Smart Middleware for Mutual Service-Network Awareness in Evolving 3GPP Networks

Network Aware Applications & Application Aware Networks

Sahin Albayrak, Muslim Elkotob, Ahmet C. Toker
DAI-Labor
Technische Universität Berlin
Berlin, Germany
{sahin.albayrak, muslim.elkotob, ahmet-cihat.toker}@dai-labor.de

***Abstract*— We have designed a smart middleware and are working on implementing and deploying it as a landmark architectural piece run by nodes of the Future Internet. The middleware we propose aims as a main goal at making applications and services fully network-aware in the sense that they utilize network resources and tune their own demands based on what the underlying networks can offer. On the other hand, the smart middleware makes networks service aware in the sense that networks also adapt their configuration to service demands. Challenges of the Future Internet such as scalability, increased interoperability, and smart collaboration between engines are addressed by our middleware. Focusing on 3GPP networks including IP Multimedia Subsystem (IMS) networks, this paper presents the architectural design of our middleware and shows in two showcases how and why it is capable of addressing future challenges. Scenarios such as service and network-aware coordinated load balancing and "always-best-connected" as a service verify the merits of our smart middleware design.**

Keywords-component; service-aware networks, network-aware services, middleware, smart middleware, 3GPP, IMS.

I. INTRODUCTION

As networks evolve today, new types of applications and services emerge. Moreover, the demand for higher dynamic support that copes with changing application and service requirements is gaining importance. Several approaches to tackle this challenge have been adopted in research for quite a while. One alternative is overlays. Other approaches for matching application and service requirements with available network capabilities in the telecommunication domain are abundant as well. Examples are OSA-Parlay [11], the Open Mobile Alliance (OMA) [12] approach, Parlay-X [13], and different middleware alternatives.

We firmly believe that a new middleware architecture with innovative aspects in terms of: full support along the whole path rather than at the front and backend nodes, highly service-aware networks, network aware services, and intelligent coordination and cooperation capabilities is the right answer to the upcoming challenges in next generation networks.

In [1] Ginis et al talk about "Autonomic Middleware". They claim their middleware to possess a series of autonomic capabilities such as self-optimization, self-deployment, and automated fault tolerance. The system they propose mainly bases itself on "smart monitoring" which can also be offered to external entities as a separate functionality. The authors at IBM Research use the middleware in question for optimizing environments which heavily perform media streaming and act as Content Delivery Networks (CDN). They define two main services whose behavior possesses a certain degree of autonomy: the Placement Optimization Service and the Deployment and Monitoring Service. The Placement Optimization Service performs estimates related to traffic (packets, flows) and message rates based on a particular network topology, available brokerage nodes, client locations and numbers, and so on. Then distribution of computational load is done according to the collected and deduced information (topology, expected traffic flow, etc.). The other service is the Deployment and Monitoring Service. It executes the placement decisions determined by the placement optimizer. Upon changes in load (requests, subscriptions, packet and message traffic, etc), the module redistributes the intelligence and computational tasks as well as the control logic (decision-making) by displacing modules based on which current configuration the system is in.

In [2], Cho presents middleware as a solution for the interoperability problem between several electronic devices. Electronic devices possess different computation and communication capabilities. In the proposed architecture, a unit called a middleware manager controls several other units in a centralized fashion. Global information is collected about the ambient (home) environment and then decisions are made centrally with distribution tasks for processing and messaging assigned by the middleware manager to different entities in the home network.

Universal Plug and Play (UPnP) [6] is a set of computer network protocols promulgated by the UPnP Forum. The goals of UPnP are to allow devices to connect seamlessly and to simplify the implementation of networks in the home (data sharing, communications, and entertainment) and corporate environments. UPnP achieves this by defining and publishing UPnP device control protocols built upon open, Internet-based

communication standards. The reference to this initiative and to its forum can be located at [6].

Jini network technology [7], which includes JavaSpaces Technology and Jini extensible remote invocation (Jini ERI), is an open architecture that enables developers to create network-centric services. They are either implemented in hardware or software and are highly adaptive to change. Jini technology can be used to build adaptive networks that are scalable, evolvable and flexible as typically required in dynamic computing environments.

The Common Object Request Broker Architecture (CORBA) [8] from the Object Management Group (OMG) [9] is the most classical middleware architecture which has evolved over the years to cope with the growing demand on interoperability between systems. CORBA is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together. This principle has been kept in this fashion without adding any network-aware or service-aware aspects, but rather attributing this privilege to software components written in various languages.

