Platform for Control and Delivery of services in Next Generation Networks

DELIVERABLE 3.1

Design of the prototype

Questions about this document or the features it describes should be directed to:

CEFRIEL

RECORD OF CHANGES

Version	Date	Description
1.0	31/10/2008	First version
2.0	30/10/2009	Document Revision

Index

1 Introduction

1.1 Rich communication

The main purpose of mobile communications over the past years has been to bring text, audio and video on mobile devices.

Recently, several initiatives have been made to enrich the experience of communication between devices. One of these is the RCS (Rich Communication Suite) moved to a GSMA (GSM Association) work program which will add some additional features to the traditional communication. The aim can be summarized in three goals:

- Enhanced Phonebook
- Enhanced Messaging
- Enriched Call

The achievement of these goals introduces the concept of presence information, image and video sharing, instant messaging, and other services.

To bring these additional services, RCS uses the IMS (IP Multimedia Subsystem) framework.

1.2 IP Multimedia Subsystem

The IP Multimedia Subsystem (IMS) is a framework for delivering IP multimedia services over CS (Circuit Switch) networks (e.g. GSM, LTE…). The aim is to bring Internet services in mobile networks making interoperable and transparent the circuit switch and packet network.

Mainly to satisfy some of these needs, IMS uses SIP and other protocols such as (SIMPLE, XCAP, MSRP, etc.).

1.3 PICO Goals

The project PICO aims to study innovative telecom services for service delivery on large bandwidth networks, focusing on convergence between fixed-mobile phone networks and on technologies for service creation, provisioning, and management.

The main project goal is the study and experimentation of innovative service delivery for fixed-mobile convergence based on IMS platform; the complex IMS structure must be adapted and extended to provide access to fix and mobile telecom operators.

The specific goals consist in the implementation of a prototype and analysis of the features offered by IMS system for "context-aware" application streaming, and, in general, in remote applications usage by means of a virtual access to different protocols compliant with IMS system.

This deliverable contains a description of the prototype, an analysis of all components that form a complete device, both from a logical view and a physical view. In this deliverable we also introduce a use case notation that better explains the interaction between the different actors of the prototype (user device, IMS subsystem, application server, etc.).

The last paragraph is dedicated to the technical specifications of how the prototype can be developed and technical suggestions of new technologies that help the development of our prototype.

2 Prototype Description

In the scenarios described in the deliverable [1], our prototype can be viewed as set up by two different elements, one with the role of the client and one with the role of the server. Our attention is focused on the client device, which is used by a human user: an Emergency Medical Services User (EMS), a Fire Fighters (FF) or a Law Enforcement Agent (LE). This device transmits with an application server belonging to IP Multimedia Subsystem; the server manages all the information regarding the different users of the system, profiles, resource access grants, and allows the authenticated user for download of an application among the downloadable set.

The following figure (Figure 1) describes the human actors working on this scenario.

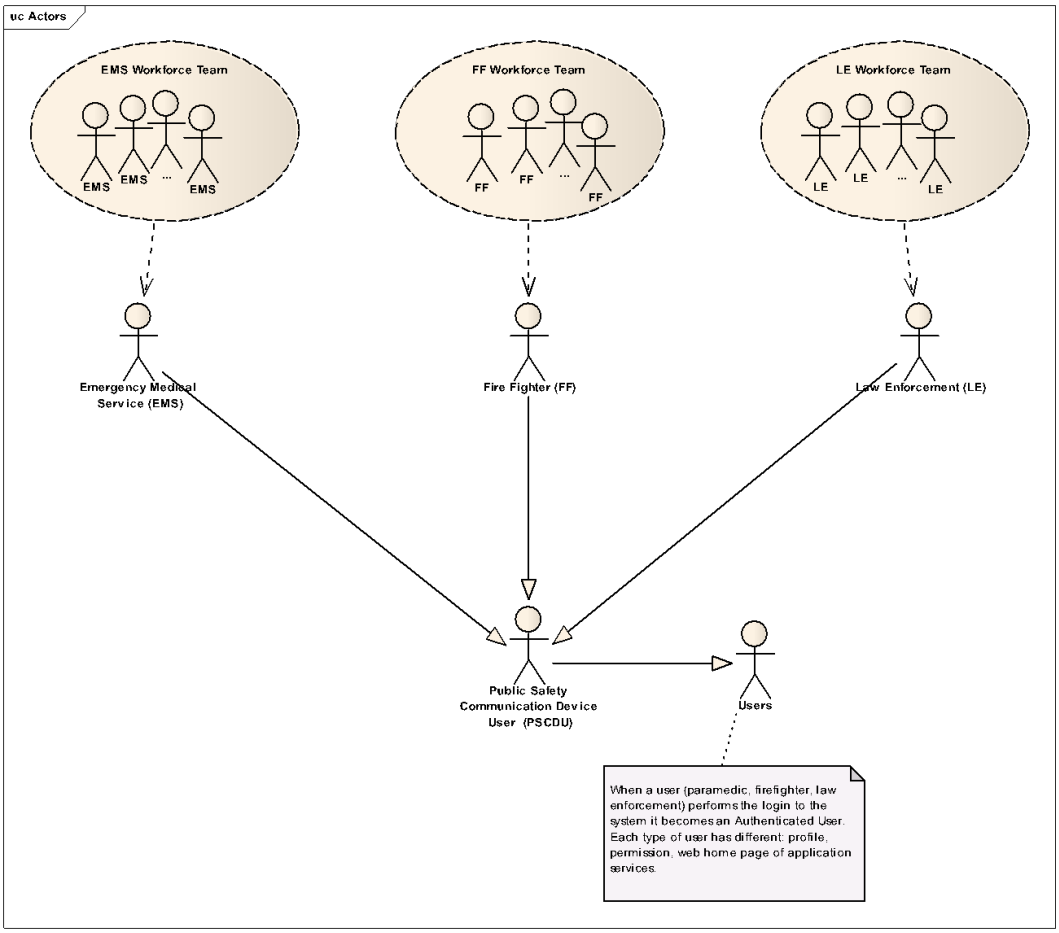

Figure 1: Actors

All the users have a different profile based on their user type, for instance a normal paramedic will be granted a information access restricted profile; he would be allowed to download a base application (an application that allows a video-conference with paramedics, cardiologist and poison team), or an application that allows the access to the database of the nearby hospitals (read access only), or would be allowed to call for help the Emergency Operations Center (EOC).

A manager of the Incident Command Team instead will be granted a broader set of application downloadable applications and information, such as an application that allows enabling or disabling the traffic lights, through he can view the different vision of all the team's camera.

In the deliverable [1] we analyzed two types of scenario, one with specific context where all the people that work in this context belong at the same group (Emergency Medical Services (EMS), Fire Fighters (FF) and Law Enforcement (LE)) and the other one where users belong to more different groups, supposed on cooperation.

In detail:

- An homogenous group: EMS Workforce Team or FF Workforce Team or LE Workforce Team
- A heterogeneous group: all type of Public Safety Communication Device Users (EMS, FF and LE) can work together; this group is called Incident Command Team. This is summarized in Figure 2.

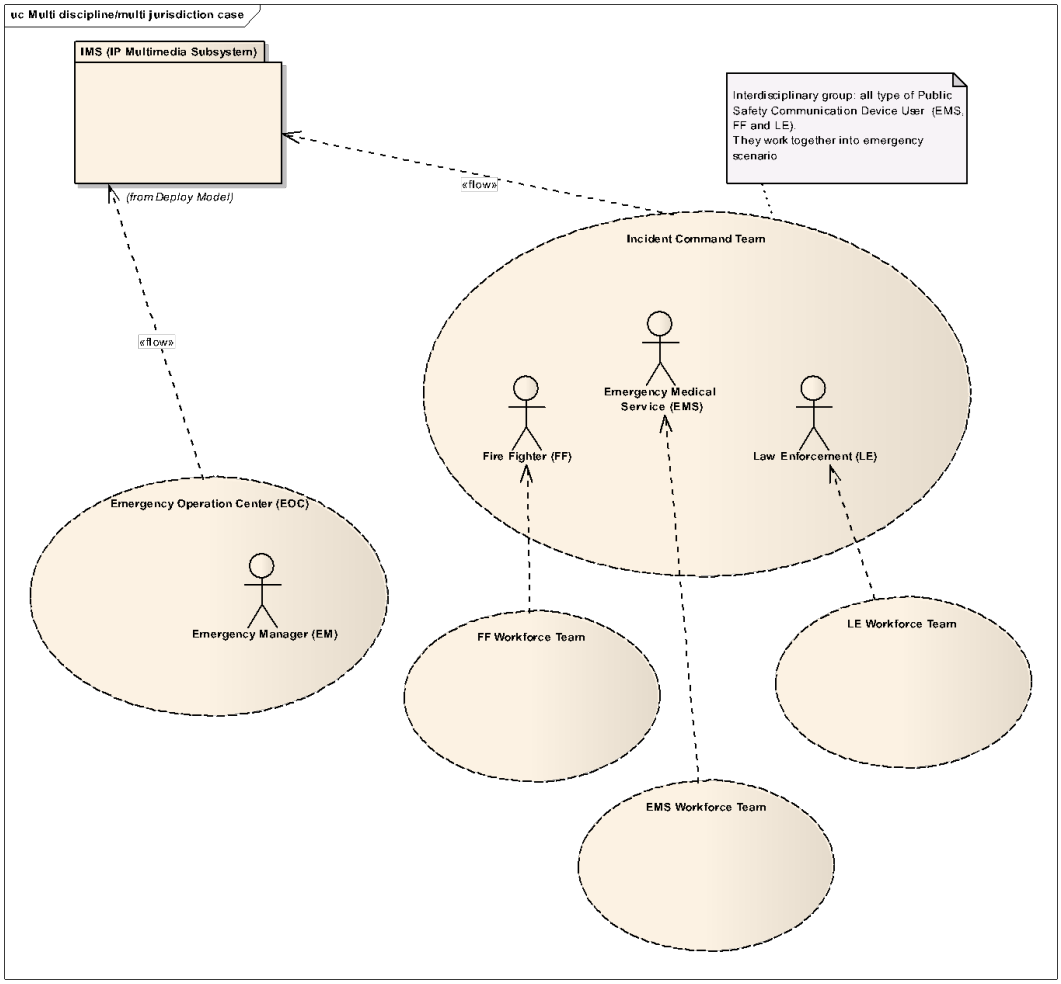

Figure 2: Interdisciplinary group

The following paragraphs describes: the IMS architecture, the basic PICO demonstrator, the main downloadable applications; we point out the interaction of different IMS architectures and the mobility case (home network and visited network).

2.1 IMS Architecture

This paragraph describes IMS architecture from the point of view of our prototype; then we point out the main elements that set up the core architecture.

The IMS (IP Multimedia Subsystem) technology is the key element in the 3G and NGN (Next Generation Networks) architectures that will merge the Internet with the cellular world. It will make possible to provide ubiquitous cellular access to all the services that the Internet provides, so that internet technologies, such as the web, email, instant messaging, presence, and videoconferencing will be available nearly everywhere. IMS fills the gap between the two most successful communication paradigms, cellular and Internet technology.

Before starting to explain IMS architecture in detail, we're going to have a view of the IMS basic principles that are interesting for our structure.

- IMS enables *access independence*. This means that all existing networks could work with IMS, through appropriate gateways and interfaces, in order of the layer considered.
- IMS works with *terminal and user mobility*.
- IMS allows operators and service providers to use *different underlying network architecture*.
- IMS offers extensive IP-based services, such as VOIP (Voice over IP), POC (Push to talk Over Cellular), multiparty gaming, videoconferencing, presence information, instant messaging, content sharing, and so on.

