DESIGN AND IMPLEMENTATION OF A GENERIC SOFTWARE ARCHITECTURE FOR THE MANAGEMENT OF NEXT-GENERATION RESIDENTIAL SERVICES

Filip De Turck*, Stefaan Vanhastel, Koert Vlaeminck,
Bart Dhoedt, Piet Demeester
Department of Information Technology, Ghent University - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium.
Tel.: +32 9 267 35 87, Fax: +32 9 267 35 99
filip.deturck@intec.rug.ac.be

Filip Vandermeulen, Frederik De Backer, Francis Depuydt
Belgacom ANS/NIS Strategy Department
Koning Albert II-laan 27, B-1030 Brussel, Belgium.
Tel.: +32 2 202 10 80, Fax: +32 2 202 71 35
filip.vandermeulen@belgacom.be

Abstract: In this paper, we address the design of a generic architecture for the management of residential services. The architecture consists of components both at the customers' side as well as at the service provider's side. The key features of the architecture are service modularity, the concept of service sessions, service packaging and subscription. The architecture allows service providers and telecom operators to rapidly provide new integrated value-added services to their customers. Layer-based design ensures that the architecture is independent of the particular service and service realization technology. The architecture provides generic access session management, service session mangement, subscription management and billing. Its implementation is based on J2EE (Java 2 Enterprise Edition). The various components of the architecture will be discussed, together with the implementation issues.

Keywords: Service Management, Multimedia Home Services, Subscription Management, OSGi, J2EE

1. Introduction

Recently, there has been a growing interest shift from the provisioning of commodity services towards value-added residential services. Commodity services are mainly connectivity services such as internet access, leased lines and VPNs. The management of this type of services has been studied in extensive detail [1, 2] and various management platform solutions exist. However, the revenue for telecommunication network operators and network service providers from these commodity services is stagnating. As a consequence, residential services are gaining more and more interest.

*Postdoctoral Fellow of the Fund of Scientific Research - Flanders, Belgium (F.W.O.-V.)

Popular examples include Video on Demand (VoD), video-conferencing, e-gaming, virtual home environments and domotics systems. Due to the customer interaction, management of such residential services is more complex and critical than the management of commodity services. The design of efficient management solutions for such services has not yet been studied in enough detail. At the time of writing, different standards and systems are emerging for the delivery of residential services to customers. The following areas and industry segments in the service provisioning market can be distinguished:

(i) *Multi-party Multi-media Conferencing Systems and Applications*: traditional telecommunications equipment vendors are introducing the concept of soft switches which take over the signalling and transport functionality of traditional TDM-based PSTN networks. Soft-switches control the voice gateways at the edge of an IP based packet transport network, and translate traditional PSTN signalling into other signalling protocols (SIP [3], H323 [4], MGCP [5], Megaco [6], etc.) which are more aligned with the IP network. Soft-switches can also support these new forms of signalling directly to the consumer's appliance endpoints, such as Intelligent Access Devices (IADs) or IP telephones (either standalone or software based IP telephony clients on a PC platform such as Microsoft NetMeeting). By means of standardized and Java based APIs such as those defined by Sun's JAIN framework [7], these protocols can be controlled and handled from within J2EE-based [8] application servers.

(ii) *The Connected Home and Home Gateways*: the concept of the connected home implies a household, which is connected to the outside world via a broadband gateway terminal (i.e. a home gateway). The home gateway acts as some kind of internal driver and application management system, which is able to perform driver life-cycle management and configuration management of in-house devices and appliances. In addition, the home gateway can act as repository for content downloaded from the Internet or from network server-side service and content delivery platforms. This temporary locally-proxied content can be consumed within the house from different rooms and consumer devices (such as TV sets, PCs, PDAs, etc.), by connecting them to a multi-room multi-source system.

(iii) *Home Theatre Systems*: home theatre systems are currently entering the residential home at an increasing speed, as high-end plasma screens and projector-based systems are gradually becoming affordable for households. In combination however with network side Video-on-Demand (VoD) server platforms, home theatre systems could be used to view content streamed in real-time (e.g. via RTP). Another area where the home theatre system could be used as multimedia endpoint consumption system is online gaming.

