Software and Middleware Technologies based on Open APIs and Protocols for modern Service Provision in Telecoms

Nikolaos D. Tselikas
Department of Telecommunications Science and Technology
University of Peloponnese
Tripoli, Greece
ntsel@uop.gr

George S. Tselikis
4Plus Technologies S.A.
Athens, Greece
tselikis@4plus.com

Nikos C. Sagias
Department of Telecommunications Science and Technology
University of Peloponnese
Tripoli, Greece
nsagias@ieee.org

***Abstract*— The presence of software and middleware technologies based on open APIs and protocols for advanced service provision in telecoms is the main subject of this article. Specifically, the role and the trade-off between open APIs and Protocols, i.e. OSA/Parlay APIs, JAIN APIs and SIP, in modern telecoms are addressed. We present a technical implementation analysis, based on a call-related telecom service, in order to set a common basis for the aforementioned technologies, since – either way – "voice" is still a common denominator for Fixed and Mobile Operators as well as for Internet Service Providers too. We summarize with a performance evaluation study regarding the implemented services.**

Keywords-service provision; middleware technologies; open APIs; OSA/Parlay; JAIN; SIP; performance evaluation;

I. INTRODUCTION

The persistent demand of more and richer value added services is an exceptional revenue opportunity for network operators. Traffic flowing through their network will increase as more and more services for end users become available. However, network operators are not able by themselves to deploy in their networks services as fast as demand requires; core network interactions tend to be complex, so integrating services right in the network core and, most importantly, requiring network operators to administer and maintain those services after deployment, inevitably becomes a slow and cumbersome procedure. Rapid service creation and deployment is achieved only if service development, deployment and administration are distributed among external service providers. The only way to achieve such a distribution is for network operators to expose parts of their networks core functions to third-party providers.

Providing access to core network functions requires compromises from both sides, in order to address the concerns raised by both of them. Network operators are concerned about security and stability, so they require control on the admitted interactions, while service providers are concerned with the effort required to develop new services and whether their investment in the developed software will be reusable in other operators. In the interests of both, interaction with the network should be simple to comprehend and implement, it should require minimal development and integration effort and the result should be reusable, in order to protect the investment.

The proper way to address these concerns is by drawing clear and simple interfaces between operators and providers. These interfaces project a simplified model of core network functions and are standardized to allow reusability and independence from underlying network architectures. Such interfaces can be implemented choosing from a variety of communication middleware technologies; most notably distributed object based systems (RMI, CORBA, SOAP with RPC semantics) or messaging frameworks like Java Message Service (JMS).

The primary concern in this approach has always been the performance impact inflicted by the additional middleware layers compared to the straightforward case of deploying services straight into the core network. To this purpose, a performance comparison of various middleware technologies supporting open interfaces is an interesting step before choosing the most appropriate communication middleware to implement an open interface for a specific value added service. This is also the objective of this paper; to provide a deep investigation and an efficient performance comparison between several similar middleware systems based on open APIs and Protocols for Service Provisioning.

The rest of the paper is organized as follows. The second section cites the state of the art of the most useable open APIs and protocols in service provision. The third section analyzes a part of the implementation of an Advanced Call Control Gateway, by trying to set a common basis, in order to render feasible a performance comparison between different platforms and technologies, through a call-related service and how this service can be implemented based on different open APIs and middleware technologies (i.e. OSA/Parlay API, JAIN API and SIP), since "voice" is for sure a common denominator for either Fixed or Mobile Operator or an Internet Service Provider. The paper is summarized with the most remarkable conclusions.

II. OPEN APIS AND PROTOCOLS IN TELECOMS

Last years, a new kind of player in Telecom Market has been appeared into the foreground. Apart from the Network Operator, which used to own both the core Network and the Services, Independent Service Providers (ISPs) are playing an increasing role in modern telecommunications by using existing network infrastructures to provide services under their own management. As a result, the final link of the telecommunication market's chain, i.e. the users, are theoretically able to select the services and applications they really need or like among a big range of services offered by different Service Providers and Network Operators.