To satisfy the requirement imposed from IMS features, a layered architecture was selected for this architectural framework. From bottom to up, in IMS layered architecture we have: Access plane, Control plane and Applications plane.

Access plane achieves the connection of all users to IMS core network. This is made directly if the user employs an IMS terminal, and through gateways if the device is not IMS. Gateways employ standard interfaces that make it possible communicating with all existing entities. This layer is hence directly responsible for carrying the traffic between endpoints.

Control plane has as core occupation the Call Session Control Function (CSCF). This function is reached by sharing the control between three different entities: Proxy, Serving and Interrogating. We're going to analyze them in detail further on.

Applications plane hosts and executes services (IP applications), and delivers them using SIP to interface with the Control layer.

Now we can go beyond this overview analyzing in detail parts that are interesting in our structure based on IMS architecture. We begin by explaining all the elements that appear in the following figure (see Figure 3).

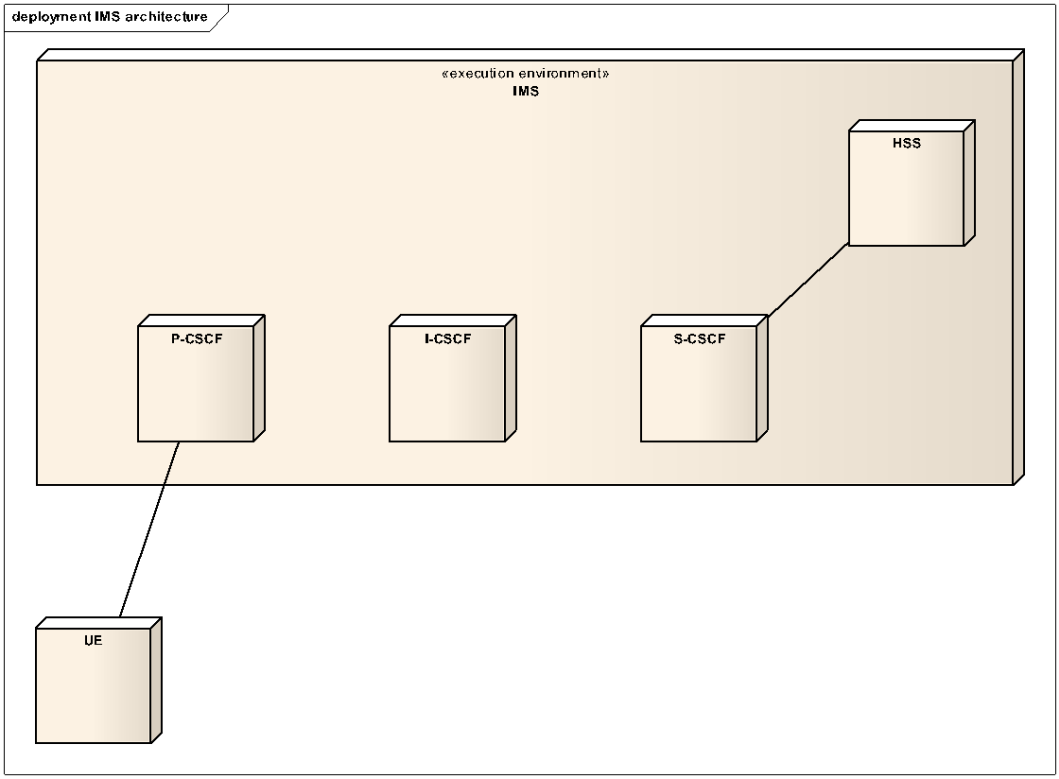

Figure 3: IMS architecture

2.1.1 Home Subscriber Server (HSS)

HSS is the main database supporting IMS architecture entities responsible for call or session management. HSS contains all the information about user profile and manages authentication and authorization at IMS level and can provide information about physical position of the user. Information about user profile includes user identity, allocated S-CSCF name (see the S-CSCF paragraph), roaming profile, authentication parameters and service information. User identities can be private or public. The private user identity is assigned by the home network operator and is used for such purpose as registration and authorization, while the public user can be used from other user for requesting communication with the end user. HSS also provides the traditional Home Location Register (HLR) and Authentication Centre (AUC) functions, required by the PS domain and the CS domain. Depending on several factors, like number of mobile subscribers, capacity of the equipment and network organization, there may be more than one HSS per home network. In case of multiple HSS, a Subscriber Location Function (SLF) is needed to map user addresses to enable I-CSCF, P-CSCF and AS to find the address of the HSS holding the required user-specific data. Both HSS and SLF communicate through Diameter protocol.

2.1.2 Call Session Control Function (CSCF)

Processing SIP signaling packets in the IMS is achieved by using a group of entities that we could call “the session management and routing family”. This family consists of three entities: Proxy CSCF (P-CSCF), Interrogating CSCF (I-CSCF), and Serving CSCF (S-CSCF).

2.1.2.1 P-CSCF

The first point of contact for users within IMS is P-CSCF. It performs a stateful SIP proxy, all the traffic from/to end users passes through this entity. The P-CSCF validates the request, forwards it to selected destinations and processes and forwards the response. It performs user authentication, can establish a secure session IPsec with IMS terminals and supports Resource Admission Control functionalities. In addition, it may behave as a User Agent, which role is needed for releasing session in abnormal conditions and for generating independent SIP transactions.

In one operator network there can be one or many P-CSCF. The terminal discover its P-CSCF using DHCP or, in GPRS, PDP Context. When an IMS terminal is assigned to a P-CSCF this association does not change during the validity period of the registration. We're going to see each function in detail:

- Forwards SIP register requests to I-CSCF based on a home domain name provided by the user in the request, and other requests and responses received by the user to S-CSCF.
- Detects emergency session establishment requests.
- Sends accounting related information to the Charging Collection Function (CCF).
- Provides integrity protection of SIP signaling and maintain an IPsec security association with user.
- Can make compression of SIP messages to reduce the round trip time over critical links.
- Executes media policing, checking the content of the SDP payload (SDP: Session Description Protocol) to ensure the media is allowed for the user.
- Maintains session timers. It can detect a free resources used up by hanging sessions.
- Interacts with Policy Decision Function (PDF), which can also be included in P-CSCF. PDF is responsible of authorizations policy on media plane information obtained from P-CSCF. This function takes a service level policy request from the application layer and translates it into IP QoS parameters.
- Can provide defenses against SIP signaling attacks.

2.1.2.2 I-CSCF

It's a stateless SIP proxy placed on the edges of an administrative domain and it's a contact point for all connections destined to a user actually roaming in a different operator network. It's the entity that is able to determine the S-CSCF with which a user should register. This is achieved by querying the HSS. The I-CSCF can be removed from the signaling path once it has been used to establish which S-CSCF is in use. The I-CSCF IP address is published in the Domain Name System (DNS) of the domain, so that remote servers can find it and use it as a forwarding point. Up to release 6, I-CSCF can also be used to hide topology, capacity and configuration of the internal network from the outside world (function THIG), but from release 7 this function is part of the Interconnection Border Control Function (IBCF). IBCF roles include the provision of NAT and Firewall functions for signaling, policing of signaling, topology hiding and conversion between IPv4 and IPv6. It also controls the media exchanged across the operator boundary.

2.1.2.3 S-CSCF

It's the brain of the IMS. This entity registers the users and provides services to them. It performs multimedia session setup, changing and release; routing, translation, provides billing information to

media systems, interrogates HSS to retrieve authorization, service triggering information and user profile, using Diameter protocol. It has no local storage of the user. It's a SIP server, always located in the home network. During the session setup it can monitor SDP to ensure the session respects user profile edges. In detail, the functionalities performed by the S-CSCF are:

- Handling registration request; it knows the user IP address and which P-CSCF is using as IMS entry point.
- User registration and de-registration (with registration timer supervision) and authentication by means of the IMS Authentication and Key Agreement schema.
- Download of user information and service related data from HSS when necessary.
- Routing of mobile-terminating traffic to P-CSCF and of mobile-originated traffic to I-CSCF, BGCF or AS.
- Session control with capability to decide when a request must be further processed by routing to an AS, that is interaction with service platforms.
- Translation of E.164 numbers to SIP URI needed from SIP signaling routing.
- Media policing checking SDP payload.
- Supporting of emergency sessions.
- Sending of accounting-related information to the Charging Collection Function (CCF).

2.2 IMS complementary protocols

Lately we are seeing an exponential growth of mobile applications and that's transforming the mobile market more and more application centric.

The enormous growth we are seeing of mobile application stores (e.g. Apple Store, Android Market, etc.) is an example that the communication is not limited just to the text, audio or video but a communication between applications.

IMS implements the MSRP[1] protocol for the exchange of files. It is based on SIP and can exist only if a SIP Session has been established. It uses SDP protocol to describe all the exchange parameters between two endpoints. An extract of an SDP Message description is shown below:

```
v=0
o=pico001 2890844526 2890844527 IN IP4
picoserver.cefriel.it
s= -
c=IN IP4 picoserver.cefriel.it
t=0 0
m=message 7394 TCP/MSRP *
a=accept-types:text/plain
a=path:msrp://picoserver.cefriel.it:7394/3f67i7ea2b;tcp
```

[1] **MSRP:** (Message Session Relay Protocol - RFC 4975 -) : is a protocol for transmitting files over SIP

SDP protocol contains all the details to route communication over TCP.

MSRP Protocol is not yet widely used and although it allows transferring files, a lack of it so far is the full support of the application sharing, managing and possibly, communication between them.

2.3 PICO Demonstrator

In the context of IMS Architecture described in the paragraph 2.1, our prototype is composed by two different parts (see Figure 4):

- User device named **Public Safety Communications Device** (PSCD): a next generation device, it is the operators' equipment that allows user to use the IMS subsystem and your services;
- Application Server named **Public Safety Communication Server** (PSCS): an application server that belongs to the IP Multimedia Subsystem, and performs the functionalities of: authentication and authorization of users based on their profiles, mobility handling, application on demand handling.

Both PSCD and PSCS must be authenticated by IP Multimedia Subsystem, and communication is based on:

- SIP protocol
- MSRP protocol
- ConteXML
- Other protocol to be defined

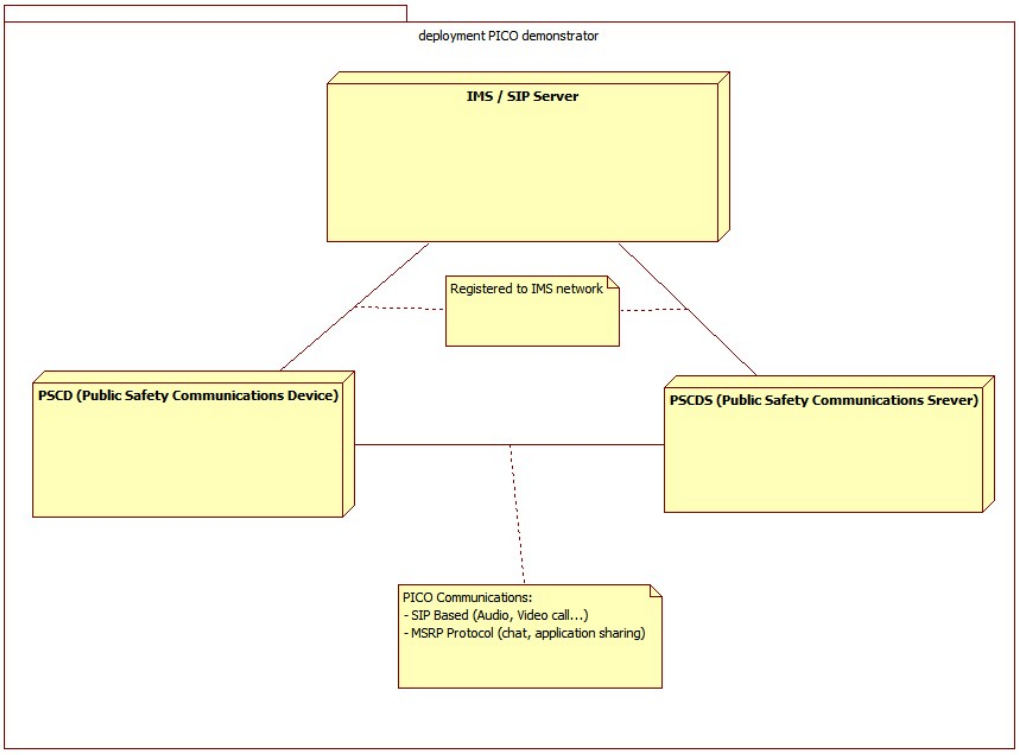

Figure 4: PICO demonstrator

The following paragraphs describe the PSCD and PSCS on the point of view of logical view and physical view.