(iv) *The Intelligent Home*: Domotics is yet another emerging industry area with increasing potential to bring added value to the home. Domotics systems allow to control systems such as lighting, heating, home surveillance, etc. More and more vendors of domotics systems provide touch-panel like controls on their system, but some of them go even further by providing IP modules that connects to the bus or the central computer of the system. Via the home gateway, the domotics system can be connected to the outside world. Through the IP

modules, the domotics system can be controlled and interfaced from anywhere. For example, events in the home could trigger the setup of a video connection from an in-house video camera to a PDA.

In each of these areas, different vendors come up with different systems and platforms that in most cases are implemented without any adherence to a standard – in so far any standard exists. None of the above areas and applications will unlock massive revenue streams in the residential market, since a key element is clearly missing in the above description. That missing element is the enabling platform that allows rendering the set of isolated potential solutions and platforms into one integrated package.
This paper describes the design and implementation of a platform which supports (i) integration of several existing stand-alone service components, (ii) generic subscription management to all services, (iii) generic access session management, and (iv) related billing. The platform consists of components both at the service provider's side as well as at the customers' side. The platform implementation takes benefit of the most recent enabling software and middleware technologies: J2EE (Java 2 Enterprise Edition) based application servers, EJBs (Enterprise Java Beans) in the backend layer, servlets in the control layer, and JSPs (Java Server Pages) in the view layer. As such, the Model-View-Control (MVC) design pattern is exploited in its most generic way. XML-RPC [9] and RMI/IIOP [10] are used as enabling middleware protocols for communication between components in the subscribers' end-terminals and the platform components. The use of J2EE for building service provisioning systems has already been reported upon in [11]. However, service subscription, access session and interaction with software components at the customers' side were not addressed in [11]. The IETF OPES (Open Pluggable Edge Services: [12]) Working group also aims at designing architectures to allow for service provisioning, but doesn't provide service providers with a platform to address all their needs.
The remainder of this paper is structured as follows: section 2 describes the key features of the platform. The platform's software architecture is introduced in section 3, whereas the μ-flow concept is described in section 4. The main components of the architecture are described in section 5. The scenario of a service access session is detailed in section 6. The database model is covered in section 7 and the gateway components are addressed in section 8. Finally, section 9 sums up the conclusions that can be drawn from this study and the important issues for further work.

2. Key Features of the Architecture

The main characteristics of the implemented service management architecture are: (i) service modularity, (ii) the concept of service sessions and (iii) service packaging and subscription. Each of these characteristics is described in more detail below:

(i) *Service Modularity*: The platform allows the subscription to, and consumption of value added services according to a session driven model. In order to cope with the heterogeneous base of multimedia and entertainment delivery and processing platforms, the architecture is built around the concept of Pluggable Service Driver Modules (PSDMs). A PSDM is a plug-in component, which controls, manages and interfaces underlying delivery platforms. Examples include a VoD delivery platform consisting of video servers spread over the country, or an e-gaming service delivery platform. Another example is a network of

soft-switches, which can handle the setup and delivery of IP-Telephony calls or Video-Telephony calls. For each of these service technologies and in particular for each vendor specific implementation of a service, a PSDM can be provided which plugs into the service architecture. These PSDMs export well-known control interfaces within the platform and act as drivers to the underlying service delivery platforms.

(ii) *Service Session Concept*: Every kind of service consumption takes place within the context of sessions. When a user accesses the application server implementing the service architecture, he/she starts and enters the context of an access session. This access session establishes a secure context with the service architecture. Within the access session, the end-user is authenticated. From within this access session, which is materialized by a server side access session user agent component, the end-user can start a service session. Within this service session the end-user can execute the logic of the service. The service session is materialized by a PSDM session instance within the application server.

(iii) *Service Packaging and Subscription*: The platform enables the easy packaging of services, applications and features into product packages to which a customer or a subscriber can subscribe online. Even for services for which new appliances are needed, the stakeholder who exploits the platform (e.g. the telecommunication operator or service provider) will deliver and install these appliances as a result of an online subscription. Subscription is at the heart of unlocking revenue streams. Combined with the concept of customer zero-concern with getting the right hardware and software environment at home for consuming certain services, subscription triggers certain company processes in order to install the customer (download software to the home gateway in a secure way or ship certain hardware appliances which the customer connects and which get configured remotely).
When an end-user starts a certain service, the service architecture checks whether the user is authorized to launch the service session according to his actual subscription profile. If this is the case, the service is launched via the appropriate PSDM module. As a result of this service consumption session, a trigger/message can be sent to the billing systems to start billing the service on whatever basis: time interval during which the service is used, type of service, exchanged traffic volume, etc.