II. MOTIVATION

Taking a look at the state of the art, we observe a multitude of approaches for middleware solutions. What we observe is a lack of network awareness from the side of services and the lack of service awareness from the side of networks. Middleware has so far provided abstraction for interoperation of different entities but not considered different view points at the same time; e.g. service provider, user, and network operator.

Another point is that middleware solutions have functioned in an end-point fashion so far. One end point is the client and the other is the middleware container entity, usually a backend box. We tackle this issue by making all nodes in our environment service engines running our middleware and thus able to communicate, co-operate, and provide the necessary middleware support along the whole data and service path.

We address those challenges with our middleware by collecting different application and service requirements and translating them into consequences and configurations for the underlying networks. At the same time, the middleware we propose closely monitors network resources and capabilities and makes the applications and services on top aware of the occurring changes in network conditions.

Our paper is organized as follows. After this introductory section we move on to describe the architecture both at system level and at node level. Then we go through the different features of our middleware and propose a deployment scenario consisting of an evolved 3GPP data network with IMS as the service delivery platform. Afterwards, we present two case studies with technical argumentation justifying how our middleware architectural approach suits real-life situations in networking and addresses future challenges. Finally we wrap up with an outlook and conclude this paper.

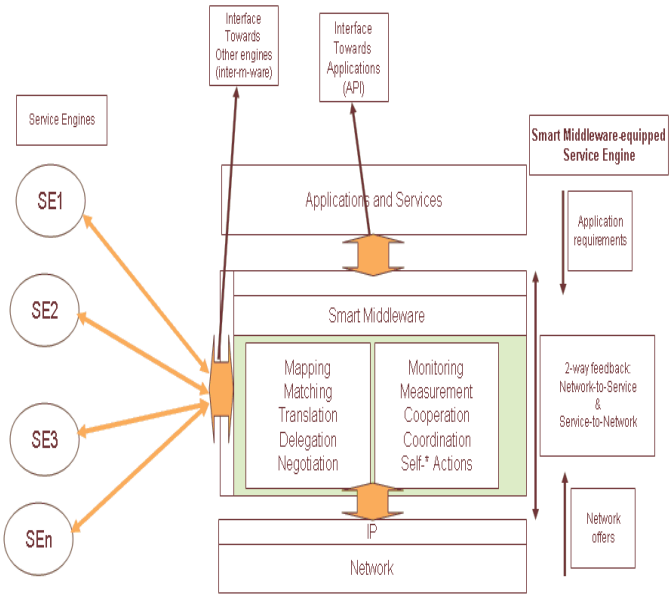

Figure 1Abstract architecture

III. ARCHITECTURE

A. *System Architecture and Features*

A high-level overview of the architecture is provided in Figure 1. Each node in our future internet vision which runs the our smart middleware will be able to cooperate with other peer nodes, which can either be terminals, servers, access routers on a middleware level. This cooperation, coupled with the intelligence that our middleware adds to the node, without changing the actual node implementation, allows applications and services to be network aware. If the node developers are willing to extend their implementations in a way that utilizes services provided by the middleware, the network nodes can be made service aware. Thus our middleware is capable of making

- the applications network aware
- the networks service aware

We now discuss these properties in more detail, and try to isolate the capabilities of the middleware that realize the aforementioned properties.

1) *Network-Aware Services*

The middleware monitors the state of the communication on different layers. For this it has to communicate with the kernel on the user and kernel level. Monitoring involves the collecting information from underlying networks about their status in terms of available resources and low-level network parameters. The collected network related information is provisioned to the related applications in a standard and structured format. According to changes in these collected network information the adaptation of the application as an answer to changes in the network layer is the duty of the application. Yet the middleware can be loaded with policies, which makes adaptation possible for network agnostic applications.

Information provided towards the applications can be given after explicit queries, subscriptions or interrupts. Interrupts that the middleware forward to the matching application paves the wave for self-aware applications where the application is aware of the node status on which it is running. Information provided towards the applications includes the feedbacks of requests on the network layer earlier, which have been delegated by the middleware.

Applications and services delegate downwards to the middleware their requirements in terms of network usage, required quality, level of security, and specific personalized settings related to the service in question. The applications, in addition to accessing the network information through a standardized format, are also able to request the network node to perform certain allowed actions which would change the resource allocation in the network node through middleware. This delegation is also provided in a standardized way. One can say that our middleware provides a virtual network node to the applications, whose functionalities and internal properties are accessible through a structured and standard format.