2.3.1 Architecture description

As described above, PICO architecture is based on IMS Framework. The PCSDS is an application which performs an IMS registration and is considered in the system as a Robot. All PSCD users using a PICO client (extended IMS client) are registered to the IMS as well for the geographical area pertaining to the PICO server. Each user has PICO robot (PSCDS) as buddy[2] and periodically sends[3] its context using MSRP protocol and the IMS-SIP session.

The context is a XML File (ConteXML[4]) which contains several contexts information such as user location, battery level, disk usage, type of user and so on. PICO Server processes the context of all users in a specific area and also analyzing the context of emergencies, it performs reasoning to offer or send relevant context application.

[2] **Buddy:** A friend or contact added to the contact list/phone book
[3] The frequency of delivery depends on PCDS location and from emergencies in a specific area
[4] **ConteXML**: Extended XML file with contextual parameters

PICO server (PSCDS) uses a rule engine (reasoner) and some preset rules to perform his actions. The application are exchanged, organized and installed using the MSRV protocol and this extends the IMS platform with full application support.

2.3.2 Public Safety Communications Device (PSCD)

A Public Safety Communications Device is a generalization of a particular device of each type of user: a Law Enforcement Public Safety Communications Device, Fire Fighters Public Safety Communications Device and Emergency Medical Services Public Safety Communications Device. This is illustrated in Figure 5.

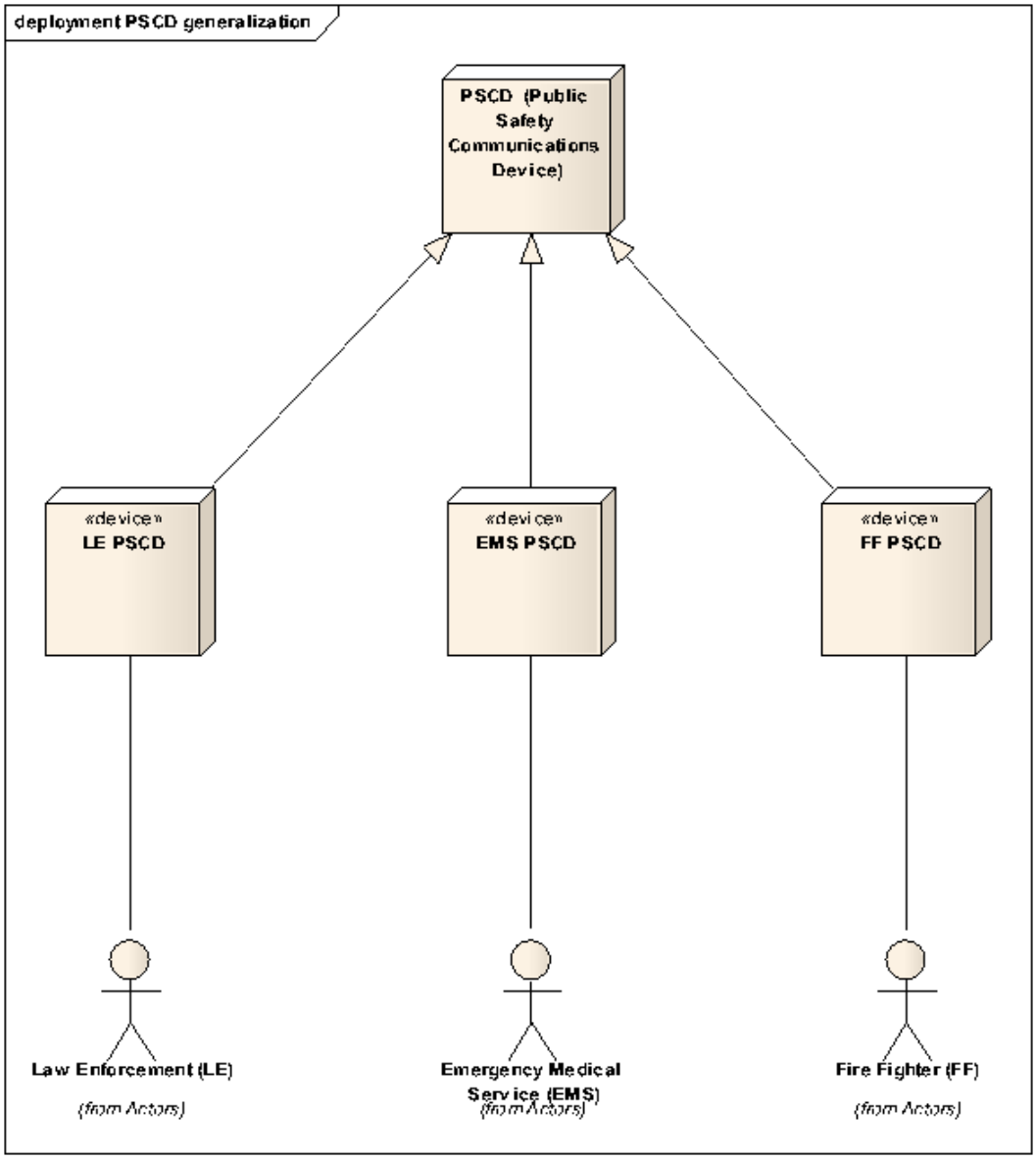

Figure 5: PSCD generalization

In the deliverable [1] we analyzed emergency scenarios and we saw the importance of application delivery and the operators' equipment during such a situation. The application and services deployment needs to be as fast as possible and the operators' devices need to be small and portable.

In the deliverable [2] we presented the available technologies regarding the application on demand service; on the other hand, the aim of this chapter is to provide a full overview of the available mobile devices including operating systems. We present indeed the most important environment regarding the development of software for small devices, we will present the operating systems and we will give a brief overview concerning the most important current devices.

Each Public Safety Communications Device is a device of next generation and it must have more additional equipment (see Figure 6) to perform all the services required.

The device must have, at least:

- a video camera to perform videoconferencing and allowing recording user sights occurred during his presence;
- a microphone to perform audio calls;
- a GPS device to perform the geo-location.

Moreover, the device could have an additional subset of sensors in order to perform registration of vital parameters, for example the EKG unit, a respirator monitor, and a blood pressure monitor.

The portable device of every paramedic could monitor vital signals but also analyze blood and chemical air composition, and also the substances that intoxicated the patient. Streaming is helpful because, in case of need, the paramedics could enable the plug in for streaming, for analysis purposes.

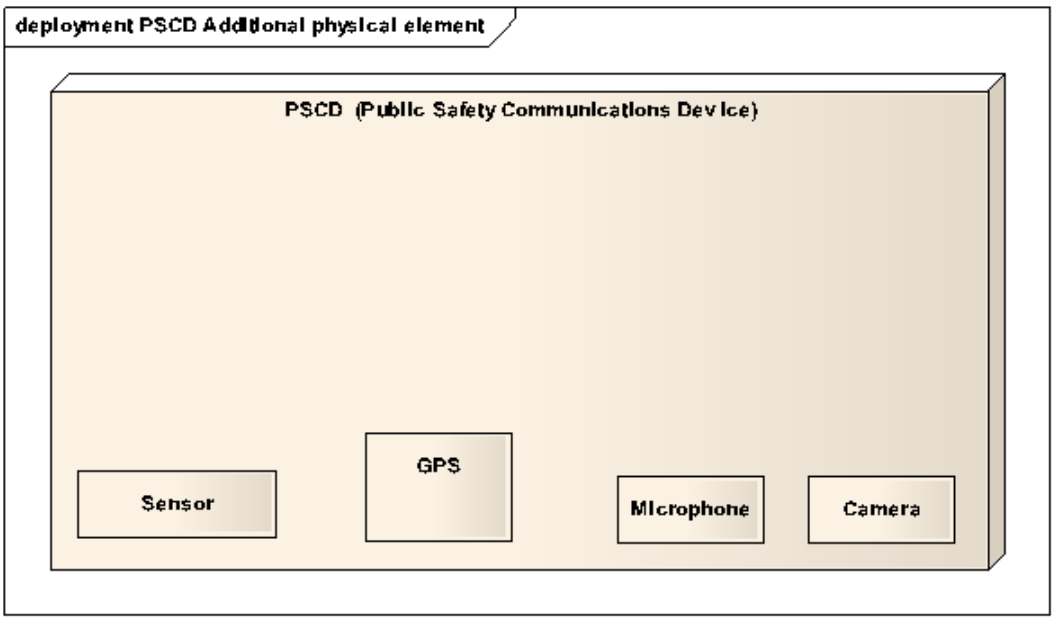

Figure 6: PSCD required and optional equipment

A logical view of the PSCD is described in Figure 7, which identifies five main blocks (or subsystems):

- **IMS client**: This subsystem is responsible for implementing all the IMS client features required by the PSCD. Most important IMS features here supported are user authentication to the IMS, audio/video call setup, presence service and IM. More generally, the IMS client manages all the multimedia sessions required by the PSCD. IMS client exports a proper set of APIs to the other PSCD subsystems. Moreover IMS client implements the MSRP protocol to allow file transfer, and more precisely it provides the application transfer from PSCD Server.
- **Base Features**: This subsystem is the core of the PSCD
- **User Driven Applications**: This subsystem collects all the applications required by the PSCD during operations and that are specified to the user profiles and needs
- **Device Driven applications**: This subsystem collects all the applications required by the PSCD during operations and that are specified to the device type and the equipment installed on it.

Each User after IMS authentication and Public Safety Communication Server authorization reaches his own home page where he can choose which Application On demand he could download.