The next section provides an overview of the service management platform.

3. Platform Overview

The platform is implemented on a J2EE based application server. The implementation consists of four levels, which are positioned according to the fragments of the Model-View-Control (MVC) design pattern:

(1) A *backend resource layer*: This layer consists of the database(s) and any underlying service delivery platform, which is under control of the platform.

(2) An *EJB backend layer*: This layer consists of a number of deployed Enterprise Java Beans (EJBs) that abstract the resources from the underlying backend resource layer. These EJBs are also referred to as the Pluggable Service Driver Modules (PSDMs). Amongst others, the following EJBs have been deployed:

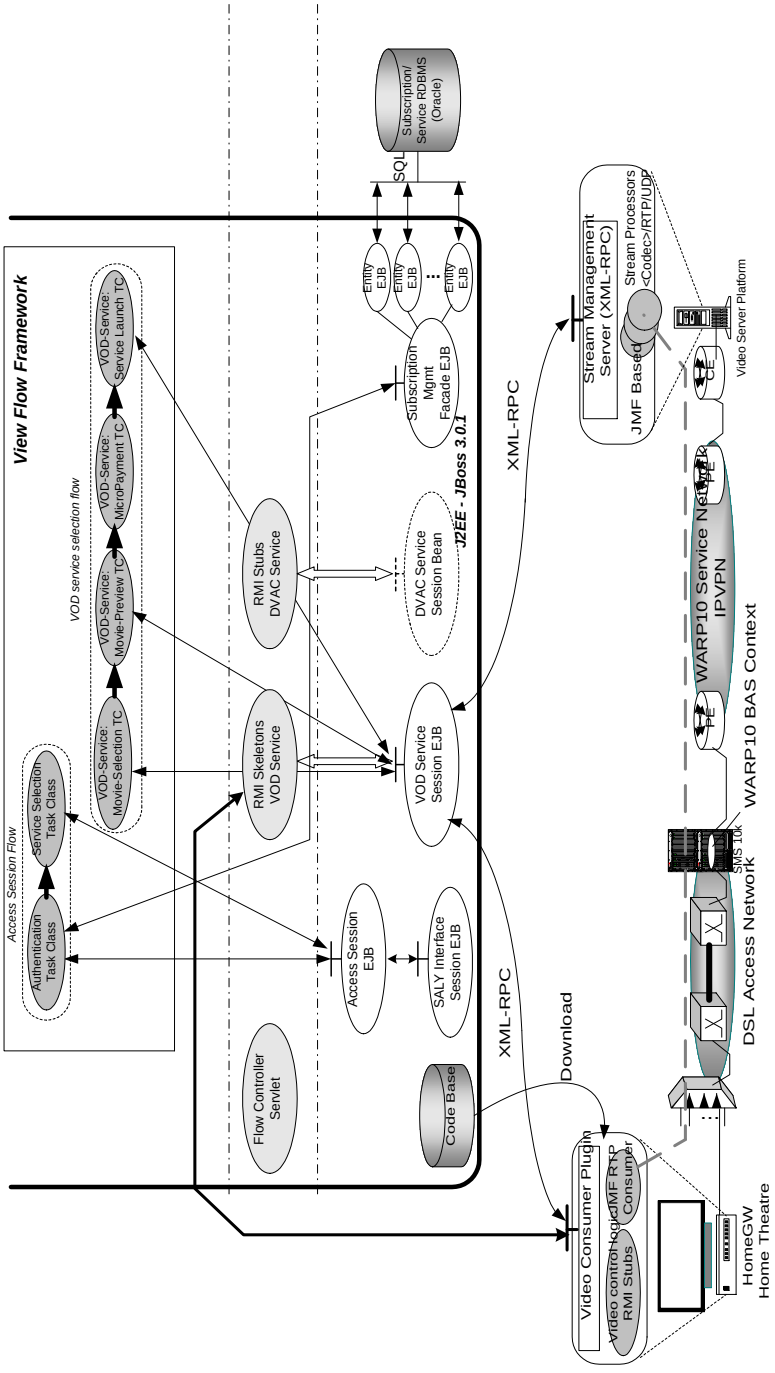

Figure 1. High-level computational decomposition of the implementation: setup of a VoD service session.