In order satisfy users' demand for new and advanced services, a different approach has been followed in the last years regarding service provisioning. This new approach is not another vertical one, but tends to be applied as horizontal as possible, covering and hiding the underlying network peculiarities in the service plane by exposing only the actual necessary functionality for creation and provision of new services. This is offered by the so-called Service Platforms, which operate over different network infrastructures. They are responsible for service deployment, manipulation, control and provision, while they provide the basis for external service creation by Independent Service Developers and Providers. Service platforms achieve communication transparency by exploiting middleware technologies like CORBA, Java-RMI, Web Services technologies etc. Moreover, a standards-based approach in such a platform is imperative for providing Independent Service Providers with homogeneous access to the underlying network resources. These standards must be open, flexible and easily programmable for everyone, thus heavy and maladjusted protocols are avoided in Service Platforms implementations. On the other hand, open Application Programming Interfaces (APIs), such as OSA/Parlay and Java APIs for Integrated Networks (JAIN) or standard Internet-based Protocols like Session Initiation Protocol (SIP) are preferred in many implementations, due to the pre-mentioned advantages [1].

OSA/Parlay constitutes a prominent forum that acknowledges the importance of designing APIs for telecommunications capabilities (The Parlay Group). The Parlay APIs specifications have been developed and defined, allowing services and applications to access transparently the core network functionality. In mobile world, 3rd Generation Partnership Project (3GPP) adopted Parlay specifications preparative to specify, define and create Parlay-like APIs for the support of service development on top of mobile networks [2]. This initiative is known as OSA Interface, or just OSA (Open Service Access). After the release of Parlay APIs version 3.0, Parlay and OSA amalgamated in the same forum. That's why it is nowadays known with the combined name "OSA/Parlay". OSA/Parlay provides 3rd parties with programming interfaces (both IDL and WSDL) that are open, language and platform independent, and include security provisions. Thus OSA/Parlay APIs can be supported on top of various middleware technologies and protocols such as CORBA (Common Object Request Broker Application), DCOM, Java-RMI and Web Services with Simple Object Access Protocol (SOAP).

Java APIs for Integrated Networks (JAIN) is defined and specified by a large number of participating telecommunication firms, the so-called JAIN Community [3]. The JAIN Community envisioned the creation of a number of open Java APIs that abstract the details of networks and protocol implementations, in order to ease the development of portable applications. JAIN provides a Java-based framework to build and integrate services and solutions that span across both packet- and circuit- switched networks. This was the major scope for the creation of the JAIN Protocol Experts Group (PEG). JAIN PEG focuses on developing Java APIs for protocols used in telephony, intelligent networks (INs), wireless networks, and the Internet. JAIN PEG is organized into two major divisions; the Signaling System No. 7 (SS7) subgroup and the Internet Protocol (IP) subgroup. The former focuses on developing Java APIs for SS7 technology – mainly used in telephony, IN, and wireless networks, while the latter focuses on developing Java APIs for Internet technologies. Within each subgroup there are Edit Groups that focus on specific protocols. Inspired by OSA/Parlay forum, but strictly in the context of the Java language, the main objective of JAIN is to provide service portability, convergence, and secure access (by services residing outside of the network) to such integrated networks, while the ultimate target of the JAIN Community is to create an open market for services across integrated networks using the already widely accepted Java technology [4].

Session Initiation Protocol (SIP) is an application layer text-based protocol standardized by the Internet Engineering Task Force (IETF) in early 1999 [5]. It is used for session initiation, modification and termination between two or more terminals. Applications based on SIP focus on interactive multimedia sessions, such as Internet phone calls or multimedia conferences, while it can also be used for instant messaging, event notification or managing other session types, such as distributed games. In setting up sessions, SIP acts as a signaling protocol, offering services similar to telephony signaling protocols such as Q.931 or ISUP, but in an Internet context. SIP uses also existing IETF protocols to support various applications (e.g. voice, presence and call control). In combination with the Session Description Protocol (SDP) [6], SIP can describe the session characteristics, while, at the same time, signaling and media streams are separated. Due to its flexibility, many extensions have been proposed for enhancement of the supported functionalities. SIP, in conjunction with the proposed extensions, supports many call control services, such as Call Fordwarding, Call Transfer, Call Hold, Call Waiting, Call Identification, Conferencing and Third Party Call Control, as well as new Internet-based services like Click-to-Dial, capability exchange, distributed gaming, messaging and presence. Because of its increased simplicity in implementation – in contrast to H.323 – SIP is the dominant architecture for VoIP telephony [7].