The components of virtualization include:

- Standard API
- Meta description of application capabilities, requirements and properties

The standard API is in fact an interface towards applications and services to be able to read their demands and provide them with feedback coming from the middleware. Moreover, autonomic self-* aspects are a crucial feature of a

2) Service-Aware Networks

Middleware also provides information towards the network protocols, if they are interested. These information pieces include, but are not limited to service classes, service requirements. By making use of service data, and also using network context data organized in a structured way and made easily accessible, the network node can be self aware and realize various self-* aspects. These include

- *Self-monitoring*

A system is self-monitoring when it is capable of measuring the different parameters which determine its state in terms of available resources, functioning modules, and general status. For self-monitoring, information collection as well as sensing and detection are essential.

- *Self-healing*

This feature corresponds to the ability of a system to recover from faults, drops, crashes (mainly software), and deadlocks. This is crucial for nodes on which many other nodes depend. For instance a core router serving many access routers has to possess recovery and self-healing capabilities otherwise all attached access routers will suffer for a long time if no fast recovery after a system fault or crash takes place.

- *Self-organizing*

This term from autonomic communication can be applied to several contexts. A self-organizing network is a set of nodes which is able to form a network by organizing and arranging itself accordingly. A self-organizing system is one which is able to organize its components and modules in such a way that they form consistent services and applications that match the system and user requirements and condition terms.

- *Self-awareness*

Self-awareness is very closely related to self-monitoring. It also involves a lot of self-monitoring and then includes a step further. The further step is about determining the concrete state and status of the node either from a state space by direct mapping and matching or by doing fuzzy logic and approximations. The important fact about this feature is that it adds a lot of autonomy and intelligence to the behavior of a node. A self-aware node or network is able to act intelligently. Particularly for our paper, self-aware networks and self-aware services are a key to making awareness mutual, namely: making networks service aware and making services network aware.

B. Node Architecture

In this section we introduce the proposed node architecture, which gives the nodes running the middleware the properties we described earlier such as simultaneous network and service awareness, virtualization and scalability. We start exploring the components of the architecture from the perspective of users of the middleware, namely the applications and the operating system.

As shown in Figure 2, the applications interact with the Application Interface module. The applications provide their functional and non-functional properties, which are a part of the node context, as well as their service requirements