(a) An Access Session Bean;

(b) A Subscription Management Facade Session Bean together with a set of subscription level RDMS entity beans;

(c) A Service Control Bean;

(d) A VoD Service Session Bean;

(e) An on-Line Gaming Session Bean;

(f) A Digital Video and Audio Conferencing Bean (DVAC Bean);

Each of these beans will be described in the next section.

(3) The *control layer*: This layer contains a number of servlets which execute service management control by invoking business functions exported by the EJBs of the underlying EJB backend layer.

(4) The *View framework layer*: At this layer, the actual components that materialize the application views to the end-users are located. The view framework is developed around the paradigm of a μ-flow-processing framework. Every logical functionality a user can invoke via the service portal is implemented through and supported by a μ-flow. This μ-flow consists of a number of steps, each implemented in an action task class. An action task class implements an atomic piece of functionality in a generic and reusable way (reusable in other μ-flows). The action class is the place where actual calls are made to backend EJBs. A flow description is read and parsed by a controller servlet which subsequently instantiates the required set of action classes. Each action class is linked to a JSP page which requests the end-user at the browser front-end for the required input, and which returns the appropriate outputs. Every JSP page contains a link to the logically next page, which invokes an operation on the next task class according to the logic of the flow. For every logical action on the service portal front-end, a μ-flow is defined: a flow for setting up an access session, a flow for starting a VoD service session, a flow for taking a new service subscription, etc.

Figure 1 shows the High-level computational decomposition of the platform for the setup of a sample Video on Demand (VoD) service. The authentication and service selection flow is shown, which are implemented as Java Server Pages. These invoke operations on the underlying servlets and Enterprise Java Beans (EJBs). The VoD Service Session EJB is responsible for the configuration of the involved Video Servers and video consumer plugins. For a video session with garantueed bandwidth, an dedicated connection can be realized in the underlying DSL/ATM network (via automatic invocation of the connection management platform).

4. μ-flow Model

The view framework uses the concept of modular configurable μ-flows, which implement the logic of a stepwise execution process with a wizard like presentation style in the browser for the end user. The view framework consists of the following components:

(1) *Description of a μ-flow in XML format*: a μ-flow consists of a number of steps. Within each step, there is indication of (i) the name of a Java task class, which

```
    <state>
     <id>2</id>
      <view>/tasks/ServiceSelect.jsp</view>
      <template>0</template>
       <controllerclass>com.belgacom.warpportal.
               task.ServiceSelect</controllerclass>
       <errorstate>1</errorstate>
      <events>
       <event>
         <id>1</id>
          <actionflow>1</actionflow>
          <actionstate>3</actionstate>
          <actiontype>forwardunload</actiontype>
        </event>
       <event>
          <id>2</id>
          <actionflow>1</actionflow>
          <actionstate>4</actionstate>
          <actiontype>forwardunload</actiontype>
        </event>

                ...
  </flow>
 </flows>
</statemachine>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
 <statemachine>
   <initialize>
     <startflow>1</startflow>
     <startstate>1</startstate>
     <security>1</security>
   </initialize>
   <flows>
    <flow id="1" type="task">
     <states>
      <state>
       <id>1</id>
        <view>/tasks/LoginTask.jsp</view>
        <template>0</template>
        <controllerclass>com.belgacom.warpportal.
                              task.LoginTask</controllerclass>
        <errorstate>1</errorstate>
        <events>
         <event>
           <id>1</id>
            <actionflow>1</actionflow>
            <actionstate>2</actionstate>
              <actiontype>forwardunload</actiontype>
         </event>
        </events>
       </state>
```

Figure 2. Sample XML fragment describing a μ-flow.

executes the logic of the step, (ii) the JSP page to be used for querying the input and presenting the output of the step, and (iii) the next steps/states to be taken in the flow based on possible outcomes of the executed logic in the step. It is possible to refer within a particular flow description to another type of flow. In other words, it is possible to jump from a step in a flow of type X to a step described in a flow of type Y. As such, flows described as a demarcation of a number of steps belonging together in a logical way can be reused within other flows. A sample μ-flow description is shown in the XML fragment in Figure 2.

(2) *A μ-flow controller servlet*: The controller servlet parses on system startup the XML code of the different flows to be supported, and instantiates within a logical scope of each flow a tree of task classes pertaining to each step/state in the flow.