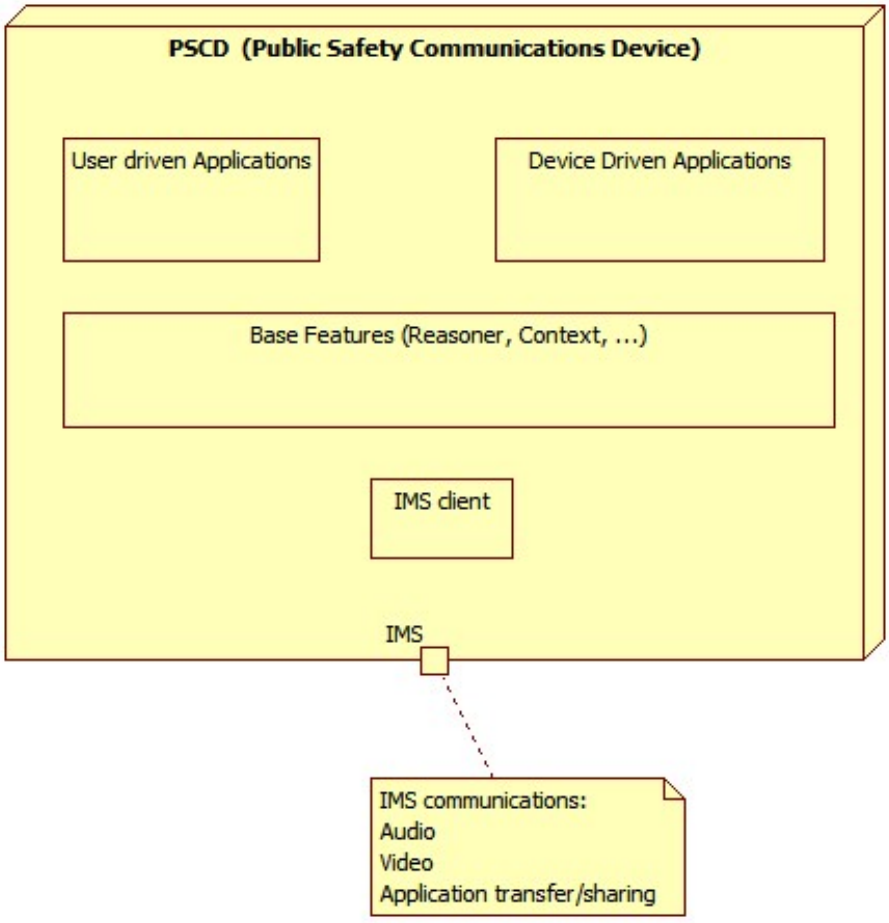

Figure 7: PSCD internal view

As described in chapter 2.3.1, PICO Client or PSCD is an extended IMS Client. It provides all SIP functionalities such as:

- Audio call
- Video call
- Chat
- Application transfer (file transfer)

This type of communications can be established among users or between Pico client and Pico server. In fact, it periodically sends it context (based on location, battery life..) using a ConteXML file to the Pico Server (PSCDS) to receive the updates (notifications or applications)

2.3.3 Public Safety Communication Server (PSCS)

The Public Safety Communication Server (PSCS) is the server of the PICO demonstrator and it enables to the PSCD the access to the PICO services and applications. We can see the PSCS as an application server of the IMS but we can also identify it as a client of the IMS with special functions.

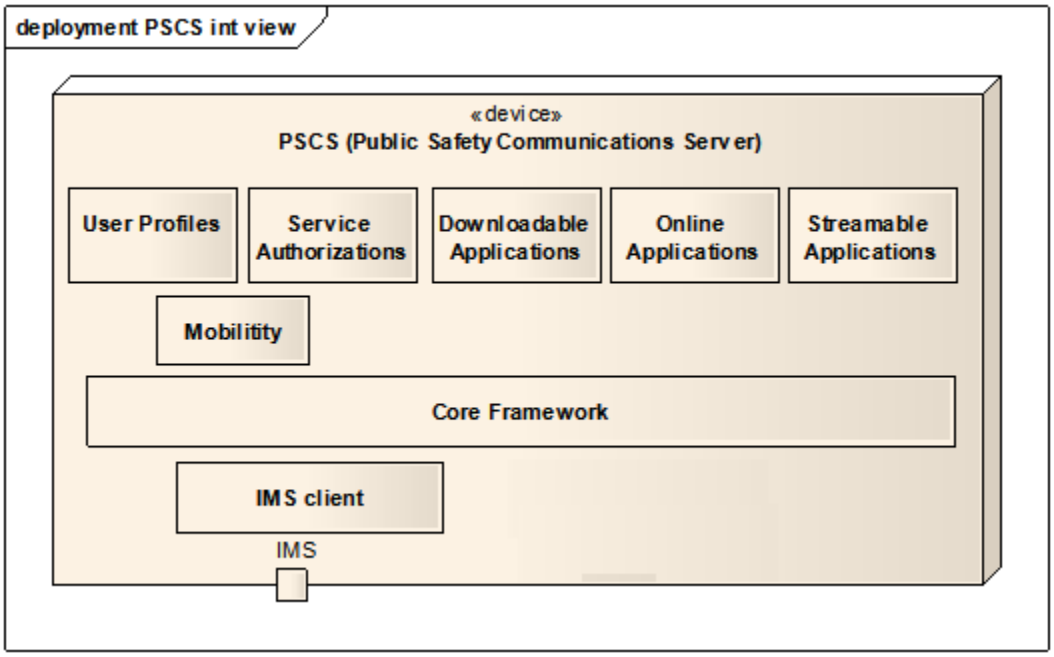

Figure 8: Public Safety Communication Server

The Figure 8 shows the internal view of Public Safety Communication Server.

It is made up by different elements:

- **User Profile:** all the users have a user profile that describes the configuration for a specific user, including the user's access permission for the applications on demand, user type (FF, LE or EMS) and preferences settings.

- **Service Authorizations:** this module sets the authorization for the service required at each user level.
- **Downloadable Applications:** these are the downloadable applications on demand.
- **Online Applications:** these are the applications on demand that are directly on line when the user perform authentication.
- **Stream Applications:** applications that can be streamed on demand.
- **Mobility:** this module manages mobility among the three IMS architectures (FF Network, LE Network and EMS Network).
- **Core framework:** this module performs all the basic functionalities.
- **IMS client:** this module performs the functionality of IMS client allowing: registration on IMS, invite a session, file transfer using MSRP protocol and so on.
- **Reasoner:** this module performs reasoning based on rules that consider emergencies for a specific area and all users near that area.
- **Context:** this module manages all Context parameters based on (GEO Coordinates of the PSCD Users, the battery level of the device, network traffic and so on.

Figure 8 describe a specific configuration of the PICO architecture where it is possible to identify a hierarchy of PSCS that enables collaboration between the different PSCS belonging to different Public Safety groups (FF, LE, and EM). It is important to notice that PSCS synchronization is not considered in the PICO project.

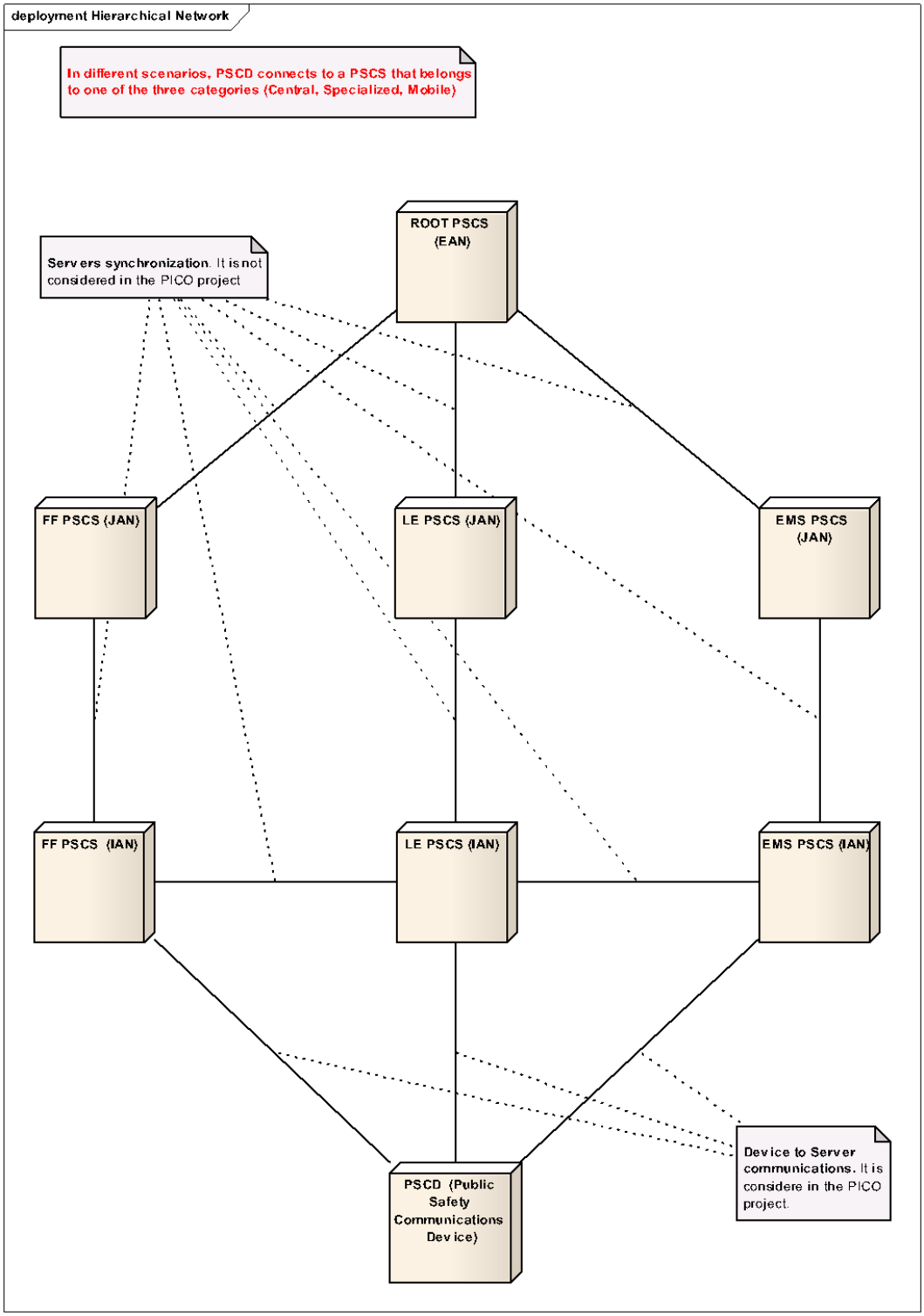

Figure 9: IMS Hierarchical Network

2.4 Mobility

In the scenario of IMS Internetworking (Figure 10) there are three different IMS networks, one of Fire Fighter, one of Law Enforcement and one of Emergency Medical Services. Each user has its home network, but we supposed that in the location of the emergency scenario not all the three networks can be reached.

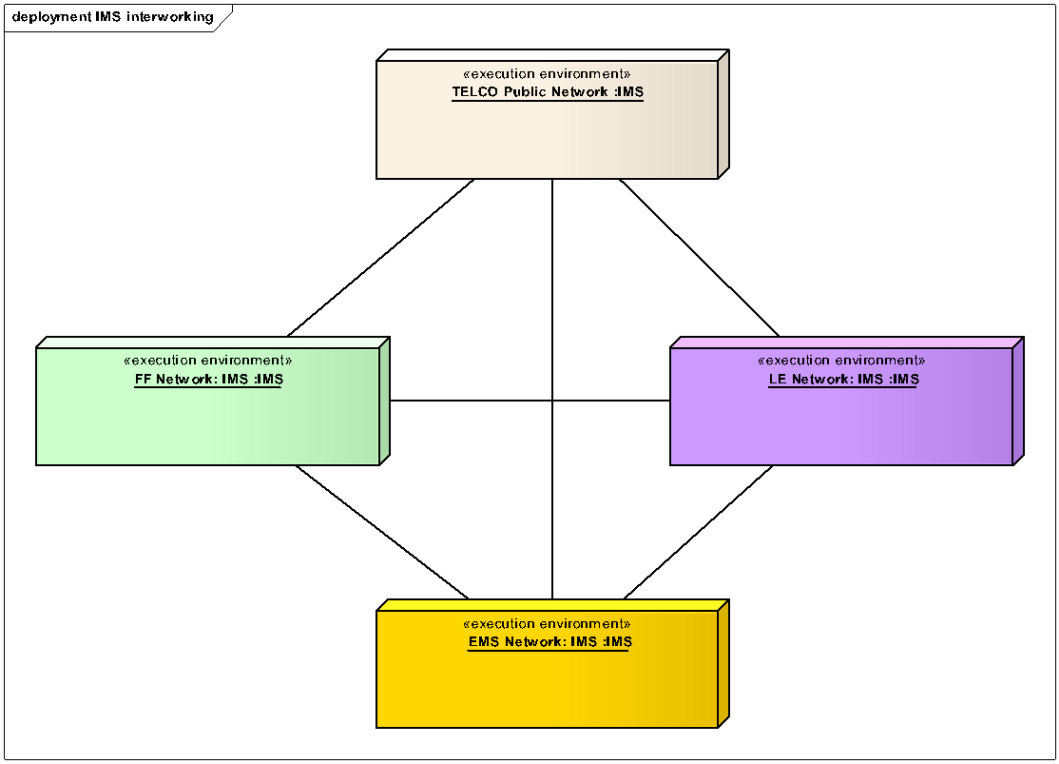

Figure 10: IMS Internetworking

Consider a scenario where Fire Fighter access network is not reachable.