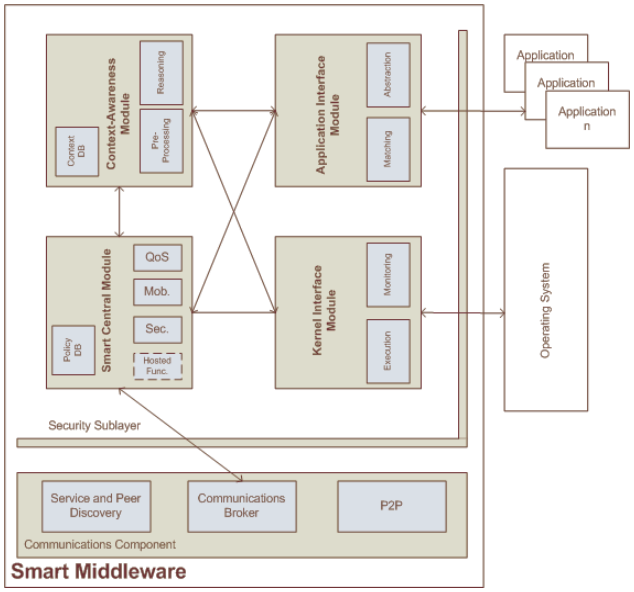

Figure 2 Abstract Architecture

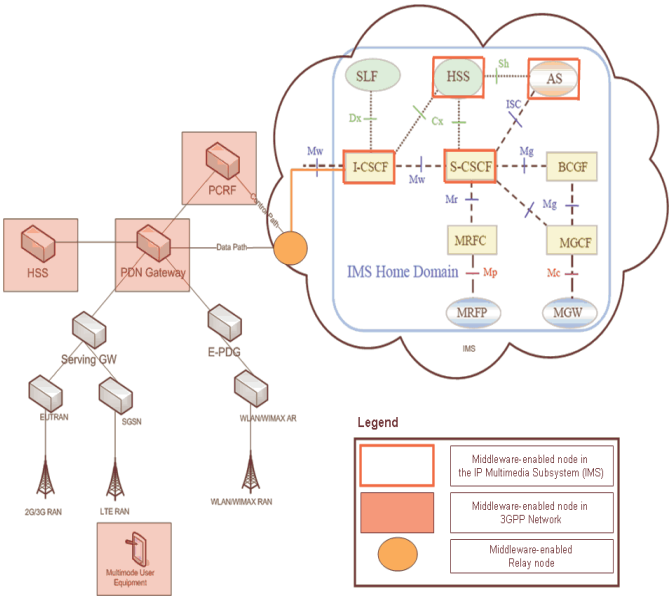

Figure 3 Deployment Architecture

The Application Interface Module is responsible for forwarding these context forming information elements to the Context-Awareness Module, which is to be described later. The Context-Awareness Module can explicitly ask for certain information elements from the applications through the Application

Interface Module. Applications are allowed to gather functional and non-functional information from the Context Awareness Module through the Application Interface Module. The applications can also request certain actions to be executed locally on the node or remotely, such as reserving of resources in an integrated services type of QoS environment or the changing of service classes in a differentiated services environment. Such requests are to be dispatched to the Smart Central Module. Matching sub-component is responsible for controlling the flow of information towards and from the applications, and act like a “dispatcher” for requests, replies and asynchronous messages exchanged between the applications and the middleware. The Abstraction sub-module presents the context information in a standard and organized format, as well as providing standard methods for accessing and modifying node internal and remote resources.

The Kernel Interface Module has a similar function, albeit it regulates the interaction with the operating system, which in turn has implementations of the networking protocols and device drivers of the installed networking hardware. The Context-Awareness Module uses the Kernel Interface Module access information offered by the operating system and device drivers, so as to be context aware on all the protocol layers that exist on the node. The Monitoring sub-module is responsible for informing the Smart Central Module for sudden changes in certain parameters or operating system interrupts, which allows the Smart Control module to demonstrate the so called self-star capabilities such as self healing. On the other direction of the information flow, the operating system may access the local and remote service and networking context information through the Kernel Interface Module. The Smart Central Module can change the values of certain operating system variables or initiate allowed kernel procedures as response to application requests, or to react to changes in local environment in a self aware manner. This is the job of the execution module.

The Context-Awareness Module is the central module which gathers local and a selected number of remote context forming information pieces, and stores in the internal Context Database. The information elements coming from the applications through Application Interface Module, cross layer through the Kernel Application Module and remote information coming from the Communications Module are pre-processed to filter irrelevant or damaged data. The processing of certain data can be influenced by the operating system or applications. For example the application can suggest averaging out the end to end delay which is originates from the operating system, or the operating system may require the maximum bandwidth request coming from the applications. Once the pre-processing is complete the plain context information is stored in the context database. Reasoning sub-module is responsible for drawing conclusions and more refined information from the plain context such as service usage statistics or predictions on local context information, so as to be pro-active. The Context-Awareness Module is also responsible for coordinating the modes of access to the information. Classical subscription based, request-reply type of access are supported. The Reasoning sub-module and Monitoring sub-module also generate asynchronous interrupts. These interrupts are sent to the Matching sub-module, to be forwarded to relevant applications and Smart Central Module for self-awareness.

The Smart Central Module is the core component of the architecture. It consists of functional sub-modules. These sub-modules are responsible for gathering application requests and local and remote networking context, comparing them and taking decisions on actions to be taken based on the policies stored in the Policy Database and taking appropriate actions.