(3) *Task classes*: Each step in a flow is materialized by a task class which implements the TaskHandler Java interface. The most important method of a task class is the execute() method. This method pushes the appropriate JSP/HTML page to the web front-end in order to query for any user input. Next the task performs the required logic by interfacing with the system backend layer, and returns the appropriate output back to the front-end.

(4) *Memory Handler class*: Each flow instance disposes of a memory handler object which maintains the data belonging to and determining intermediate states within a flow. Within the same flow instance all that state data is maintained on the memory handler object which acts as some kind of data clipboard associated to the servlet session. By default, the memory object is cleared when a jump is made from a state in one flow to a state in another flow, however, this default behavior can be changed by configuration in the flow's XML file.

5. Component Description

Access Session Bean. The Access Session Bean is invoked by classes in the access session μ-flow. The first method invoked on the Access Session Bean is `loginOnAccessSession`. Subsequently, the Access Session Bean invokes a method in the Subscription Utility Class (SU) in order to check the subscriber's existence. Directly after authentication, the IP address assigned to the end terminal or home gateway of the end-user is determined. It is initially supposed that the subscriber connects from his home environment via the DSL access network. When the subscriber has been successfully authenticated, the access session flow can take a number of directions depending on the user's preference. The following functionalities can be chosen: (i) service selection and setup, (ii) on-line subscription management, and (iii) identification management. Each of these functionalities will be detailed in section 6.
Figure 3 shows the generic EJB interface operations of the Access Session Bean.

Subscription Management Facade Session Bean. The Subscription Management Facade Session Bean allows the control of the subscription management. A distinction is made between a `customer`, a `subscriber` and a `subscribergroup`. Each `subscribergroup` has an associated Service Access Group (`SAG`), which contains the different services, the `subscribergroup` is subscribed to. The internal database model will be further detailed in section 7.
Figure 4 shows the generic EJB interface of the Subscription Management Control EJB. Only the creation operations are shown. Obviously, the interface also offers deletion and modification operations.

Service Control Bean. This bean allows the generic addition, modification and deletion of services and the associated service parameters. The EJB Service Control interface is shown in Figure 5.

```
interface AccessSession extends EJBObject {

        public String loginOnAccessSession(
                String loginName,
                String loginPinCode,
                String password
        ) throws AccessSessionLoginError,
                          RemoteException;

        public Collection getAllServiceTypes(
        ) throws AccessSessionInfoError,
                   RemoteException;

        public Collection getSubscribedServices(
        ) throws AccessSessionInfoError,
                   RemoteException;

        public EJBObject setupServiceSession(
                String serviceName
                ) throws
                AccessSessionActionError,
                   RemoteException;

        public SubscriptionMgmtControl
                setupSubscriptionMgmtSession(
        ) throws AccessSessionActionError,
                   RemoteException;

        public void changeLoginCredentials(
                String loginName,
                String password
        ) throws AccessSessionInfoError,
                   RemoteException;

        public void changeIdentification(
                SubscriberValObj subscriberInfo
        ) throws AccessSessionInfoError,
                   RemoteException;
}
```

Figure 3. The Access Service Session EJB interface.

```
public interface SubscriptionMgmtControl extends
EJBObject {
        // Customer operations
        public Long createCustomer(
                 CustomerValObj customer)
        throws CustomerAlreadyExists,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        // Subscriber group operations
        public Long createSubscriberGroup(
            String customerRefNr,
             SubscriberGroupValObj subscriberGroup)
        throws CustomerNotInExistance,
               SubscriberGroupAlreadyExists,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        // Subscriber operations
        public Long createSubscriber(
                 String customerRefNr,
                 SubscriberValObj subscriber)
        throws CustomerNotInExistance,
               SubscriberAlreadyExists,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        // Service SAG operations
        public Long createSag(
                 String customerRefNr,
                 String groupName,
                 SagValObj sag)
        throws CustomerNotInExistance,
               SubscriberGroupNotInExistance,
               SagAlreadyAssigned,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        // Service SAG operations
        public Long addServiceToSag(
               String customerRefNr,
               String groupName,
               String sagName,
               SagServiceValObj sagService,
               Collection sagServiceParams)
        throws CustomerNotInExistance,
               SubscriberGroupNotInExistance,
               SagNotInExistance,
               InvalidObjectAttributes,
               ServiceAlreadyInSag,
               InvalidObjectAttributes,
               ManagementError,
               RemoteException;

}
```

Figure 4. The Subscription Management Control EJB interface.

