Introduction of Media Gateway Control functions in Java Call Control

M. Femminella, *IEEE Member*, F. Giacinti, G. Reali, *IEEE Member*
DIEI, University of Perugia, Via G. Duranti 93, 06125, Perugia, Italy
{mauro.femminella, francesco.giacinti, gianluca.reali}@diei.unipg.it

***Abstract*— With the ongoing convergence of telecommunication networks towards an all IP architecture, service providers are challenged by the need of continuously renewing their offer of value-added multimedia services. In this paper, we propose an abstraction layer to be used in application servers, based on the concepts proposed in the so called Java Call Control (JCC) specifications. It simplifies the creation of multimedia services based on both the Session Initiation Protocol (SIP) and the Media Gateway Control Protocol (MGCP). In order to show its effectiveness in simplifying service design and implementation, we have implemented a JCC Resource Adaptor for a JAIN Service Logic Execution Environment (JSLEE), using the Mobicents application server, which is the only existing open-source JSLEE implementation. Experimental results, obtained by implementing a complex VoIP service, show both the feasibility and the very good performance of our proposal.**

Keywords-Java Call Control (JCC), MGCP, SIP, JSLEE

I. INTRODUCTION

Improving the design and implementation processes of multimedia services is essential for telcos and application service providers willing to rapidly develop new applications by resorting to the use of third-party applications and libraries. Due to its platform independence, Java is a natural candidate for implementing advanced services. In particular, the Java APIs for Integrated Networks (JAIN) Server Logic Execution Environment (SLEE) have been created to explicitly aid developers in creating, deploying, and managing advanced telecom services [1]. JSLEE specifications explain how to realize communication platforms able to fulfill the requirements of multimedia services in terms of latency, throughput, and availability), also providing a point of integration of different resources and protocols.

A further step in simplifying service creation is the set of Java Call Control (JCC) APIs [3][4][5]. It has been created with the specific objective of lightening service developer from the burden of handling the underlying communication and signaling protocol system (e.g. SS7, H.323, SIP, MGCP).

So far, few papers have shown proposals for mapping JCC onto SIP [6][7] without including detailed implementation information, but none of them show a JCC mapping onto MGCP. Focusing on the JSLEE, implementations of JSLEE JCC adaptors currently exist only for intelligent network (IN) protocols (e.g. CAMEL, CAP, INAP). Even the main JSLEE implementers (OpenCloud [8] and Amdocs jNetX [9]) have not provided a JCC support for SIP and/or MGCP yet, probably because SIP and MGCP are rather different from other IN protocols, and the JCC-SIP and/or the JCC-MGCP mapping is not straightforward.

The contribution of this paper is the definition of a novel JCC-SIP-MGCP mapping. In particular, we focus on the detailed description of the JCC over MGCP mapping added to the our previous JCC to SIP mapping [13], along with a performance comparison with respect to pure SIP-MGCP-based services. For the reasons explained above, in our proposal the semantic of some JCC methods has been slightly modified. All changes and extensions are thoroughly explained in what follows, along with their implementation within the Mobicents Communication Platform [10], which is the only available open-source JAIN SLEE.

The paper is organized as follows. Background concepts are reported in Sections 2. JCC-MGCP operation is described in Section 3. Section 4 reports some implementation details and experimental results achieved by using the implemented JSLEE component. Concluding remarks are reported in Section 5.

II. BACKGROUND AND RELATED WORKS

A. SIP and MGCP protocol basics

The SIP protocol [11] is an application-layer signaling protocol used for creating and managing application sessions over IP networks. Session related information, such as media encoders, is negotiated through Session Description Protocol (SDP) [12] contents transported in SIP messages. SIP is a simple, text based request-response protocol, allowing multimedia sessions to easily include messages, voice, and video communications. Sessions are established through a three-way signaling exchange between a so-called User Agent Client (UAC) and a User Agent Server (UAS). An exchange starts with an INVITE message, sent by the UAC to the UAS, one or more provisional responses (e.g. 180 Ringing) replied back by the UAS, a final response sent by the UAS to the UAC (200 OK), and a final acknowledgment (ACK) sent by the UAC to the UAS. The set of messages including a request and the relevant responses consists of a SIP transaction. Typically, SIP signaling is exchanged through SIP proxies, allowing end-users to be identified and located by SIP Uniform Resource Identifier (SIP URI) and not by (time-varying) IP addresses.