Fire Fighter PSCD can register itself to the Law Enforcement Network that is connected with fire fighter home network. In this case the authorization and authentication is performed by a FF Public Safety Communications Server.

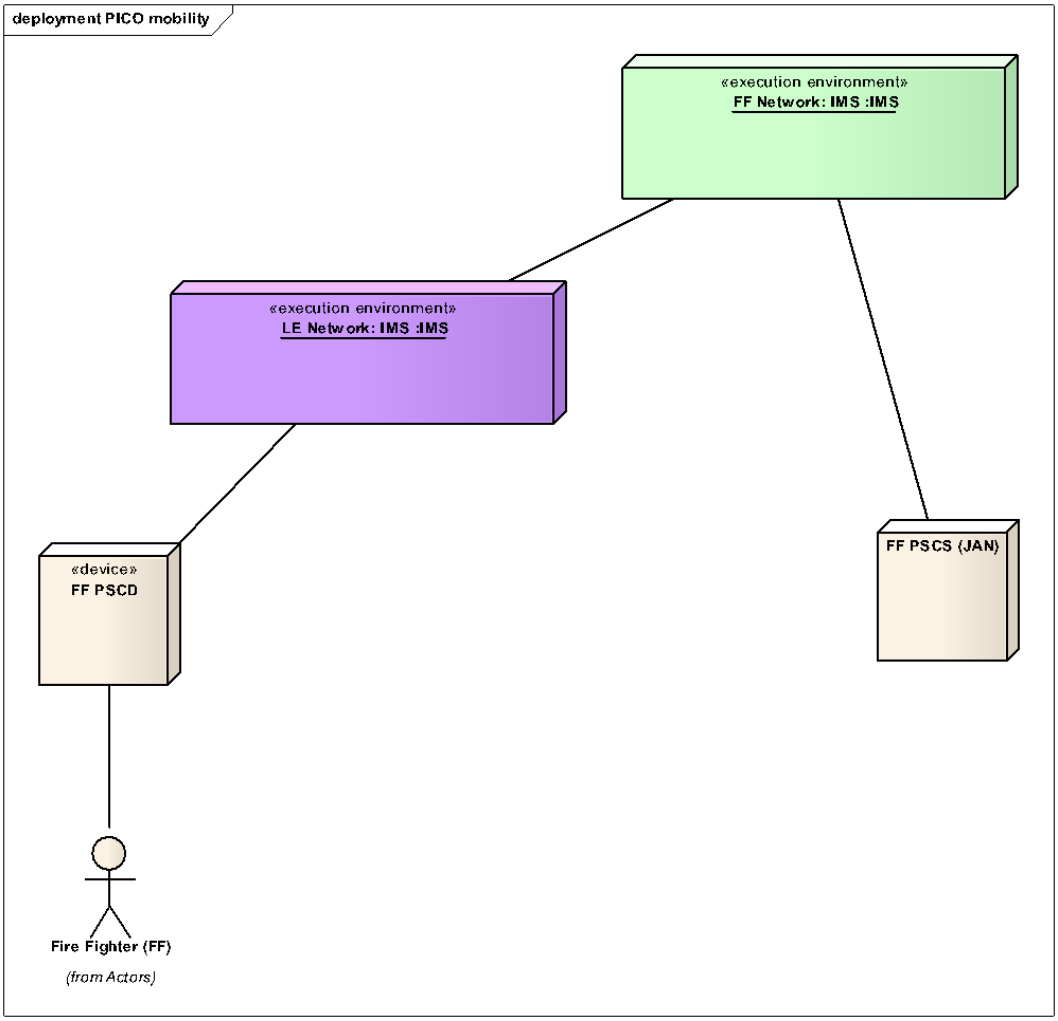

Figure 11: PICO mobility

2.5 Applications On Demand

Another aspect to be managed in an application-centric network is the context of application. Assumed that the framework is extended for the application sharing (actually is one of the goals of PICO), the operating system should handle the applications that could be installed onto device. Many are the contextual parameters that could be analyzed for each device when an application starts downloading, for instance:

- Disk usage
- Location
- Battery Level
- Network traffic
- …

Considering these parameters, an application could be downloaded or not, or in case of low disk space, a lite version of an application could be downloaded instead of full version, or an application with less media usage instead of full media in case of low battery/bad network.

PICO already takes care of all these assumptions adding more context parameters such as:

- User account and privileges
- Existing emergencies in a geographical area
- Status and proximity of neighbors (Paramedics, FF and Police)
- …

This paragraph describes the possible applications on demand that a user can choose among the available applications, then download and run locally

We have divided the applications into different classes; three classes are specific for type of user (EMS Applications, LE Applications and FF Applications), the other classes are instead shared by all the users (Workforce, Incident Command and On board resources status).

The Figure 12 describes the classification of applications on demand.

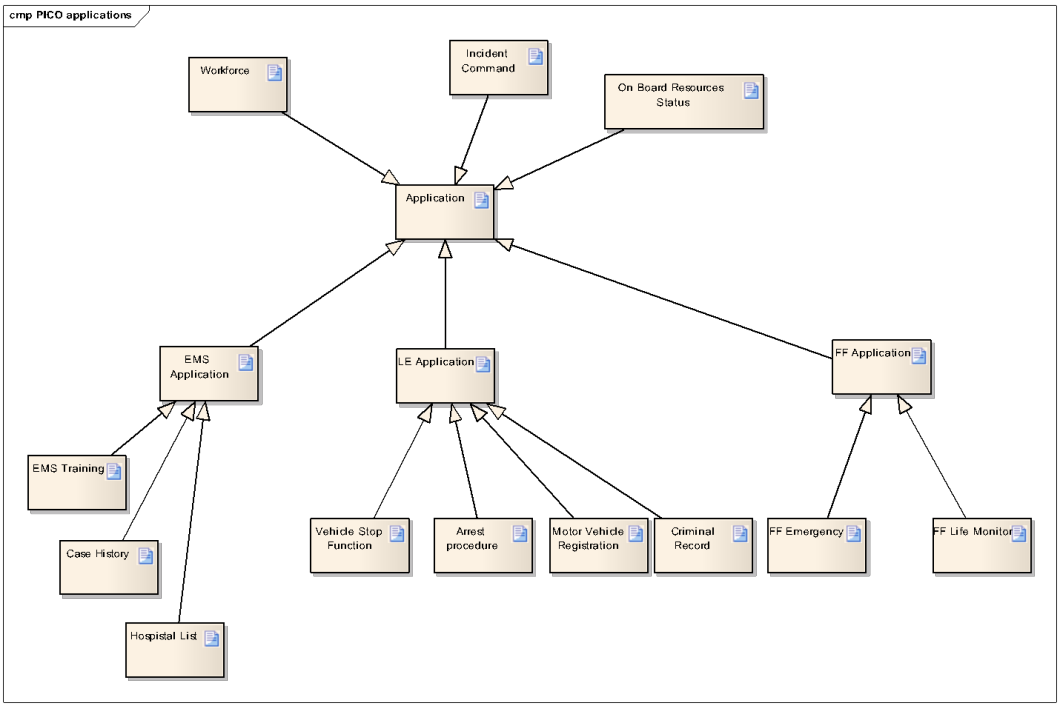

Figure 12: Applications on demand

The **Workforce application** is the main application, when all the users belong to the same group (or EMS or LE or FF), which performs the following functionalities:

- Online workforce on Google Maps;
- Online workforce inside building (using building maps if available);
- User exported services (depends on profile and device features);
- Alarm notification (e.g. fire fighter in troubles, etc.);
- Start communication (audio, video, chat);
- Create collaboration sessions;
- Filter for specific user roles;
- Filter for specific device features;
- Incident hot zones (red, green) and identification of risk's type (contamination, fire, etc.).

The **Incident command application** is the main application in case of different type of users present on the incident place, in this case where there are interdisciplinary group. This application performs the following functionalities:

- Grouping Functionality: the ability to create an interdisciplinary group to put in touch dissimilar user on incident site;
- Workforce Interoperability: fill user group on the Map (Allocation of all group of user on the map with its position and available services);
- Video, Voice and Chat communication between users of the groups;
- List of all services exported by users (Textual);
- Broadcast Messages:

- Emergency Button to ask for help.

The **On Board Resources Status application** performs a continual and up to date status of all on board resources (Inventory, available cams, sensors, and so on).

The following paragraphs describe the specific application: EMS applications, LA applications and FF applications.

2.5.1 EMS Applications

These applications are specific applications of Emergency Medical Services Users and they are divided into three categories:

- **EMS training**: the user can download emergency procedures from sources (as instructional aides of poison center and other instruction of EMS training packages).
- **Case History**: when a user acquires personal information about the patient he can download an application to view a history of the patient (previous hospitalize, possible allergy, etc.)
- **Hospital list**: the user can download an application to view a list of available hospitals, the distance from the incident location and all the available services.

2.5.2 LE Applications

These applications are specific applications of Law Enforcement Users and are divided into four categories:

- **Vehicle Stop Function**: is a special application that is active when the police identified a suspect vehicle. Vehicle position is sent in Central station; Vehicle's data is collected through place recognition and displayed on device, moreover video camera on the officer's vehicle dashboard begins recording video of the vehicle that can be accessed at any time, on-demand, by the authorized PSCDU users.
- **Arrest procedure**: collects all information about the person arrested (The PSCD submits the scan data to the biometric ID database for identification). Then the information will be used by transport unit for transport him to the prison. Moreover this information will be used by tow truck for vehicle transportation and tow report.
- **Motor Vehicle Registration**: allows access to the MVR Database to get as much detail as possible on the vehicle. All information will be displayed directly on the PSCDU device.
- **Criminal Record**: allows access to the CR Database to get as much detail as possible on the person. All information will be displayed directly on the PSCDU device.

2.5.3 FF Applications

These applications are specific applications of Fire Fighters Users and they are divided into two categories:

- **FF Emergency**: is an application that automatically (through risk recognition) or manually (through PSCDU button) communicates with paramedics or other FF, a state of emergency
- **FF Life Monitor**: through this application, in an Incident Area Network, the EMS unit outside the apartments monitors the vital signs of all the firefighters in and around the fire scene.

3 Use Cases

This chapter contains a sequence diagram of a typical PICO Application use case and an example of application on demand the spatial localization.

3.1 Typical Use Case

The following figure describes a typical use case of PICO Application. The first step is the authentication (see 3.1.1) of the user and of the server on IP Multimedia Subsystem. This operation is performed at the home network or, in case of own access network is not reachable, at the visited network.