Service Session Bean. Different specific Service Session Beans have been developed: (i) a VoD service session bean, (ii) an on-line gaming session bean and (iii) a digital video and audioconferencing bean. Other types of beans can easily be integrated with the described platform.

6. Access Session Scenario

When an end-user logs in to the service portal, the access session μ-flow's first task class instantiates an Access Session Bean, of which the reference is put on the HTTP session context object of the user's HTTP session (created and managed by the view framework controller servlet). After login, the IP address of the end terminal or home gateway of the end-user is determined and the following choices are presented to the customer:

- **Choice 1: Service selection and setup** The user is guided via a number of subsequent task classes through the selection and setup of a service.

- **Choice 2: On-Line subscription management** The user enters subscription management mode and can take a new subscription to a set of services and service packages, or can change and adapt the detailed characteristics of an existing subscription. Note that in this case the subscriber should have the authorisation to adapt subscription assignment groups and services within subscription assignment groups.

```
interface ServiceControl extends EJBObject {

        public Long addService(
                 ServiceValObj service,
        Collection serviceParameters)
        throws ServiceAlreadyExists,
      InvalidObjectAttributes,
               ManagementError,
               RemoteException;

        public void modifyService(
                 String currentServiceName,
                 ServiceValObj service)
        throws ServiceAlreadyExists,
                 ServiceNotInExistence,
                 InvalidObjectAttributes,
                 ManagementError,
                 RemoteException;

        public void deleteService(
                 String serviceName)
        throws ServiceNotInExistence,
               LinkedObjectsError,
               ManagementError,
               RemoteException;
```

```
        public Collection addParametersToService(
                 String serviceName,
                 Collection serviceParameters)
        throws ServiceNotInExistence,
               ParameterInExistenceError,
               ManagementError,
               RemoteException;

        public void modifyServiceParameters(
                 String serviceName,
                 Collection serviceParameters)
        throws ServiceNotInExistence,
               ParameterInExistenceError,
               ManagementError,
               RemoteException;

        public void deleteServiceParameter(
                 String serviceName,
                 String parameterName)
        throws ServiceNotInExistence,
               ParameterInExistenceError,
               LinkedObjectsError,
               ManagementError,
               RemoteException;

}
```

Figure 5. The Service Control EJB interface.

- **Choice 3: Identification management** The end-user enters a mode where personal identification parameters can be altered, e.g. : login names, passwords, address information, phone numbers, etc.

Each of these choices is detailed in the subsequent paragraphs.

Choice 1: Service Selection and Setup.

(1) In case of service selection, the flow passes via a service selection task class, which calls the `getSubscribedServices` method of the Access Session Bean. This method returns all the service types and corresponding service parameter ranges for which the subscriber has the right to launch service session instances.

(2) Subsequently, the subscriber selects a service and indicates he/she wants to start a service session instance. For this purpose, the access session flow will invoke for this purpose the `setupServiceSession` operation and passes the name of the service type. As a result, the access session bean will start up a service session bean of the right type and places the obtained reference on the HTTP Context Session of the subscriber's HTTP/servlet session. The access session flow stops here, and hands over the control to a service specific flow, which then handles the steps specific to the setup of the selected service.

Choice 2: On-Line Subscription Management.

(1) In case of on-line subscription, the access session flow delegates further control to the subscription management flow, which starts the subscription management feature selection class. This class starts a subscription management session through the operation `setupSubscriptionMgmtSession`. This operation

instantiates a subscription management session bean and returns its reference. This reference is put on the HTTP/Servlet Session Context object.

(2) In a subsequent flow task class, the subscriber can select one of the basic features of subscription management which are:

   (a) Addition/Deletion of service subscriptions to the portfolio of one of the customer's subscription assignment groups;

   (b) Modification of a service subscription within a subscriber group's SAG; Amongst others:

      - Change of subscription periods and intervals;
      - Change of service quality degrees and profiles;
      - Adaptation of basic service parameters;

   (c) Addition of new subscribers to one or more subscriber groups;

   (d) Modification of properties pertaining to the customer identity or pertaining to the subscriber group(s) created within the customer's scope;

   (e) Granting on-line subscription management authorisation at different levels to subscribers belonging to the customer's scope.