The MGCP protocol is a transactional or command-

response text-based protocol [2] used for managing a Media Gateway (MG) by an external network element, called Call Agent (CA), implementing call control functions. A MG is a network element that mainly provides media playing/recording functions, including conversion between the telephone audio signals and media data packets. The MGCP connection model is based on endpoints and connections. The endpoints can be physical or virtual and are both sources and sinks of data (e.g announcement server access point, interactive voice response (IVR) access point, packet relay). The connections can be either point-to-point (i.e. an association between two endpoint) or multipoint (i.e. an endpoint connected to a multipoint session). The MGCP protocol defines nine commands, along with relevant responses, illustrated below.

- Create/Modify/DeleteConnection: sent by the CA to the MG to create/modify/delete connections with endpoints (e.g packet relay and IVR).
- NotificationRequest: used by the CA to instruct the MG to play/record a signal on the connection and/or to detect events on the endpoint/connection.
- Notify: used by the MG to inform the CA of an detected event, previously required with a NotificationRequest command.
- RestartInProgress: used by the MG to inform the CA of the endpoints change state.
- AuditEndpoint/Connection: used by the CAs to check the state and parameters of an endpoint/connection.
- EndpointConfiguration: used by the CA to instruct the MG about the coding characteristics of the endpoint.

B. *Java Call Control application programming interface*

The JCC API can handle communication sessions through a variety of heterogeneous networks [3][4][5], hiding most of complexity of underlying network protocols. It included the entities illustrated below.

- JCC Provider: it is the interface through which an application can access the implemented JCC functions. A JCC Provider manages a number of JCC Calls.
- JCC Call: it represents a communication between two or more parties through a dynamic collection of physical/logical entities involved in the communication relationship. It can be in the IDLE state (not associated with any JCC Connection), in the ACTIVE state (it has some ongoing activity and must have at least one associated JCC Connection), in the INVALID state (all its JCC Connection objects are lost).
- JCC Connection: it is a dynamic relationship between a Call and an Address; each JCC Call is composed of a number of JCC Connections. Figure 1 shows the Finite State Machine (FSM) associated with a JCC Connection, along with the modifications proposed in [13] to the FSM presented in [5]. The IDLE state is the initial state. The AUTHORIZE CALL ATTEMPT is associated with authorizing involveed terminals for the Call. In the ADDRESS COLLECT state, the initial address information package is collected and processed according to a "dialing plan" to determine the final address. In the ADDRESS ANALYZE state, the collected information is processed to resolve routing address and call type; the CALL DELIVERY state of the originating party implies the selection of the route and forwarding a message to notify the called party of the willingness to start a call. On the terminating side, this state implies checking the busy/idle status of the terminating access so as to notify the terminating party of an incoming call; the ALERTING state means that the Address is being notified of an incoming call; the CONNECTED state implies that a JCC Connection and its Address are part of a call. Thus, two communicating parties are represented by two JCC Connections of the same JCC Call staying in the CONNECTED state. In the DISCONNECTED state, a Connection is no longer part of the call; the FAILED state indicates that a Connection has failed. Each state transition of the JCC Connection FSM generates an event. In order to deliver events to the appropriate listeners, the JCC specifications use EventFilters.
- JCC Address: it is a logical endpoint (e.g. an IP address or a directory number), and has a string representation.