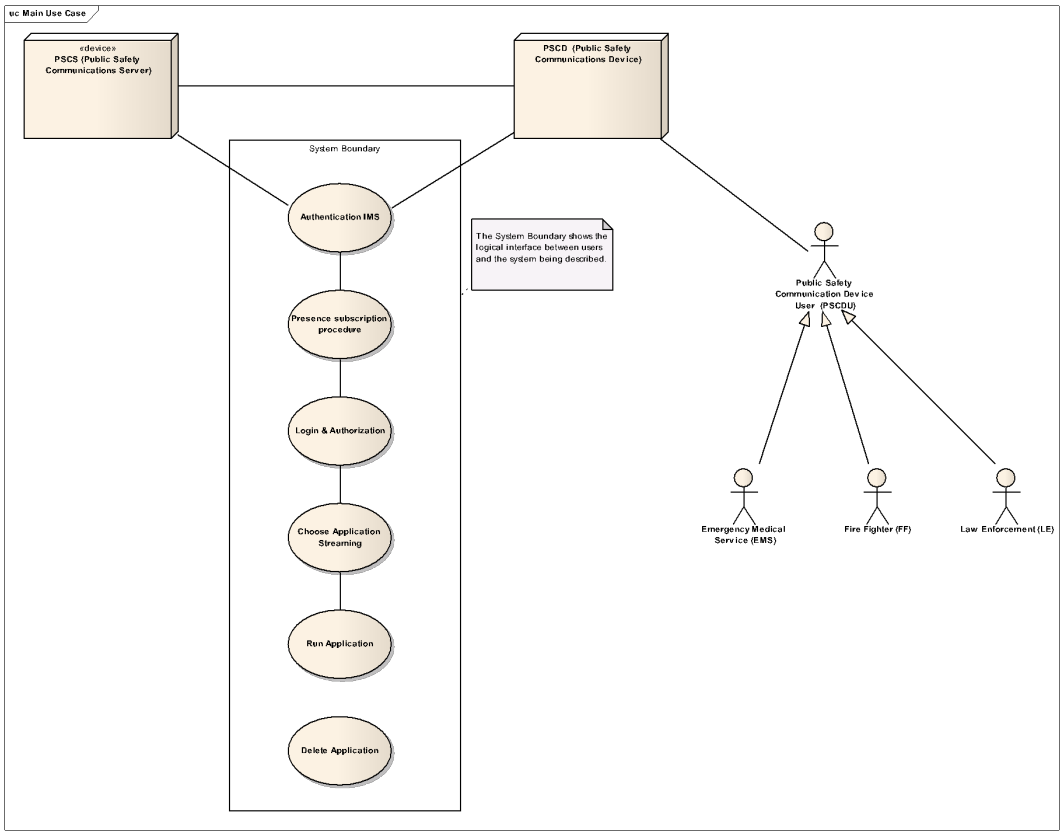

Figure 13: Main Use Case

After registration to IP Multimedia Subsystem (IMS) of both the active element (Public Safety Communication Device and Public Safety Communication Server) PSCS performs the presence subscription procedure. In this document we also describe process of PSCD presence update (see 3.1.2).

Subsequently the user performs the application server authentication (see 3.1.3) by a sip invite message and a PICO Application home page is showed.

After Login to the PICO Application, the user can view a list of downloadable applications and hence choose one (3.1.4). When download is completed, the user can locally run the application (3.1.5) and subsequently delete it (3.1.6).

The following paragraphs describe these steps.

3.1.1 Authentication IMS

Both the entities (PSCD and PSCS) are client of IP Multimedia Subsystem and they must perform authentication on IMS. This is performed by a sip message (register) and there are two scenarios, one when the user performs authentication on the own access network and one when the own access network is not reachable and the user performs authentication at a visited network.

The most simple scenario is when the own access network is reachable and the user (for example a Fire Fighter user) can register itself to the Home Network.

The Figure 14 describes this scenario. Both Public Safety Communication Device (PSCD) and Public Safety Communication Server (PSCS) perform a registration with Registrar Server (belongs to IP Multimedia Subsystem).

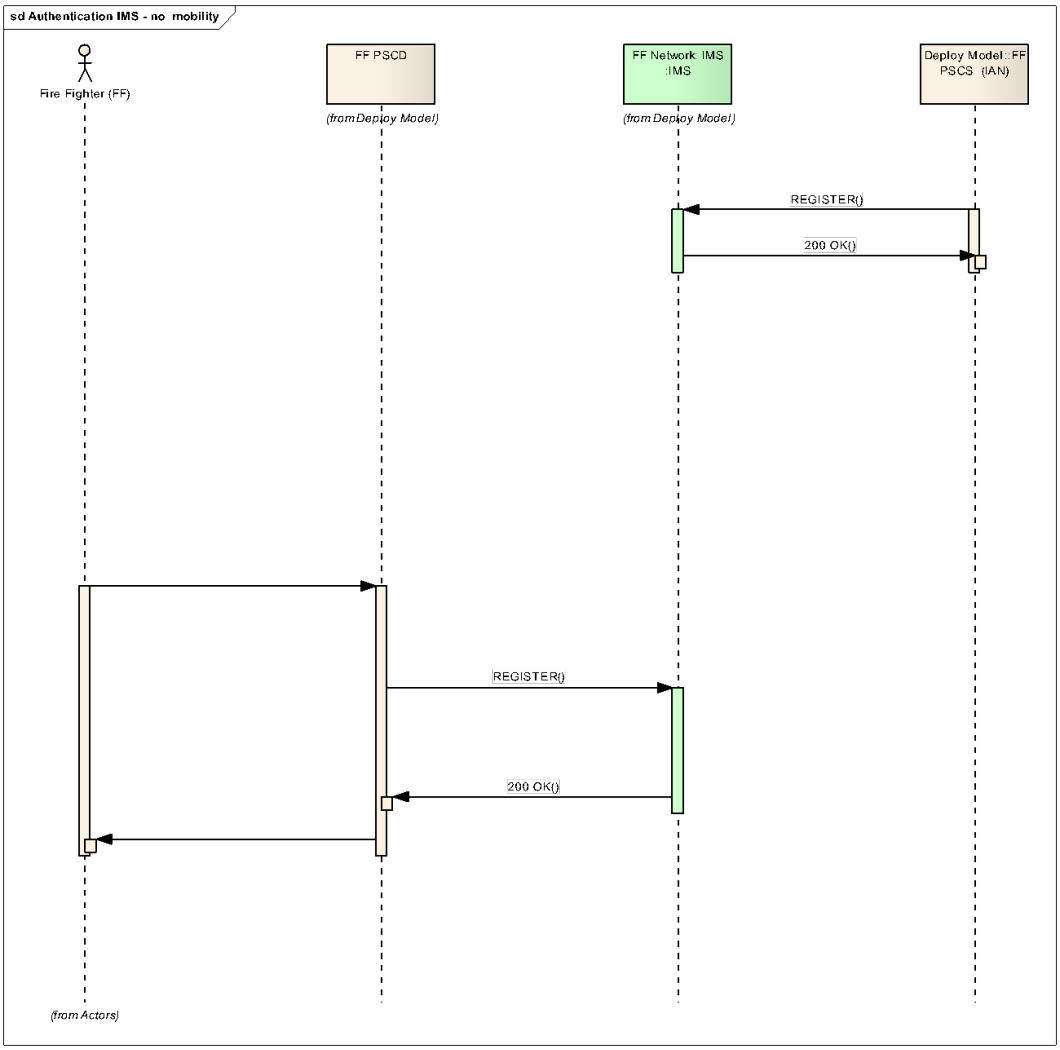

Figure 14: Authentication IMS - no mobility

In case of user own access network is not reachable, the visited network forwards all the messages of user to his home network. This scenario is described by a Figure 15.

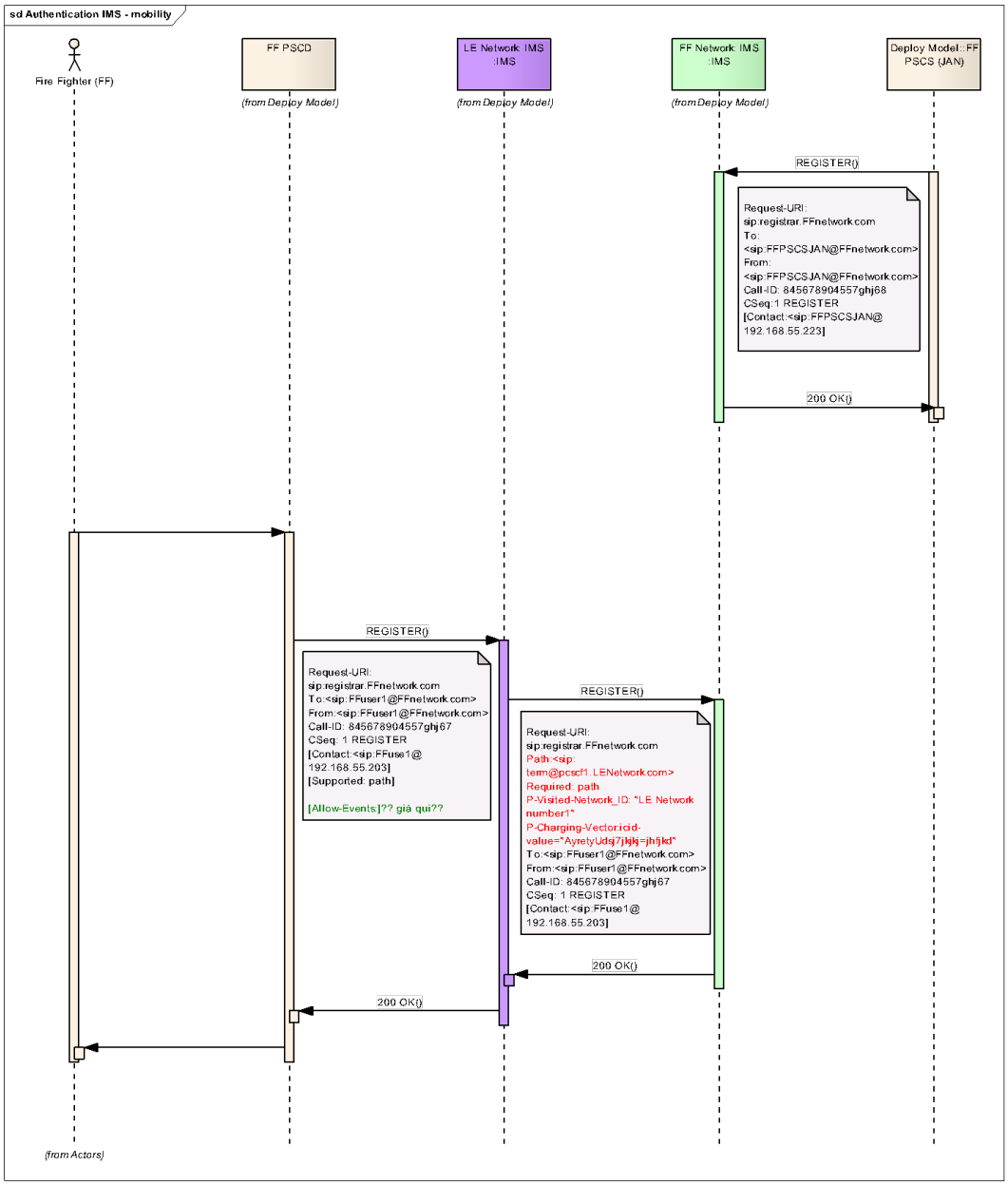

Figure 15: Authentication IMS – mobility

In all scenarios the authentication procedure is performed by a REGISTER Message; it is a SIP message described in the following paragraph.

3.1.1.1 REGISTER Message

REGISTER requests add, remove, and query bindings. A REGISTER request can add a new binding between an address-of-record and one or more contact addresses. Registration on behalf of a particular address-of-record can be performed by a suitably authorized third party. A client can also remove previous bindings or query to determine which bindings are currently in place for an address-of record.

