically true) if and only if the fields of the SPT match the headers in the SIP request.

All the data configured through the web interface can be modified at any point in time. The modifications are first taken into account for the next test initiated immediately after a modification has been made.

Figure 7. Creation of a Service Point Trigger

4. TEST SCENARIO

To emphasize the strengths of the IMS Core Emulation platform, Figure 8 illustrates a more complex test scenario. A single application server is used to run all the software packages: the IMS Core Emulation and the SIP applications that have to be tested. The main advantage of using a single application server is the ease of use and a decrease in computing resource consumption. Application A and B contain very simple logic to record statistics about the processed SIP requests. These two applications are used only to validate the behavior of the test target application, Call Forwarding.

For this particular test scenario the developer needs three user agents. The UA associated with Alice is the caller, Bob is the initial receiver of the call, and John is the receiver after the call is redirected by the Call Forwarding service. At first, the developer needs to configure the database with the right information. According to the test configuration, the services must be executed in the following order: for Alice no services must be executed; for Bob the first service to be executed is Call Forwarding. If the Call Forwarding application detects that Bob is available then application A must be executed on behalf of Bob before the call is forwarded to the destination. If Bob is not available, Call Forwarding routes the call to John, and application B must be executed on behalf of John.

When the test is initiated from Alice's UA, the emulation platform fetches the information from the database and builds the service chain. It first routes the SIP request to the Call Forwarding application. As assumed, the Call Forwarding decides to change the destination of the call to John. The emulation platform detects the change in destination and rebuilds the service chain according to the new destination; it adds the services associated to John in the chain instead of those associated to Bob. The processing continues with application B which is executed on behalf of John and the SIP message is finally forwarded to John's UA.

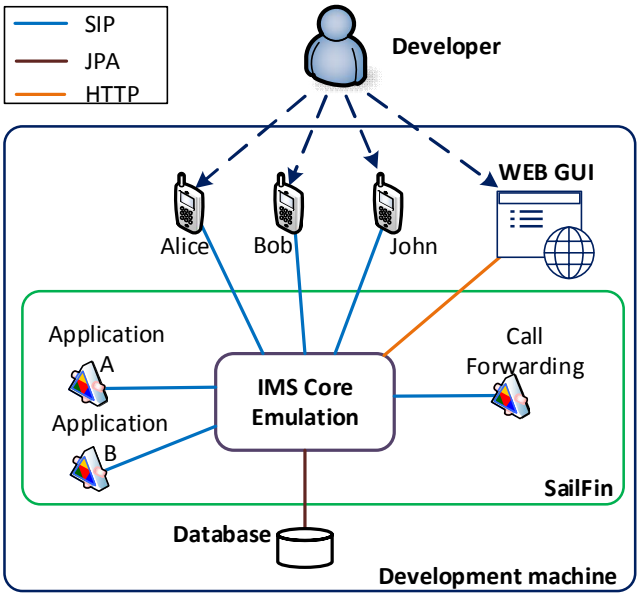

Figure 8. Test scenario

By investigating the log messages from application A and B and from the UAs, the test is considered passed, meaning that the Call Forwarding application functions properly. Without the emulation platform, the developer would have to write a significant amount of extra code to implement the routing of messages between the applications in order to validate the behavior of the Call Forwarding service. Furthermore, different scenarios can be tried just by adjusting the test configuration in the GUI.

5. DISCUSSION AND FUTURE IMPROVEMENTS

The emulation platform provided in this project has been created for the purpose of functional testing of SIP applications. It offers capabilities for service composition, using network triggers, similarly to the S-CSCF entity from IMS. Furthermore, it integrates with the SailFin application server offering a unified platform for creating, deploying and testing SIP applications. The main advantage over already existent solutions is that developers will experience an integrated environment in the same development machine, and the same application server used for

SIP applications. The emulation captures only a part of the functionality of the IMS S-CSCF entity. The implementation of the ISC interface follows the 3GPP standards specification to reduce the eventual mismatches between the emulation platform and the tested SIP applications.

The IMS Core Emulation has been designed to be easy to use. It requires one SailFin application server and a relational database system. The provided software packages must be deployed in the application server and the emulation platform is up and running. Furthermore, it can trigger the execution of services running on application servers at remote locations, even if the SIP applications under test are created with different APIs (e.g. JAIN) or they run in a different application server (e.g. Mobicents) [13], [14].

The GUI is intuitive and offers great flexibility in setting up and modifying test configurations. To create different test setups, the developer needs to configure a number of iFC and SPTs without being worried about the implementation of the extra code needed to orchestrate the execution of several SIP applications.

The most notable limitation that the IMS Core Emulation exhibits at the time of writing is its sensitiveness to stress testing. The interactions between the application router and the CSCF Servlet object have a significant contribution to the delay of the SIP session establishment. Therefore, testing the performance of an application in terms of handled calls per second is going to be limited by the emulation platform rather than the application itself.

Moreover, the implemented system does not include a DNS subsystem. In conclusion domain names cannot be resolved and are not supported. The entire configuration must be made with IP addresses and port numbers.

The present work was performed by the main author as his Master Thesis project. The resulting platform is expected to be used by students interested in service development for telecommunication networks within the Technical university of Denmark. As future plans we would like to extend the emulation platform to the Mobicents SIP servlet [14]. Both Mobicents and SailFin use the same standard (SIP Servlet API) as a reference [3].

6. REFERENCES

[1] Mei-Chen Hsueh, Large Complex System Test: Objectives & Approaches, First IEEE International Conference on Engineering of Complex Computer Systems, pp. 405-408, 1995.

[2] 3GPP TS 23.218 V11.3.0, pp. 12-18, 2012.

[3] M. Kulkarni, Y. Cosmadopoulos, SIP Servlet Specification, version 1.1, August 2008.

[4] P. Zave, Modularity in Distributed Feature Composition, 2009.

[5] 3GPP TS 24.229 V11.4.0, 2012.

[6] 3GPP TS 29.228 v11.4.0, IP Multimedia (IM) Subsystem Cx and Dx interfaces, June 2012.

[7] http://sailfin.java.net/, accessed 17 June 2013.

[8] http://www.openimscore.org/, accessed 17 June 2013.

[9] http://sipp.sourceforge.net/, accessed 17 June 2013.

[10] 3GPP TS 23.228 V11.5.0, 2012.

[11] J. Rosenberg et al., SIP: Session Initiation Protocol, IETF RFC 3261, June 2002; http://www.ietf.org/rfc/rfc3261.txt.

[12] J. van Elburg, The SIP P-Served-User Private-Header (P-Header) for the 3GPP IP Multimedia (IM) Core Network (CN) Subsystem, IETF RFC 5502, April 2009; http://tools.ietf.org/html/rfc5502.

[13] https://jsip.java.net/, accessed 17 June 2013.

[14] http://www.mobicents.org/, accessed 17 June 2013.