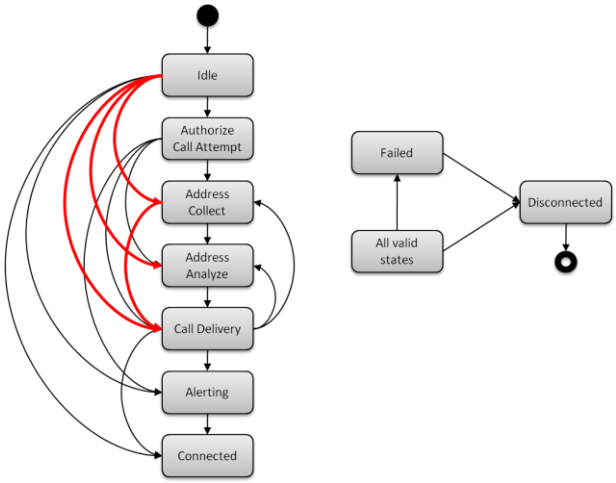

Figure 1: FSM of the JCC Connection, red lines indicates new transitions introduced in [13], dark lines those proposed in [5].

C. *JSLEE specifications*

The JSLEE is a Java standard specification designed for executing telecom services. It is event-driven and designed for hosting high performance, asynchronous, and fault tolerant application servers (ASs). In more detail, a JSLEE AS is a software container able to provide the non logical features for executing applications, which relieves programmers from the burden of dealing with low-level implementation aspects. It includes tools such as: (i) Resource Adaptors (RAs), realizing an abstract interface layer used for accessing external resources such as protocols and databases; (ii) an Event Router, which delivers events to the appropriate service modules; (iii) facilities, useful to implement the service logic. Service logic is organized in components called Service Building Blocks (SBB), which operate asynchronously by receiving, processing,

and firing events. Events coming from the external world, such as reception of a SIP message, are translated into internal Java events by the relevant RA (e.g. the SIP RA). Currently three main implementations of the JSLEE specification exist: Rhino, a commercial AS owned by Open Cloud [8], Convergent Service Platform, a commercial AS owned by Amdocs [9], and Mobicents JSLEE [10], which is owned by Red Hat and is open source. It includes a JSLEE container, a Media Server (MS), a Presence Server, and a SIP Servlet container.

III. JCC-MGCP MAPPING

Our JCC-MGCP mapping extends the JCC-SIP mapping presented in [13]. This extension adds the possibility to use both SIP and MGCP protocols under the JCC abstraction layer instead of using them separately, thus including all the aforementioned facilities and benefits. In more detail, this allows handling communication with MSs by JCC. Note that we are not dealing with the *control* of simple streaming servers, such as VLC or Windows Media Server, but rather with MSs able both to play and to record user input (voice/video) and capture DTMF tones, typically used in IVR facilities. In these cases, the usage of MGCP (or a similar protocol such as MEGACO [19]) is strongly recommended, since its specifications [2] already support these control functions. In this section we focus on the JCC-MGCP mapping only, since details relevant to the JCC-SIP mapping can be found in [13].

We have implemented the JCC-MGCP Provider, the JCC-MGCP Connection, and the JCC-MGCP Address. We have decided to not allow the creation of a call that has only JCC-MGCP Connections. This means that each JCC-MGCP Connection must be associated to an existing JCC-SIP connection, with the SDP information already provided, since this is the case of interest in current deployments of service provider networks. In our implementation, the JCC Call instance decides the type of JCC Connection to be created, based on the target address provided in the JCC createConnection(…) method (see [3] for further details).

The JCC-MGCP Provider implements a MGCP listener, the main tasks of which are the following:

- receiving the MGCP messages coming from the network (through the MGCP stack) and delivering them to the relevant JCC-MGCP Connection;
- delivering the MGCP messages coming from the JCC-MGCP Connections to the MGCP stack, which must be sent into the network;
- creating JCC-MGCP Addresses;
- managing associations between JCC-MGCP Connections and the MGCP transactions and handling transaction timeouts.