The following header fields, except Contact, must be included in a REGISTER request. A Contact header field may be included:

- **Request-URI**: The Request-URI names the domain of the location service for which the registration is meant (for example, "sip: chicago.com"). The "userinfo" and "@" components of the SIP URI must not be present.
- **To**: The To header field contains the address of record whose registration is to be created, queried, or modified. The To header field and the Request-URI field typically differ, as the former contains a user name. This address-of-record MUST be a SIP URI or SIPS URI.
- **From**: The From header field contains the address-of-record of the person responsible for the registration. The value is the same as the To header field unless the request is a third-party registration.
- **Call-ID**: All registrations from a UAC should use the same Call-ID header field value for registrations sent to a particular registrar. If the same client were to use different Call-ID values, a registrar could not detect whether a delayed REGISTER request might have arrived out of order.
- **CSeq**: The CSeq value guarantees proper ordering of REGISTER requests. A UA MUST increment the CSeq value by one for each REGISTER request with the same Call-ID.
- Contact: REGISTER requests MAY contain a Contact header field with zero or more values containing address bindings.

A registrar is a user agent server that responds to REGISTER requests and maintains a list of bindings that are accessible to proxy servers and redirect servers within its administrative domain. A registrar has to know (for example, through configuration) the set of domain(s) for which it maintains bindings. REGISTER requests must be processed by a registrar in the order that they are received.

REGISTER requests must also be processed atomically, meaning that a particular REGISTER request is either processed completely or not at all. Each REGISTER message must be processed independently of any other registration or binding changes.

When receiving a REGISTER request, a registrar follows these steps:

- The registrar inspects the Request-URI to determine whether it has access to bindings for the domain identified in the Request-URI. If not, and if the server also acts as a proxy server, the server should forward the request to the addressed domain, following the general behavior for proxying messages (for details see specification SIP specification [3]).
- A registrar should authenticate the user agent client. Mechanisms for the authentication of SIP user agents are described in SIP specification [3].
- The registrar extracts the address-of-record from the To header field of the request. If the address-of-record is not valid for the domain in the Request-URI, the registrar sends a404 (Not Found) response and skips the remaining steps.
- The registrar checks whether the request contains the Contact header field. If not, it skips to the last step. If the Contact header field is present, the registrar checks if there is one Contact field value that contains the special value "*" and an Expires field. If the request has additional Contact fields or an expiration time other than zero, the request is invalid, and the server returns a 400 (Invalid Request) and skip the remaining steps. If not, the registrar checks whether the Call-ID agrees with the

value stored for each binding. If not, it removes the binding. If it does agree, it removes the binding only if the CSeq in the request is higher than the value stored for that binding. Otherwise, the update must be aborted and the request fails.

- The registrar now processes each contact address in the Contact header field in turn. For each address, it determines the expiration interval as follows:
    - If the field value has an "expires" parameter, that value must be taken as the requested expiration.
    - If there is no such parameter, but the request has an Expires header field, that value must be taken as the requested expiration.
    - If there is neither, a locally-configured default value must be taken as the requested expiration.

For each address, the registrar then searches the list of current bindings using the URI comparison rules. If the binding does not exist, it is tentatively added. If the binding does exist, the registrar checks the Call-ID value. If the Call-ID value in the existing binding differs from the Call-ID value in the request, the binding must be removed if the expiration time is zero and updated otherwise. If they are the same, the registrar compares the CSeq value. If the value is higher than that of the existing binding, it must update or remove the binding as above. If not, the update must be aborted and the request fails. This algorithm ensures that out-of-order requests from the same user agent are ignored.

Each binding record records the Call-ID and CSeq values from the request.

The binding updates must be committed (that is, made visible to the proxy or redirect server) if and only if all binding updates and additions succeed. The request fails with a 500 (Server Error) response and all tentative binding updates must be removed when any one of them fails (for example, because the back-end database commit failed).

3.1.2 Presence subscription procedure

The presence subscription has more importance for Workforce or Incident Command applications, a server PSCS could subscribe at all the event of presence for all the users that are in to the place of incident.

The presence service serves to accept information, store it, and distribute it. The information stored is (unsurprisingly) presence information. The presence service has two distinct sets of "clients": one set of clients, called **presentities**, provides presence information to be stored and distributed. The other set of clients, called **watchers**, receives presence information from the service.

There are two kinds of **watchers**, called fetchers and **subscribers**. A fetcher simply requests the current value of some presentity's presence information from the presence service. In contrast, a **subscriber** requests notification from the presence service of (future) changes in some presentity's presence information. The procedure is simpler and is described by Figure 16.

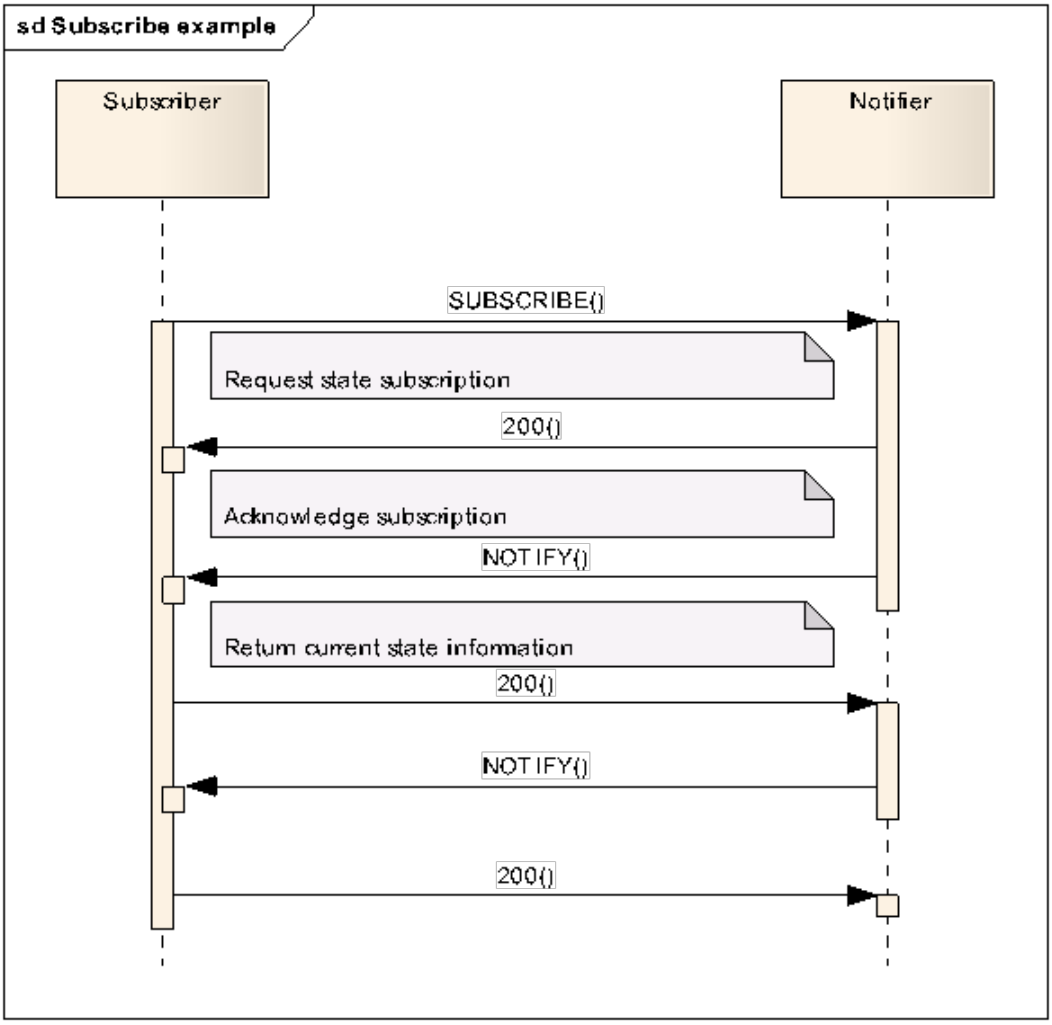

Figure 16: Flow of messages – SUBSCRIBE

The following paragraphs described all the SIP messages used to perform this procedure. The Figure 17 describes a sequence diagram for subscription procedure of PICO Application.

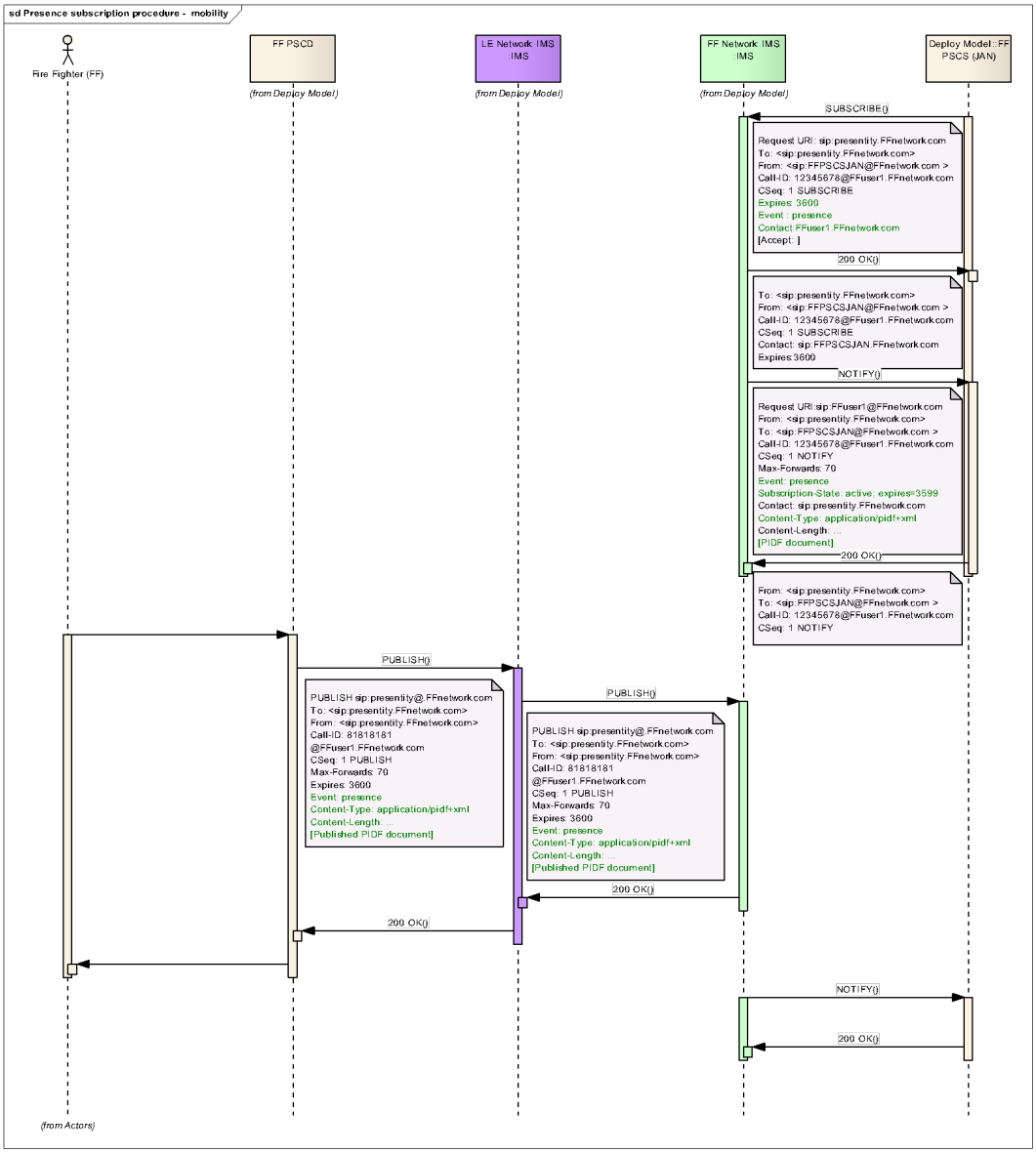

Figure 17: Presence service

3.1.2.1 SUBSCRIBE Message

The ability to request asynchronous notification of events proves useful in many types of SIP services for which cooperation between end-nodes is required.