(3) Once the desired subscription management feature has been selected, the subscription management flow passes via the actual feature execution task, which invokes the subscription management bean created in the previous step. Its logic is executed through aid of the subscription management utility class.

Choice 3: Identification Management.

(1) In case of on-line identification management, the access session flow passes the action task class allowing identification change management feature selection. Two basic features are exported:

   - Modification of login credentials: login name and password. The login pincode has been assigned to the involved subscriber at the time of initial off-line subscription of the initial set of subscribers, or at on-line subscription when an authorized subscriber created the account for the involved subscriber. This pincode cannot be changed.
   - Modification of detailed affiliation data pertaining to the subscriber: address info, phone numbers, aliases, etc.

(2) After selection of the desired feature, the flow passes the feature execution task class, which on its turn invokes either `changeLoginCredentials` or `changeIdentification` dependently.

7. Database Model

The database's internal structure is detailed in Figure 6, which shows the scheme of the involved records. Every user of the system is described as a `subscriber`. If a user does not have a subscriber object configured in the system for which he knows the login name, password and pincode, that user cannot be authenticated to the service platform. There are two types of subscribers: `residential subscribers` and

`corporate subscribers`. A subscriber object can only be of one type. A residential subscriber is acting independently and has full control over the services with the associated parameters he/she subscribes to. A corporate subscriber is always linked to a larger affiliation, which we represent in the inventory as a `customer`. Under the authority of a customer, a number of `subscriber groups` can be created. These subscriber groups contain one or more corporate subscribers. A subscriber group represents a logical group of users who have the same subscription contract, i.e. who have the same set of services (and service customizations) to which they are subscribed. A corporate subscriber is made member of a subscriber group through a `subscriber membership` object. As such a corporate subscriber can be part of multiple subscriber groups. A subscription to one or more services is always represented by a `subscription assignment group` or a `SAG`. A SAG contains one or multiple `sagservice` objects. A `sagservice` is a customization object of a corresponding singleton service object. This service object has a number of associated service parameter objects. These service parameters represent the customizable properties of a service together with maximum/minimum values or a set of allowed values or strings. For example, a video conferencing service has as property the number of parties that can be connected together in a session. The minimum value will be 0 and the maximum value could be 10. A sagservice object is a customization of a service object. A SAG is connected to either a subscriber group or directly to a residential subscriber. In case it is connected to a subscriber group, all corporate subscribers belonging to the group have the subscription as described by the SAG. In case it is connected to a residential subscriber, the SAG describes the subscription contract of only that residential subscriber. Finally, there are a number of auxiliary tables which are service specific. In the figure, the `VODSERVER` and `MOVIE` tables are specific for the VoD service implementation.

8. Home Gateway

The terminal architecture, also referred to as the home gateway, from which end-users consume multimedia and domestic services, is designed on a Linux based platform. The home gateway is designed according to emerging OSGi (Open Service Gateway initiative: [13]) concepts. A JVM based OSGi control kernel and corresponding bootstrapping software will be automatically downloaded from the platform on initial powering and sign-on of the gateway. The service management platform takes all responsibilities to download, remotely configure, and manage the Java based service consumption bundles on top of the JVM based OSGi execution kernel. Remote version management of these software bundles is under control of the platform.

9. Conclusion and Further Work

A generic architecture has been designed for the integration of stand alone service components, be it off-the shelf or developed in-house, through Pluggable Service Driver Modules (PSDMs). PSDMs act as service abstracting computational components that wrap and abstract the technology and vendor specific service logic within the generic service delivery architecture. The architecture has been implemented by making use of J2EE technology.
Examples of service technologies integrated in the platform are video conferencing control systems (e.g.soft-switches enabling SIP based video telephony, VoD platforms,