Each MGCP connection has been associated to a JCC connection through a JCC-MGCP Connection Handler. The main tasks of this handler are the following:

- storing the MGCP connection parameters (e.g. endpoint types and identifiers, announcement signal path, DigitMap, connection mode, etc.) and updating them when required by the application service, by using the JCC selectRoute(…) method of the JCC-MGCP Connection, explained in detail below;
- managing all MGCP messages (both commands and responses) relevant to the associated MGCP connection; to instruct both the JCC-MGCP Connection to fire the right JCC events and the JCC-MGCP Provider to send commands and responses relevant to the previous messages;
- automatically handling the MGCP connection modifications and updating its internal FSM, as shown in Figure 2.

We have chosen to adapt the JCC selectRoute(...) method of the JCC-MGCP Connection to insert/change parameters of MGCP connections, due to the absence of a well defined method to do this in the JCC specifications. Thus, the proposed usage of the JCC selectRoute(...) method slightly changes its original semantic [3]. Neverhteless, since it allows passing a String value, we forced it to pass configuration parameters to the MGCP stack below. The relevant JCC-MGCP Connection Handler parses the String when the method is invoked and then inserts/changes the relevant parameter. Since each String passes a single parameter, the information about String length is not needed, and its format is [Parameter_Type]:[Parameter_Value]. The [Parameter_Type] is a code of four characters that indicates the parameter to insert or modify. The [Parameter_Value] is compliant with the MGCP specification format of the relevant parameter type.

The MGCP connection parameters have been classified in:

- Mandatory parameters: these parameters (e.g. endpoint types, IP address of the MS) are fixed for each MGCP connection. Thus, when the service changes one of them and invokes the JCC attachMedia() method of a JCC-MGCP Connection, the underling system automatically deletes the existing relevant MGCP connection(s) by sending the MGCP Delete Connection command(s) and creates a new MGCP connection. All these operations are performed without requiring the service to handle the MGCP message exchange. The service will be informed of this change by the JCC Connection MidCall event with code "CONNECTION_CHANGED".
- Modify parameters: these parameters (e.g. user SDP, connection mode) requires only to send a MGCP Modify command to update them on the relevant MGCP connection. As before, this is done automatically by the underling system and the application service will be informed of this change by the JCC Connection MidCall event with code "CONNECTION_CHANGED".
- Request parameters: these parameters (e.g. audio path, DigitMap) are relevant only to a request and are used to create the MGCP Notification Request command that instructs the MS to execute a task.

Each JCC-MGCP Connection Handler has an associated FSM representing the status of the MGCP connection (Figure

2). Remember that the JCC-MGCP is an abstraction layer used for handling in a simplified way the MGCP protocol exchanges implemented in the AS, which acts as a CA. The meaning of the FSM states is explained below:

- Idle: in this state the MGCP connection has not been created yet or it has been lost for any reason;
- InConnection: this is a transient state during which the handler is trying to establish a connection with the MS. The first time that the handler enter in this state, the associated JCC-MGCP Connection changes its JCC state to CALL_DELIVERY (see also Figure 1);
- Connected: in this state the MGCP connection has been created and is ready to receive requests. The first time that the handler enter in this state, the associated JCC-MGCP Connection changes its JCC state to CONNECTED;
- Reconnection: this is a transient state due to control reasons. In this state the handler is trying to delete the existing MGCP connection and to create a new one;
- Error: in this state, the MGCP connection has been deleted due to unexpected event (e.g. a negative answer from the MS to a MGCP Modify Connection command), but the JCC-MGCP Connection is still alive and can be used to create a new MGCP connection (depending on the service logic);
- InDisconnection: this is a transient state during which the handler is trying to delete the MGCP connection with the MS.
- Disconnected: this is the final state. When the handler enters this state, the associated JCC-MGCP Connection changes its JCC state to DISCONNECTED.