Open APIs and standard protocols have already penetrated in the Telecom Market. Vendors such as Ericsson, Siemens, Alcatel and Aepona have already incorporate OSA/Parlay APIs in their Service Platforms, i.e. Jambala, @dvantage, A-8601 and Causeway Parlay Gateway respectively [8-11]. Open Cloud and Sun implement JAIN in their corresponding Service Platforms Rhino and jNETx, while the latter actually implements both JAIN and OSA/Parlay APIs in parallel [12]. On the other hand, SIP is the referential protocol in the IP Multimedia Subsystem (IMS), which is an architectural framework for delivering internet protocol (IP) multimedia to mobile users. IMS was originally designed by the wireless standards body 3rd Generation Partnership Project (3GPP), and is part of the vision for evolving mobile networks beyond GSM and GPRS, such as WLANs, 3G and fixed line. All the aforementioned vendors provide in parallel SIP-based service platforms. Siemens (Nokia Siemens Networks since 2007) has extended the Parlay @vantage Service Platform to IMS@dvantage supporting SIP too [9].

Such service platforms – or even middleware implementations developed from scratch – based on SIP or on open APIs like OSA/Parlay or JAIN, are used to create a cost-effective and highly-flexible IP-based infrastructure, in order to deliver revenue generating services for the convergence of data, voice and mobile network technologies, as required in Next Generation Networks.

Regarding the Network Operators' and Service Providers' concern of making new and advanced services in an open, easier and more rapid manner, this kind of approach seems suitable [13]. However, there are performance implications regarding middleware and the implemented open-API, when a new service is deployed towards the end-users. Subscribers are always impatient. They demand services that respond as fast as possible with the minimum possible delay. But, advanced services may include several interactions between different network and application entities, which are – sometimes – generated in real time introducing big delays.

III. Implementation Aspects

In this section we describe and analyze a middleware implementation based on a subset of all three pre-mentioned technologies, i.e. OSA/Parlay APIs, JAIN APIs and SIP, regarding call control functionality. The subsets of APIs are the Generic Call Control (GCC) API and the JAIN Call Control (JCC) API for OSA/Parlay and JAIN respectively. Performance evaluation results are also analyzed, in order to define whether (or not) an open API based middleware solution or SIP can be a consistent advocate of performance. A simple call-related service is examined to indicate the service response time by using the pre-mentioned solutions. The performance analysis is not based on the service processing time (which depends directly on the service logic and thus is varying), but on the estimation of the raw performance of an open API based middleware or SIP stack implementation. This is actually the reason for not examining a more complex and advanced service. The observation of a limited performance for such a simple call-control service renders prohibitive the future expectations for real advanced services based on open API middleware solutions or SIP. Contrariwise, positive results can provide further boost in this service engineering trend.

Based on the rationale described in the previous section, different implementations – realizing an advanced Call Control Gateway, based on different open APIs or SIP –will be presented, analyzed and evaluated. The advanced Call Control Gateway relies in parallel on all three types of Call Control, i.e. OSA/Parlay Generic Call Control, JAIN Call Control APIs and SIP Call Control. All three implementations have been tested regarding their performance over a commercial Vocaltec's Softswitch, for applying call related services in a pure SIP-based VoIP network [14]. Similar Gateway implementations can be applied for both PSTN and PLMN access nodes, to certify the layer of abstraction provided by the usage of open APIs in fixed or mobile communication networks. Figure 1 presents a high level view of the service layer used during our experiments for the VoIP network.

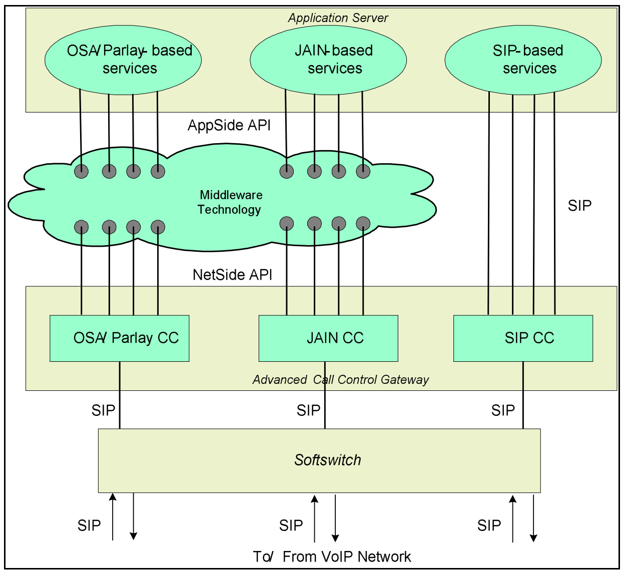

Figure 1: A high-level view of the reference implementation

The Call Control Gateway mediates between the VoIP Softswitch and the Application Server and communicates via SIP with the former and via open APIs (OSA/Parlay and JAIN respectively) or directly via SIP with the latter. For the experimental purposes, services residing in the Application Server are triplicated supporting OSA/Parlay, JAIN and SIP.