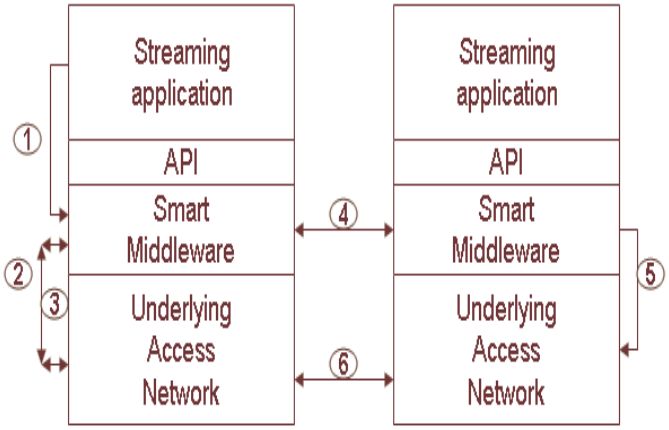

Figure 4 Applications, middleware and network interactions for load balancing

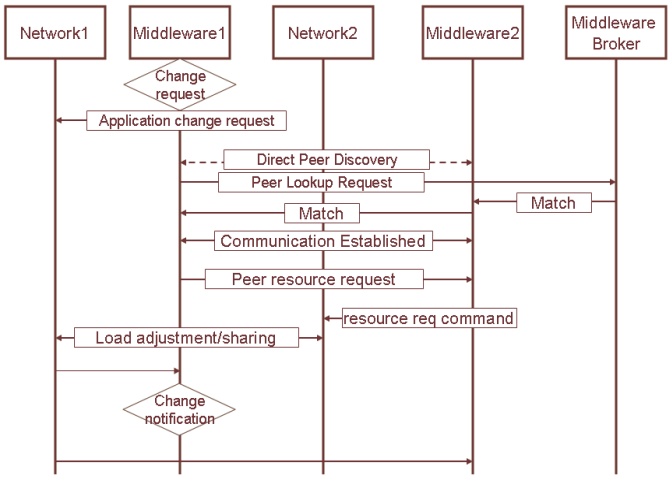

Figure 5 Load balancing message sequence chart

These actions include setting the values of certain operating system variables, calling operating system or device driver functions to change software or hardware behavior, requesting similar actions on a remote node through the Communications Module or asking the applications to change their requests. It should be noted that policy based decisions allow adaptations to the applications to be done by the applications themselves in the case of intelligent applications, or by the middleware itself in the case of context agnostic applications. In addition to standard functional sub-modules such as QoS, Mobility, Security and Routing, it is also possible to run optional functional modules, which are able to use a certain set of operating system functions and understand a set of application requirements and networking context.

Communications Module is composed of three sub-modules. The central sub-module is responsible for coordinating the communication to other nodes running the middleware. Message scheduling, message advertising, soliciting for services and answering these solicitation messages are tasks of the Communications Broker. It uses the help of other two sub-modules, namely the Service and Peer Discovery sub-module, which locates services on different peers running the middleware, and the P2P sub-module, which handles the addressing of found peers and services.

By gathering application requirements and properties, and making it available to the operating system our middleware makes the network node on which it is running service-aware. Similarly one can say by gathering local and remote networking context, and providing this to interested applications, our middleware makes them network-aware. The reasoning and monitoring properties of our middleware combined with the policy based decision making mechanism makes the nodes running our middleware self-aware and self-healing. The abstraction of the operating system functionalities and the context information provides the much needed virtualization of the networking nodes.

C. Deployment Scenario

In this section we propose how our middleware can be deployed in an evolved operator network. The deployment scenario is of a future operator network an integrated all-IP core connected to heterogeneous access networks for providing ubiquitous connectivity, and using IMS to present user with data intensive interactive multimedia services, based on the proposal given in [15], and depicted in Figure 3.

We propose our middleware running on the user equipment, as it is the only place to obtain real-time information about the networking and computing context, in which one end of the applications used is running. Furthermore we foresee our middleware running on the PDN Gateway. As described in [15] and [16] PDN gateway acts as a mobility anchor point -Home Agent in the context of Mobile IPv6- and a policy execution point where QoS policies for individual users are executed. In addition deep packet inspection, in which packets destined to users are inspected for their contents, takes place at this node. So in order to obtain detailed service, mobility and resource allocation information, and use network controlled mobility functionalities middleware should be running on PDN Gateway. PCRF, which is described in [17] is the core network node where the resource allocation decisions are taken. In addition it is the node that has access to service providing entities. In a similar reasoning, in order to obtain service and resource allocation information and make the resource allocating functionalities available to the applications our middleware should run on this node. Finally HSS where user and service relations are kept is also a suitable place for our middleware to run.