A JCC application willing to establish a communication between a MS and a SIP UA (represented by a JCC-SIP Connection [13]) must execute the following steps:

- invoking the createConnection(A,B,-,-) JCC call method to create a new JCC-MGCP Connection, where A is the string that represents the MS address (IP and port) with the "mgcp:" prefix and B is the string that represents the address of an existing JCC-SIP Connection;
- setting the endpoint type by invoking the selectRoute(…) method on the JCC-MGCP Connection. The endpoint type can be Announcement, IVR, Packet Relay with Announcement or Packet Relay with IVR. The support for others endpoint types will be added in future works;
- invoking the routeConnection(false) method on the JCC-MGCP Connection to start the message exchange so as to create the connection on the MS and to allow the UA and MS to exchange SDP information. In this regard, it is useful to look at Figure 3, which reports the signaling exchange of the proposed test service, and to consider the initial signaling exchanges from the first INVITE to the reception of 183 with matched SDP information. This last step allows choosing the right codec and makes the communication from the MS to the SIP UA, and vice versa, possible. Note that the choice and the handling of the suitable sequence of signaling messages is automatically done by the underlying system, according to the state of the relevant JCC-SIP Connection. Thus this is *completely transparent* to application developers, who have to implement only the service logic relevant to the application. This is one of the main advantages of using the JCC abstraction layer to develop complex telecommunication services.
- waiting the state changing event of the JCC-MGCP Connection to CONNECTED. This indicates that the MS connection has been created and is ready to receive requests, and the SDP information has been exchanged.
- sending MGCP requests by invoking JCC methods in the following order: i) selectRoute(…) invocations to specify the parameters of MGCP connection, one invocation for each parameter until all of them have been communicated, ii) one attachMedia() invocation on the JCC-MGCP Connection. After these operations, the JCC application waits the result relevant to the requests (e.g. successful or failed, DigitMap matched or not), which will be deliver with a JCC connection MidCall event. The above sequence of signaling messages can be repeated depending on the application service logic.
- invoking the release(…) method on the JCC-MGCP Connection to delete the MS connection when either there are no more requests to be sent or the application wants to terminate it according to the service logic.

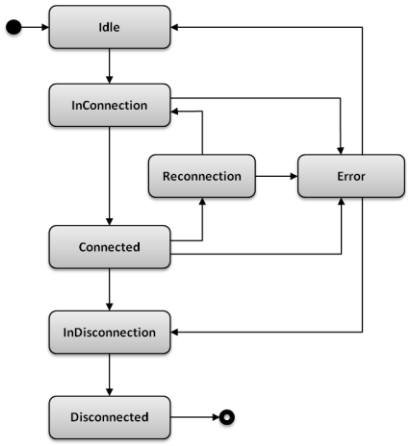

Figure 2: FSM of the JCC-MGCP Connection Handler.

IV. PERFORMANCE EVALUATION

In order to test the effectiveness of our solution, we have implemented a service that requires the usage of both MGCP and SIP protocols. The service, called Prepaid Card Service (PCS), consists of a third party call control based on a central back-to-back UA (B2BUA), where the caller (UAC) wants to use its own credit available in a prepaid card to perform a call towards a callee (UAS). The audio menu of the service requests

the caller to insert his card number together with the PIN code and the phone number of the callee. Then, after having validated the authentication data provided by the caller as DTMF tones, the service decides if starting or rejecting the call.