The first part of the functionality of the advanced Call Control Gateway adopts the OSA/Parlay Generic Call Control Service and specifically the Generic Call Control API. The selection of Generic Call Control API (version 4.1) depends on the requirements of the services that are going to be offered. Since the target is not to provide a very complex service, but to evaluate the performance of the open API middleware implementation, the implementation of a simple service is more than enough for the experimental part of the study. For offering more advanced and complicated services Multiparty or Conference Call Control API can be implemented instead of Generic Call Control API.

OSA/Parlay APIs are divided in two parts, called "sides", namely the Network Side API and the Application Side API respectively. The former is implemented in the Gateway, while the latter is implemented in the Application Server and is offered by the actual services. The Call Control model of OSA/Parlay is based on the traditional IN call model. The requirement for the implementation of the Generic Call Control API is to provide two interfaces on the Network Side, namely the `IpCallControlManager` and the `IpCall`.

From an implementation point of view, the two Network Side interfaces can be considered as two CORBA or RMI servers exposing their methods to the Application Part, since OSA/Parlay provides the corresponding interfaces' definitions in IDL (Interface Definition Language). They can also be considered as Web Services, since the same interfaces are also available in WSDL (Web Service Definition Language) by OSA/Parlay. Actually, the two Network Side objects (`IpCallControlManager` and `IpCall`), constitute the OSA/Parlay part of the Call Control Gateway, which is responsible to provide an OSA/Parlay-oriented view of network resources and observe the respective Application Side OSA/Parlay objects as foreseen by the signatures of their methods. The latter are the `IpAppCallControlManager` and `IpAppCall`, exposing the callback methods to the Network Side ones. Briefly, the most common of them are: the `IpAppCallControlManager::callEventNotify()` method, which notifies the application of the arrival of a call related event, the `IpAppCall::routeRes()` method, indicating that the routing request to the destination party was successful, and finally the `IpAppCall::routeErr()` method, which indicates that the routing request to the destination party was unsuccessful, as well as the corresponding reason. Figure 2 gives the picture of the OSA/Parlay part of the advanced Call Control Gateway in terms of a UML class diagram, while Figure 3 depicts the sequence diagram of the implemented "Call Forwarding Service", which clarifies the interaction between Generic Call Control Interface's objects.

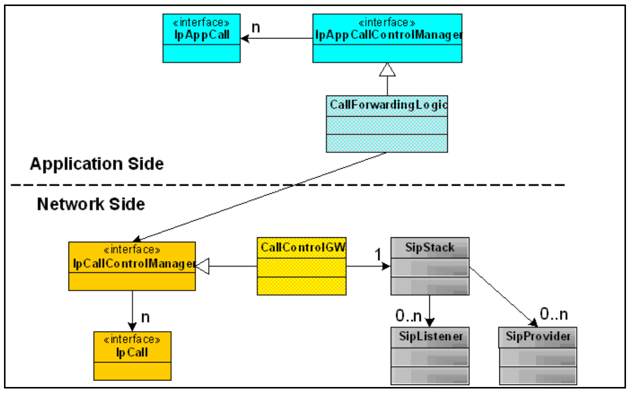

Figure 2: UML class diagram of OSA/Parlay GCC API implementation

Further technical details regarding the afore-mentioned implementation as well as the implementation of the rest part of the Call Control Gateway implementation, can be found in [14].

IV. PERFORMANCE EVALUATION STUDY

This section is a kind of performance evaluation for the different service implementations. An indicative criterion useful for a direct comparison between all the three afore-mentioned implementations is the mean value of the measured total time for a whole life cycle of a service request ($\overline{t_{total}}$). The whole life cycle of a service request is defined as the required time period assuming as start-time-point the arrival of a SIP request in the Softswitch and as corresponding end-time-point the successful response (by the Service Logic) arrival in the Softswitch respectively. $\overline{t_{total}}$ is estimated for the following cases, as described in the previous subsections, i.e. OSA/Parlay API using CORBA, OSA/Parlay API using RMI, JAIN API using RMI and plain SIP implementations respectively. The used model for the generation and arrival of service requests is based on the Poisson distribution.