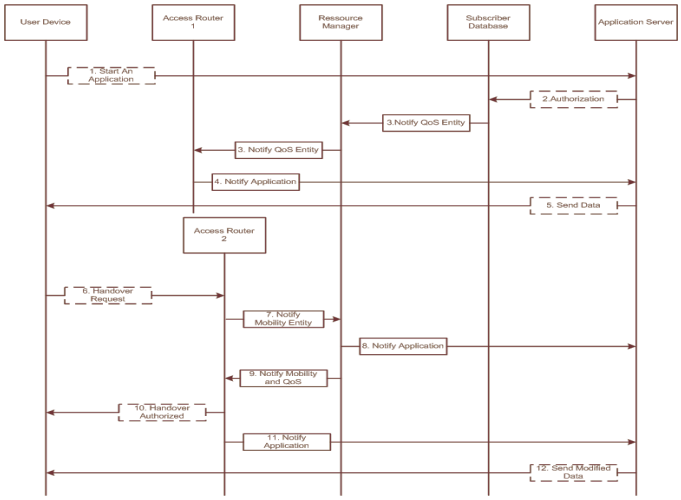

Figure 6 ABC message sequence charts

Similarly, in the IMS domain, we propose our middleware running on Application Server (AS)-where the applications are formed and launched-from the service provider-, as it is the only place to obtain real-time information about the networking and computing context, in which the other end of the applications is running. The Incoming Call Session Control Function (I-CSCF) intercepts and processes incoming calls and requests in the signaling domain. Using our smart middleware, the I-CSCF will be able to handle incoming request from peer IMS domains as well as from evolving 3GPP domains in a context aware fashion. This is because our smart middleware provides two types of awareness: service aware networks and network aware services. The Serving Call Session Control Function (S-CSCF) in IMS contains most of the proxying and call routing logic in the IMS plane. When our smart middleware is used on S-CSCF, then call routing and proxying which better match application requirements to network capabilities can be done. This increases overall system efficiency and offers better performance of the single IMS domain where the middleware over S-CSCF is deployed.

Use Scenarios

1) Application-Aware Load Balancing with Streaming in Content Delivery Networks

In next generation heterogeneous network environments, it is often the case that a mobile terminal has a choice among several networks (and network types) to get connectivity. Application demand for resources changes depending on the service usage pattern of a user, the type of subscription, the content of the data used, etc. For instance, when playing several music or video news clips from different providers, then a multi-peak curve is obtained.

In a scenario where a user requests several content pieces such as video clips for music or news, then the pattern will be a sudden increase to a certain value, then held constant for some time, then again going down while emptying the buffer. Then when the 2nd content piece is requested, then the demand then increases again in the mentioned form, and so on.

Classical adaptation mechanisms would have a hard time coping with such a pattern although it is quite common. Our smart middleware however, makes the application whose demands vary according to some part (e.g. such as the one depicted above) network aware and makes networks application aware. We come to this issue on an entity-messaging level as shown in Figure 5. Applications forward their change requests to the smart middleware which then in turn talks to the network. Moreover, different middleware entities after performing mutual discovery and establishing a link can negotiate different parameters or exchange demands and other pieces of information.

We strongly pursue the point that underlying networks are able to coordinate among themselves to improve load distribution.

The knowledge on how to distribute load, how to redistribute resources, how to grant what to whom is a feature of the middleware. As seen below, after receiving a change request, the middleware talks to the underlying network which then returns a response in terms of what can be offered and how close to the change-request the network can adapt itself .

After seeing that the network is unable to satisfy the change request desired by the application to grant it the necessary resources, the middleware issues a discovery process, finds a peer node running an instance of the smart middleware. Upon establishing a link with the peer node, the latter requests its own underlying network to share some resources and allocate them to the former requesting node on which the application is running. Once this is done, the load is cooperatively redistributed and the application demands are properly met.

2) "Always Best Connected" as a Network Service

"*Always Best Connected*" (ABC) is a term coined down by Gustafsson and Jonsson [14]. In a network providing always best connected service the mobile user is not only able to connect to his services all the time, but also using the most appropriate access technology chosen among the available heterogeneous access technologies, where the "*best*" for a user is related to his service profile and preferences. The proposed architecture in 1 requires an ABC operator undertaking different tasks required for providing the ABC service. In a joint project with Nokia Siemens Networks [15] we have explored the possibility of providing always best connected services with the help of coordinated policy based mobility decision engines running in the core network and the user equipment. In this section we try to demonstrate the services provided by the ABC operator may be considered as a distributed application made possible by middleware running nodes.