The control unit is implemented by a Mobicents JSLEE (MSLEE) which acts as an AS implementing both the SIP B2BUA and the MGCP CA, and uses the JCC-SIP-MGCP RA as bridge between the SLEE logic and the underlying signaling protocols. Five main signaling entities are involved: SIP UAC, SIP UAS, the MSLEE, the MS and a Database (DB), used to store user profiles and service related parameters. Signaling flow is illustrated in Figure 3. It is started by the SIP UAC, which sends a SIP INVITE message towards the MSLEE. When this INVITE is received, the MSLEE event routing subsystem is invoked and a Selector root SBB is created. It immediately queries the DB in order to retrieve the subscriber profile. Now the Selector SBB has all the information needed and can activate a child SBB called PrepaidCardCallControl and leave the signaling flow control to it. At this point, the child SBB uses the SDP provided by the UAC to create a communication channel with the MS. To do this, a Create Connection (CRCX) and a Modify Connection (MDCX) MGCP commands are sent to the MS including the UAC SDP. The MS allocates the needed resources for the connection and generates its SDP. This last is transmitted back by the MS to the MSLEE within the 200 (MDCX) MGCP response. The MS SDP is then forwarded to the UAC within the 183 (Session Progress) SIP response to allow establishing the communication between the MS and the UAC. At this point, the business logic of the service running in the MSLEE instructs the MS to play an audio message to the caller and/or specifies which UAC inputs the MS has to listen to. This is done by the Notification Request (RQNT) MGCP command. A Notify (NTFY) MGCP command is sent from the MS to the MSLEE when the task is finished. This message contains both the task result (successful of failed) and the list of caller's captured inputs. Based on this information, the business logic of the service can make its choices and apply its policies. After that all needed information have been collected and validated, the child SBB creates the second call leg towards the UAS by sending a SIP INVITE to it including the UAC SDP. The SIP signaling exchange continues until the two SIP three-way-handshake are completed and thus the media session between UAC and UAS is established. Periodic DB queries are used to apply the operator-defined billing profiles and record the remaining credit within the DB during call lifetime. A final DB query, upon call termination, is used to update service-related information in the DB. In our implementation, the call is closed by the AS by sending BYE messages towards the two SIP parties, modeling a typical scenario of online charging when the user credit finishes.

The JCC RA for SIP and MGCP has been implemented as part of the MSLEE version 1.2.6 GA, compliant with the JAIN SLEE 1.0 specifications. The link between the RA and the underlying SIP network has been realized by using the JAIN SIP API and JAIN SIP Reference Implementation (RI) v1.2, while the MGCP part has been implemented with the support of the JAIN MGCP RI v1.0. The same implementations have been used to implement the providers in the JCC RA.

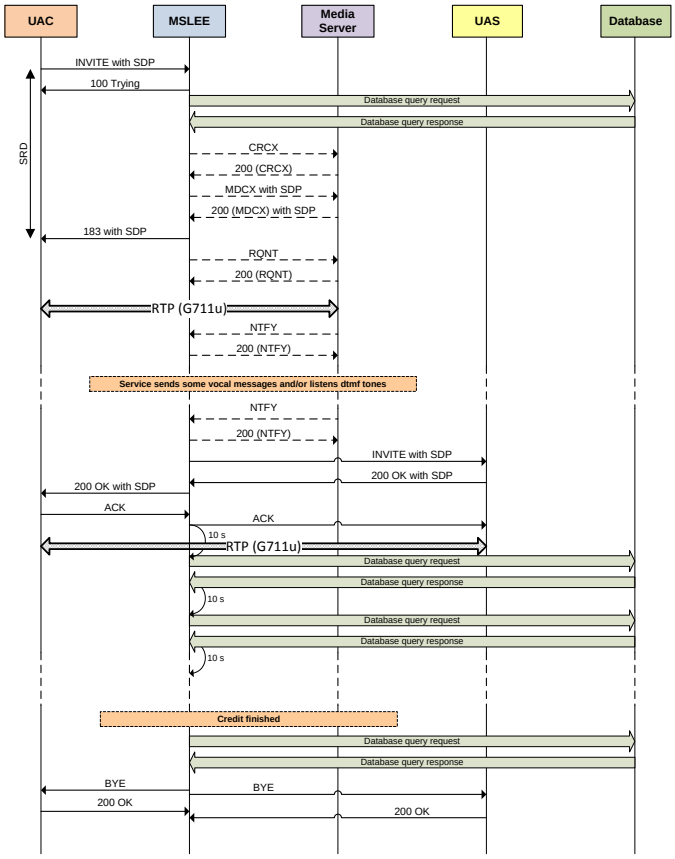

Figure 3: Prepaid card service signaling flow.