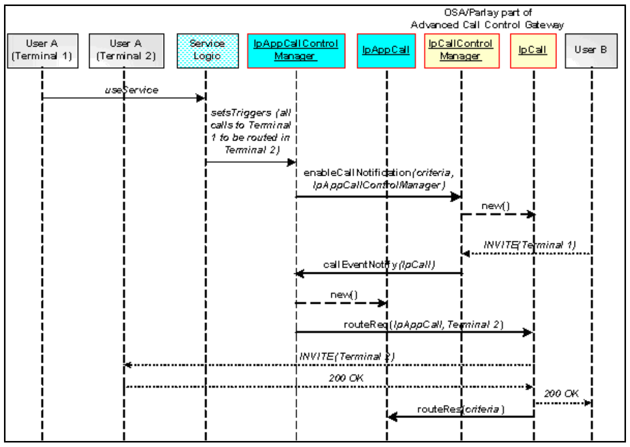

Figure 3: UML sequence diagram of OSA/Parlay-based execution of "Call Forwarding" service

Five reasonable classes of different request rates have been taken into account, representing the λ-value of Poisson rate. These are 20, 40, 60, 80 and 100 service requests per minute respectively. It is mentioned that each experiment lasted about three hours, in order to conclude to as much reliable results as possible, using the Poisson distribution.

The cluster columns chart in Figure 4 presents the mean total serving time of each one of the five different rates' experiments for all four cases (usage of OSA/Parlay Call Control API over CORBA and RMI respectively, usage of JAIN JCC API over RMI and usage of plain SIP). For lower traffic loads (i.e. 20, 40 and 60 requests per minute) JAIN JCC API over RMI is presented as the most efficient implementation concerning $\overline{t_{total}}$, while both SIP and CORBA implementations of OSA/Parlay GCC API are following and the corresponding RMI-based comes last. For higher traffic loads (i.e. 80 and 100 requests per minute) SIP implementation seems to be the most efficient, followed by JAIN and OSA/Parlay over CORBA and RMI in the row.

An important observation is that for all three middleware based implementations, i.e. the implementations based on OSA/Parlay and JAIN API, $\overline{t_{total}}$ is increased when the arrival request rate is increased. Exactly the opposite happens in SIP case. How can this paradox be explained? The communication mechanism in all four cases has the same basis. For the communication establishment over the RMI communication link, actually JRMP is used, which is TCP/IP based. This communication bus is used in two out of four experimental cases (OSA/Parlay GCC API over RMI and JAIN JCC API over RMI). In the third one (OSA/Parlay GCC API over CORBA), the CORBA bus of IIOP is used, which is also TCP/IP based.

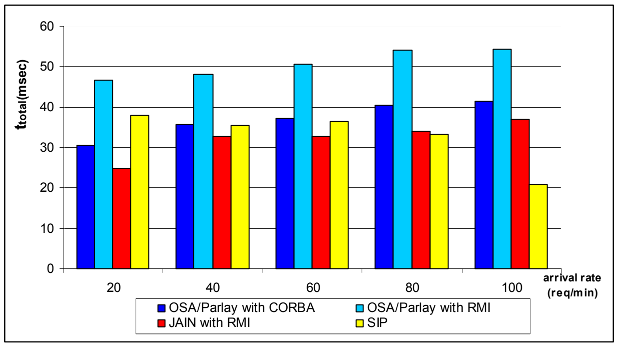

Figure 4: Mean Total Serving Time ($\overline{t_{total}}$)