In the scenario summarized by the flow diagram in Figure 6 middleware runs on the end device, on the access routers of different access technology subnets, on the node that stores the subscription data and the application server to which the user is connected to for accessing multimedia services. In the figure the decisions and the related message exchanges that are not middleware initiated are a denoted with dotted lines. The scenario starts with the user requesting a new multimedia application from the application server. The normal procedure without the middleware would be application server contacting the subscriber service to authorize the request, and sending the data on successful authorization. In the case of the middleware enabled network the kernel interface monitor on the subscriber

database would detect this authorization request and forward it to the ABC application running on the node. The ABC application notifies the resource manager middleware about the change in bandwidth need. The resource manager has an ABC application running on it which uses node internal functionalities through the middleware to reserve resources. It also can notify the access router of the access network the user is connected to reserve radio resources. The ABC application there would again be using the node internal capabilities through the middleware. The applications would be dealing with virtualized network nodes thanks to the middleware. The application running on the application server would only start streaming data after receiving from the middleware a message, which notifies the successful reservation of the resources. This is an example how our middleware makes the networks service aware without modifying the node internal procedures. Let us now go through another scenario which demonstrates how does our middleware makes the services network aware without modifying the services.

In the second scenario let us assume the user device decided to use another access technology, as it is the best for his needs. After completing a layer 2 handover it will first try to obtain a layer 3 address from the access router 2. The ABC application running on access router 2 senses this through the middleware and notifies the mobility entity of the middleware on the resource manager. The ABC application may use the node internal functionalities to reserve resources before notifying the application about the available resources again through the middleware running on both of these nodes. The resource manager middleware again can contact the access router to reserve radio resources through the middleware. When the access router finally gives the layer 3 address, the middleware notifies the application server about the finalization of the handover and the application sends the data in a modified format which matches best to the access technologies capabilities. Note that this adaptation can also be executed by the middleware on the application service, in case of dummy applications.

IV. Summary and Outlook

We presented a middleware architecture that makes networks service aware and services network aware. The verification of the fact that this architecture of our smart middleware is solid and beneficial took place using two case studies one for load balancing and the other for “always-best-connected” for a terminal. We presented a system level as well as a node level architecture and discussed how such a middleware can be deployed and used on different types of nodes in the 3GPP and IMS domains. Bearing in mind that most of the control logic and functionality of the presented middleware is already implemented in our labs; the major next step would be to bring everything together in a systematic fashion. This would ensure the availability a consistent and running middleware that can be simply deployed and used in future networks.

References

[1] R. Ginis and R. Strom. An Autonomic Messaging Middleware with Stateful Stream Transformation; Proceedings of the International Conference on Autonomic Computing May 17-18, 2004

[2] S. Y. Cho. Framework for the Composition and Interoperation of the Home Appliances Based on Heterogeneous Middleware in Residential Networks. IEEE Transactions on Consumer Electronics, Vol. 48, No. 3, August 2002.

[3] R. Schreiber, "Middleware Demystified." Datamation 41, 6 (April 1, 1995): 41-45.)

[4] Elkotob et al. Next Generation Service Architecture: Challenges and Approaches. Sixth International Workshop on Applications and Services in Wireless Networks, May 2006, Berlin, Germany.

[5] Toker et al. Managing Heterogeneous Access Networks - Coordinated policy based decision engines for mobility management. The First IEEE LCN Workshop On User MObility and VEhicular Networks (ON-MOVE)

[6] Universal Plug and Play (UPnP) Forum: www.upnp.org/

[7] Sun Microsystems Jini Technology. http://www.sun.com/software/jini/

[8] Common Object Request Broker Architecture (CORBA), OMG site: www.corba.org

[9] Object Management Group (OMG), www.omg.org

[10] Toker et al. Network Economy in Service- and Context-Aware Next Generation Mobile Networks. 18th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Athens, Greece, September 3-7, 2007

[11] OSA-Parlay, Parlay Group site: http://www.parlay.org/

[12] Open Mobile Alliance (OMA) site: http://www.openmobilealliance.org/

[13] Telcordia Parlay-X site: http://www.argreenhouse.com/parlayx/

[14] E. Gustafsson and A. Jonsson, “Always best connected,” IEEE Wireless Communications, Vol. 10, Feb. 2003

[15] 3GPP, TS 23.402 “Architecture enhancements for non-3GPP accesses,” V1.2.0, Jul. 2007

[16] 3GPP, TS 23.401 “GPRS enhancements for E-UTRAN access,” V1.0, May. 2007

[17] 3GPP, TS 23.203 “GPRS Policy and charging control architecture,” V3.0, Jun. 2007