The SIP UAC and UAS have been implemented by using two PCs, with the 32-bit CentOS v5.5 operating system (OS), running the SIPp traffic generator [15]. Four MSLEE instances have been virtualized by using the VMWare ESXi 4.1 hypervisor, installed on a Fujitsu-Siemens PRIMERGY TX300 S4 server with dual Intel Xeon E5410 (i.e. 8 CPU) and 16 GB of RAM. One VM hosts the MSLEE under test, running the PCS service, configured with CentOS 32-bit v5.5 OS and 2 virtual CPUs and 3 GB of RAM; the JVM version is 1.6.0.30 32-bit configured with a heap size of 2.5 GB. The other three VMs are identical and host the MSs (Mobicents MS v1.0.3GA) configured with Ubuntu Server 64-bit v11.10 OS, 2 virtual CPUs and 4 GB of RAM each one; the JVM version is 1.6.0.30 64-bit configured with a heap size of 3.5 GB. The choice of measuring the performance of a VM with limited CPUs and RAM is justified by the results presented in [14], which shows that this deployment optimizes usage of computing resources for java servers. Using three MSs selected in a round robin fashion ensures that the MS is not the system bottleneck.

We have tested the service using our JCC-SIP-MGCP RA implementation. We have compared the results achieved with those obtained by the same service implemented by using both the SIP RA and MGCP RA methods (standard implementation), already tested in [17]. With respect to the settings described in [14], we have used the Parallel garbage collector with the parameter NewRatio set to 10, as suggested in [16]. In addition, we have optimized the JBoss logging system by restricting package category priority and by rising

the global logger threshold. Subscriber policies have been stored into a MySQL DB version 5.0.51a running in a PC with CentOS 32-bit v5.5 OS.

The SIPp UAC generates SIP calls according to a deterministic arrival process. The duration of each test was 60 minutes, and each call duration was 180 seconds. We measured i) the maximum call throughput, expressed in calls per second (cps), defined as the maximum handled load with a call loss percentage less than 1%, and ii) session request delay (SRD), defined as the time interval from the initial SIP INVITE to the first non-100 provisional SIP response (see Figure 3). SRD values are important since they are related to the latency experienced by a caller initiating a session, which cannot be unbounded. In fact, as shown in the real field measurements presented in [18], just transport delays in 3G networks, which are likely the most challenging environments, are in the order of 1-2 seconds. Thus, the maximum processing delay in data centers has to be limited to few hundred of ms in order to make the overall service set up latency acceptable.

Figure 4 shows of the measured latency (95-th percentile of SRD in ms) as a function of call throughput for both standard and JCC-based implementation. Additional working points have not been shown in the figure, since they are characterized by a call loss percentage larger than 1% and thus not acceptable. It appears that the use of JCC not only increases the call throughput by approximately 62%, but also the SRD is *always lower* than that obtained by the standard implementation. In this regard, it is worth considering that while our MCGP-SIP-JCC implementation is highly optimized for the provided functions, the SIP RA and the MGCP RA includes a larger set of capabilities and relevant control procedures (e.g. the JCC is not designed to send SIP MESSAGES). Thus, even if we have added a further abstraction layer to speed up service implementation and deployment, our implementation still definitely outperforms the SIP RA and the MGCP RA. Nevertheless, since these results are relevant to a service with complexity equivalent to that of commercial one, they demonstrate not only the feasibility of our proposal, but also the effectiveness in operation. Finally, using the JCC abstraction layer, we succeeded in implementing the same service with less code lines (the saving is about 18%), which translates also in reduced implementation effort [13][17].

V. CONCLUSION

In this paper we have presented a JCC-SIP-MGCP mapping and the relevant implementation as a JCC-SIP-MGCP RA for the Mobicents JSLEE. We have shown the effectiveness of the implemented JCC RA in allowing service developers to realize carrier-grade services in an easier and faster way with respect to creating the same services by directly using the SIP and MGCP methods. Through a higher layer abstraction, provided by the JCC RA, a complex service logic does not have to include specific SIP and/or MGCP signaling issues, and the achievable performance results appreciable.

Future work will focus on simplifying service design and implementation, integrating the JCC RA proposed here with a workflow engine to allow developers to easy build complex, stateful services using a graphic interface [18].