Finally, in SIP case TCP sockets are also used for the communication. So the answer is not hiding in the communication mechanism, but probably in the time required in the three middleware-based cases to construct and de-construct the SIP INVITE messages to OSA/Parlay and JAIN methods and vice versa. JAIN JCC over RMI case needs less time to construct the new structure of the JCC API method against the corresponding of OSA/Parlay GCC API over RMI one. This can be explained, because OSA/Parlay APIs' methods contain more complicated data types and structures than the ones JAIN JCC API uses. On the other hand, JAIN JCC API is actually a Java-oriented implementation of OSA/Parlay GCC API, thus it is expected to be designed for RMI usage (RMI is a Java product too), while OSA/Parlay does not indicate a particular underlying middleware technology. In SIP case, no such a construction or de-construction is needed. Furthermore, RMI and SIP present better performance, because a more efficient socket manipulation policy is used versus CORBA. A new RMI or SIP connection can either open a new socket or reuse an already opened one, which is currently idle [15]. An already used RMI-socket stays alive for a few seconds. If – during this time – a new RMI connection is required, the already alive socket can be reused, saving time and recourses, instead of opening a new one. Moreover, the possibility to meet a lot of sockets alive is higher, when a lot of connections have been already established. This is probably another reason justifying the better performance of JAIN and SIP. Contrariwise, in CORBA case, the underlying socket is destroyed after the usage, thus every new request arrival requires the generation of a new socket. Consequently, more time is spent for higher traffic loads.

V. CONCLUSIONS

The paper describes the implementation of telecom services based on open interfaces and standard protocols exposing call control network functions. The current trend on service provision in telecoms was presented. Finally, some basic implementation issues as well as a performance evaluation study based on mean values of time measurements were also analyzed.

The most important conclusion one can draw from the performance analysis is that the performance impact inflicted by the use of a middleware layer implementing open interfaces between the network and services is – without any doubt – an acceptable overhead compared to the indisputable gains of exposing network functionality through Open APIs. The use of standardized interfaces to interact with the network is a significant factor contributing transparency, modularity, reusability and clear distribution of administrative responsibilities to the service development process. This bears significant gains for all those involved; network operators increase the traffic and utilization of their networks through the use of novel services developed by 3rd parties; the simplified development and deployment process for services – transparent to the network – spurs activity in the service provider world and – as a consequence – users enjoy a large variety of value added services.

REFERENCES

[1] Carvalho, R., P., & Alberti, A., M., "Java technologies for NGN service creation: discussion and architecture to improve SIP addresses discovery," Internet and Multimedia Systems and Applications Conference 2007, pp: 56 – 62.

[2] 3GPP: www.3gpp.org

[3] Java Technology, The JAIN initiative: http://java.sun.com/products/jain/

[4] Tait, D., Keijzer, J.D., Goerdman, R., "JAIN: A New Approach to Services in Communication Networks," IEEE Communications Magazine, Vol. 38, Issue 1, Jan. 2000, pp. 94-99, doi: 10.1109/35.815458.

[5] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston A., Peterson, J., Sparks, R., Handley, M., Schooler, E., "SIP: Session initiation protocol," RFC 3261, Internet Engineering Task Force, 2002.

[6] Handley, M., Jacobson, V., Perkins, C., "SDP: Session Description Protocol," RFC 4566, Internet Engineering Task Force, 2006.

[7] Papadakis, A. E., Chaniotakis, E .S. ,Giannakakis, P. E., Tselikas, N. D., Venieris, I., "Parlay-based service provision in circuit- and packet-switched telecommunications networks," International Journal of Communication Systems, Vol. 17, Issue 1, 2004, pp. 63-83, doi: 10.1002/dac.631.

[8] Ericsson Jambala Parlay SCS, at: www.ericsson.com

[9] Nokia Siemens Parlay@vantage:http://www.nokiasiemensnetworks.com

[10] Alcatel 8601 Parlay/OSA Gateway, at: www.alcatel.com

[11] AePONA Causeway, at www.aepona.com

[12] SUN jNETx OSA Platform at www.sun.com

[13] Moerdijk, A. & Klostermann, L., "Opening the networks with OSA/Parlay APIs: Standards and aspects behind the APIs," IEEE Network Magazine, Vol. 17, Issue 3, May/June 2003, pp. 58-64, doi: 10.1109/MNET.2003.1201478.

[14] Tselikas, N. D., Dellas, N. L., Koutsoloukas, E., Kapellaki, S., Prezerakos, G. N., Venieris, I., "Distributed service provision using open APIs-based middleware: "OSA/Parlay vs. JAIN" performance evaluation study," Journal of Systems and Software Vol. 80, Issue 5, May 2007, pp 765-777, doi:10.1016/j.jss.2006.06.035.

[15] Stefano, C., Heikki, H., Oskari, K. , "Performance enhancing proxies for Java2 RMI over slow wireless links," Second International Conference and Exhibition on The Practical Application of Java (PA JAVA2000), 12-14 April 2000, Manchester, UK, pp. 76-89.