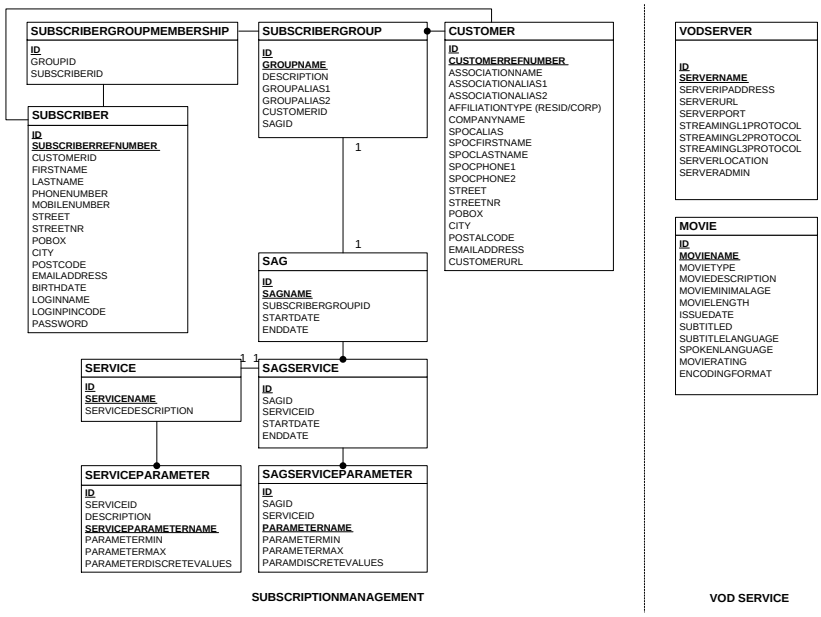

Figure 6. The internal database structure.

e-gaming control platforms, domotics systems at home, etc.) By means of an extensible internal RDBMS architecture, the platform will allow extensible on-line subscription management of customers, customer subscriber groups, and subscribers. The whole process of setting up an access session from the home with the platform, selecting a next generation service, consuming it and billing it within the controlled and managed scope of a single/multi-party, multimedia session, is handled according to a flow-driven model in a transactional execution context.

Important issues for further work include: (i) a thorough performance evaluation of the platform, (ii) application of load balancing techniques for the distribution of the platform load and (iii) development and thorough testing of several other types of service control beans, especially for the control of services delivered to wireless devices and (iv) designing Terminal QoS matching algorithms for deciding the best terminal parameters for conversational services between users with heterogeneous terminal equipment. The design of these QoS matching algorithms will be based on the work, presented in [14].

Acknowledgments

The first author wishes to thank his colleagues at the IBCN (INTEC Broadband Communication Networks) research group, Ghent University for the stimulating discussions.

References

[1] F. De Turck, F. Vandermeulen, P. Demeester, *On the design and implementation of a hierarchical, generic, scalable open architecture for the network management layer*, IEEE/IFIP Network Operations and Management Symposium, pp.745-758, Honolulu, April 2000.

[2] F. De Turck, S. Vanhastel, F. Vandermeulen, P. Demeester, *Design and implementation of a Generic Connection Management and Service Level Agreement Monitoring Platform Supporting the Virtual Private Network Service*, IFIP/IEEE IM 2001 conference, Seattle, May 2001, pp. 153-166.

[3] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, *SIP: Session Initiation Protocol*, IETF RFC 2543, March 1999.

[4] ITU H.323 Standard: *Packet-based multimedia communications systems*

[5] M. Arango, A. Dugan, I. Elliott, C. Huitema, S. Pickett, *Media gateway control protocol (MGCP)*, IETF RFC 2705, October 1999.

[6] Media Gateway Control (megaco) Charter, *www.ietf.org/html.charters/megaco-charter.html*

[7] Java API for Intelligent Networks (JAIN), *http://java.sun.com/products/jain/api_specs.html.*

[8] Java(TM) 2 Platform, Enterprise Edition (J2EE), *java.sun.com/j2ee/*

[9] XML-RPC Specification, *http://www.xmlrpc.com/spec*

[10] SUN Microsystems, *Java Remote Method Invocation Specification*, 1998, *http://java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmiTOC.doc.html*

[11] S. Sengal, J.W. Gish, J.F. Tremlett, *Building A Service Provisioning System using the Entreprise Java Bean Framework*, Network Operations and Management Symposium, 2000, pp. 367-380.

[12] IETF Open Pluggable Edge Services (opes) Working Group, *http://www.ietf.org/html.charters/opes-charter.html*

[13] Open Service Gateway initiative (OSGi) - Specification Overview, *http://www.osgi.org/resources/spec_overview.asp*

[14] F. Vandermeulen *et al*, "*A Multimedia Terminal Architecture for Dynamically Configurable Protocol Stacks*", ICME 2000 conference, New York, August 2000.