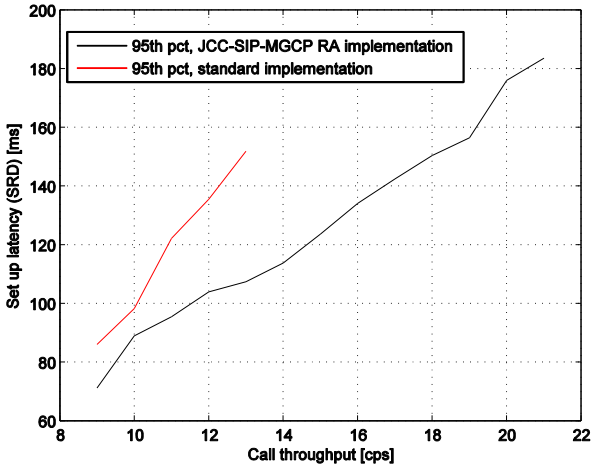

Figure 4: Set-up latency versus call throughput.

REFERENCES

[1] H. Khlifi, J.-C.Gregoire, "IMS application servers: roles, requirements, and implementation technologies," IEEE Internet Comp., 2008, 12(3).

[2] F. Andreasen, B. Foster, "Media Gateway Control Protocol (MGCP) Version 1.0", IETF RFC 3435, January 2003.

[3] Java Specification Requests, "JSR 21: JAIN JCC Specification", July 2002, http://www.jcp.org/en/jsr/detail?id=21.

[4] R. Jain et al., "Java call control, coordination, and transactions", IEEE Comm. Magazine, January 2000, pp. 108-114.

[5] Sun Microsystems, "Java Call Control (JCC) Application Programming Interface (API) Version 1.0b, Overview of the API", 29 July 2002, http://www.argreenhouse.com/JAINRefCode/jcc-overview-1.1.doc.

[6] H. Sasaki, J. L. Bakker, P. O'Doherty, "Java Call Control v1.0 to Session Initiation Protocol Mapping", White Paper, Oct 2001, http://java.sun.com/products/jain/JCC2SIP.pdf.

[7] R. Jain, J.-L. Bakker, F. Anjum, "Java Call Control (JCC) and Session Initiation Protocol", IEICE Transactions on Communications, E84-B(12), December 2001

[8] Open Cloud Web Site, http://www.opencloud.com.

[9] Amdocs Web Site, available at: http://www.amdocs.com/Products/Service-Delivery/Convergent-Service-Platform/Pages/index.aspx.

[10] Mobicents Web Site, http://www.mobicents.org.

[11] J. Rosenberg et al., "SIP: Session Initiation Protocol", IETF RFC 3261, June 2002.

[12] M. Handley, V. Jacobson, C. Perkins, "SDP: Session Description Protocol", IETF RFC 4566, July 2006.

[13] M. Femminella, F. Giacinti, G. Reali, "An extended Java Call Control for Session Initiation Protocol", IEEE MultiMedia, in press, DOI: 10.1109/MMUL.2011.58.

[14] M. Femminella, E. Maccherani G. Reali, "Performance Management of Java-based SIP Application Servers", IFIP/IEEE IM'11, Dublin, Ireland, 2011.

[15] SIPp Traffic Generator, available at: http://sipp.sourceforge.net.

[16] K.-Y. Chen, J.M. Chang, T.W. Hou, "Multi-Threading in Java: Performance and Scalability on Multi-Core Systems", IEEE Trans. on Computers, 60(11), November 2011, pp. 1521-1534.

[17] M. Femminella, E. Maccherani, G. Reali, "Workflow Engine Integration in JSLEE AS", IEEE Comm. Letters, 15(12), 2011, pp.1405-1407.

[18] D. Vingarzan, P. Weik, "IMS signaling over current wireless networks: experiments using the Open IMS Core", IEEE Vehicular Technology Magazine, March 2007.

[19] F. Cuervo et al., "Megaco Protocol Version 1.0", IETF RFC 3015, November 2000.