The general concept is that entities in the network can subscribe to resource or call state for various resources or calls in the network, and those entities (or entities acting on their behalf) can send notifications when those states change.

A typical flow of messages would be that shows by the Figure 16.

"SUBSCRIBE" is added to the definition of the element "Method" in the SIP message grammar. The SUBSCRIBE method is used to request current state and state updates from a remote node.

SUBSCRIBE requests should contain an "Expires" header (defined in SIP [3]). This expires value indicates the duration of the subscription. In order to keep subscriptions effective beyond the duration communicated in the "Expires" header, subscribers need to refresh subscriptions on a periodic basis using a new SUBSCRIBE message on the same dialog as defined in SIP [3].

If no "Expires" header is present in a SUBSCRIBE request, the implied default is defined by the event package being used. Notifiers may also wish to cancel subscriptions to events; this is useful, for example, when the resource to which a subscription refers is no longer available (for more details see specification [4]).

Identification of events is provided by three pieces of information:

- Request URI: contains enough information to route the request to the appropriate entity per the request routing procedures outlined in SIP [3]. It also contains enough information to identify the resource for which event notification is desired, but not necessarily enough information to uniquely identify the nature of the event (e.g., "sip:adam@dynamicsoft.com" would be an appropriate URI to subscribe to for my presence state; it would also be an appropriate URI to subscribe to the state of my voice mailbox).
- Event Type: Subscribers include exactly one "Event" header in SUBSCRIBE requests, indicating to which event or class of events they are subscribing. The "Event" header will contain a token which indicates the type of state for which a subscription is being requested. This token will be registered with the IANA and will correspond to an event package which further describes the semantics of the event or event class. The "Event" header may also contain an "id" parameter. This "id" parameter, if present, contains an opaque token which identifies the specific subscription within a dialog. An "id" parameter is only valid within the scope of a single dialog.
- (optionally) message body: If the event package to which the event token corresponds defines behavior associated with the body of its SUBSCRIBE requests, those semantics apply.

An example of SUBSCRIBE message in PICO Application is described by Figure 17 (zoom image SUBSCRIBE).

3.1.2.2 NOTIFY Message

NOTIFY messages are sent to inform subscribers of changes in state (see Figure 16) to which the subscriber has a subscription. Subscriptions are typically put in place using the SUBSCRIBE method; however, it is possible that other means have been used. A NOTIFY does not terminate its corresponding subscription; in other words, a single SUBSCRIBE request may trigger several NOTIFY requests.

Identification of events being reported in a notification is very similar to that described for subscription to events (see 3.1.2.1).

As in SUBSCRIBE requests, NOTIFY "Event" headers will contain a single event package name for which a notification is being generated. The package name in the "Event" header must match the “Event" header in the corresponding SUBSCRIBE message. If an "id" parameter was present in the

SUBSCRIBE message, that "id" parameter must also be present in the corresponding NOTIFY messages.

Event packages may define semantics associated with the body of their NOTIFY requests; if they do so, those semantics apply. NOTIFY bodies are expected to provide additional details about the nature of the event which has occurred and the resultant resource state (to more details see specification [4]).

When a SUBSCRIBE request is answered with a 200-class response, the notifier immediately constructs and sends a NOTIFY request to the subscriber. When a change in the subscribed state occurs, the notifier should immediately construct and send a NOTIFY request, subject to authorization, local policy, and throttling considerations.

A NOTIFY request is considered failed if the response times out, or a non-200 class response code is received which has no "Retry-After" header and no implied further action which can be taken to retry the request.

3.1.2.3 PIDF

The Presence Information Document Format (PIDF) specifies the baseline XML-based format for describing presence information. One of the characteristics of the PIDF is that the document always needs to carry all presence information available for the presentity. In some environments where low bandwidth and high latency links can exist, it is often beneficial to limit the amount of transported information over the network. For details see specification [8].

3.1.3 Login & Authentication

The PSCD use INVITE SIP message to login at PSCS and after that a HTTP session is created and correlated with the SIP session.

A home page is displayed to the user where he can choose which application he could download (see 3.1.4).

The Figure 18 describes a flow between PSCD and PSCS.

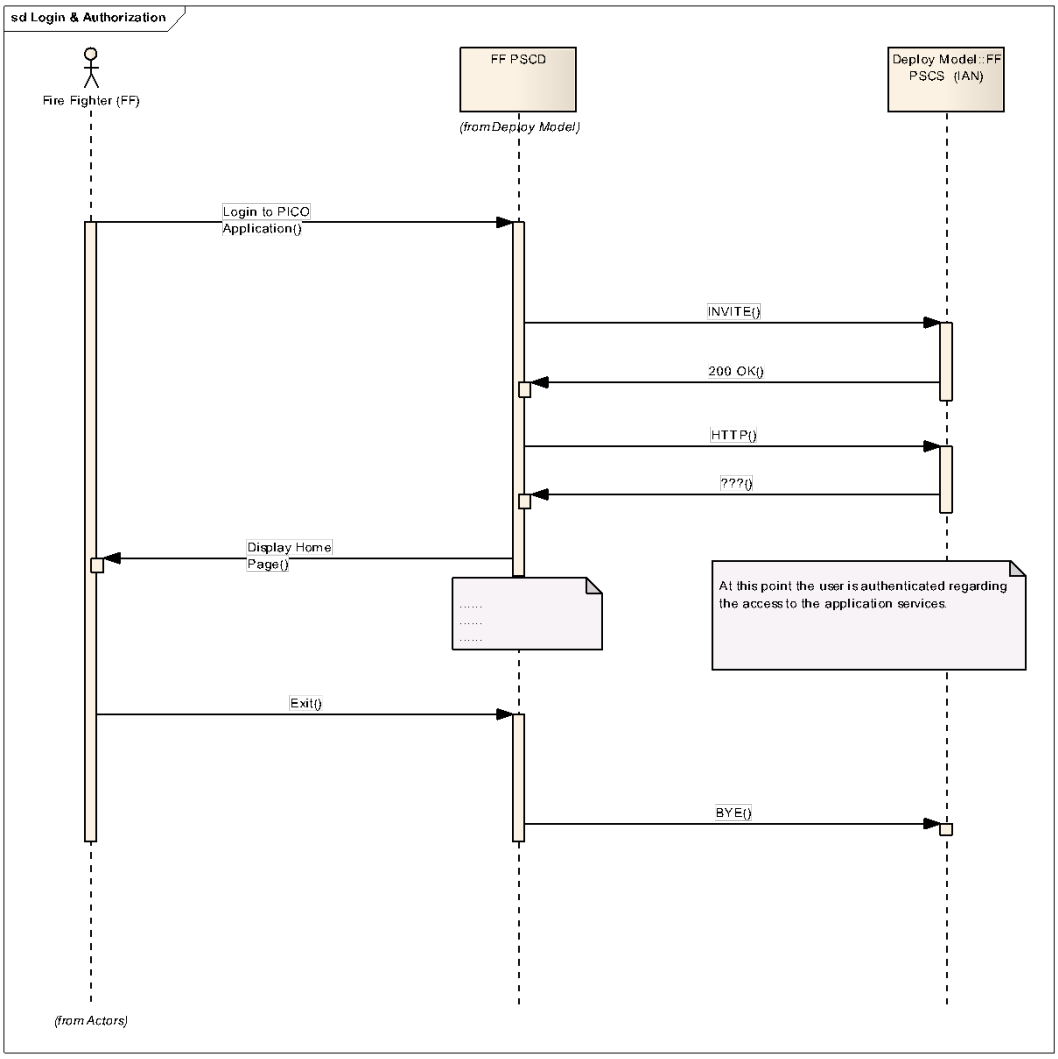

Figure 18: Login and Authentication

3.1.3.1 INVITE Message

When a user agent (UA) client desires to initiate a session (for example, audio, video, or a game), it formulates an INVITE request. The INVITE request asks a server to establish a session. This request may be forwarded by proxies, eventually arriving at one or more user agent server (UAS) that can potentially accept the invitation. These UASs will frequently need to query the user about whether to accept the invitation. After some time, those UASs can accept the invitation (meaning the session is to be established) by sending a 2xx response. If the invitation is not accepted, a 3xx, 4xx, 5xx or 6xx response is sent, depending on the reason for the rejection.

A 2xx response to an INVITE establishes a session and it also creates a dialog between the UA that issued the INVITE and the UA that generated the 2xx response.

Since the initial INVITE represents a request outside of a dialog, its construction follows the procedures of request message (see specification [3]). SIP requests are distinguished by having a Request-Line for a startline. A Request-Line contains a method name (INVITE), a Request-URI, and the protocol version separated by a single space (SP) character.

- **Request-URI**: The Request-URI is a SIP or SIPS URI as described in specification [3]) or a general URI (RFC 2396 [9]). It indicates the user or service to which this request is being addressed. In case of INVITE message the initial Request-URI of the message is set to the value of the URI in the To field.
- **SIP-Version**: Both request and response messages include the version of SIP in use, and follow [H3.1] (with HTTP replaced by SIP, and HTTP/1.1 replaced by SIP/2.0) regarding version ordering, compliance requirements, and upgrading of version numbers. To be compliant with this specification, applications sending SIP messages must include a SIP-Version of "SIP/2.0".

A valid SIP request formulated by a UAC must, at a minimum, contain the following header fields (all of these header fields are mandatory in all SIP requests):

- **To**: The To header field specifies the desired "logical" recipient of the request, or the address-of-record (AOR) of the user or resource that is the target of this request. There is no "tag" parameter since the dialog is not established yet.
- **From**: The From header field indicates the logical identity of the initiator of the request, possibly the user's address-of-record. The "tag" parameter identifies this UA as a peer of the dialog.
- **CSeq**: The CSeq header field serves as a way to identify and order transactions. It consists of a sequence number and a method, matching that of the request.
- **Call-ID**: The Call-ID header field acts as a unique identifier and must be the same for all requests and responses sent by either UA in a dialog.
- **Max-Forwards**: The Max-Forwards header field serves to limit the number of hops a request can transit on the way to its destination. It is decremented by one at each hop.
- **Via**: The Via header field indicates the transport (UDP) used for the transaction and identifies the location where the response is to be sent. The "branch" parameter is used to identify the transaction created by that request. It is mandatory and must always begin with the characters "z9hG4bK".

Additional processing is required for the specific case of INVITE.

3.1.4 Choose Application on demand

When the PSCD completes the bootstrap and the user is authenticated by the PICO framework, the PSCD device will display the user the main application page (PICO dashboard). This PICO dashboard displays the list of PICO applications available to the users depending on the context and device capabilities. These applications are not usually available on the device but they must be downloaded or streamed from the PSCS. When user chooses an application the PICO framework will start the application on the PSCS and stream it to the PSCD. In other situations, the PSCD will download the application, for example packed into a Java applet, or will start a web application using web 2.0 (i.e. Ajax) architecture.

3.1.5 Run Application

When download is completed, the user can unzip the file, if it is necessary, and locally run the application. The Figure 19 shows the interaction between user and PSCD.

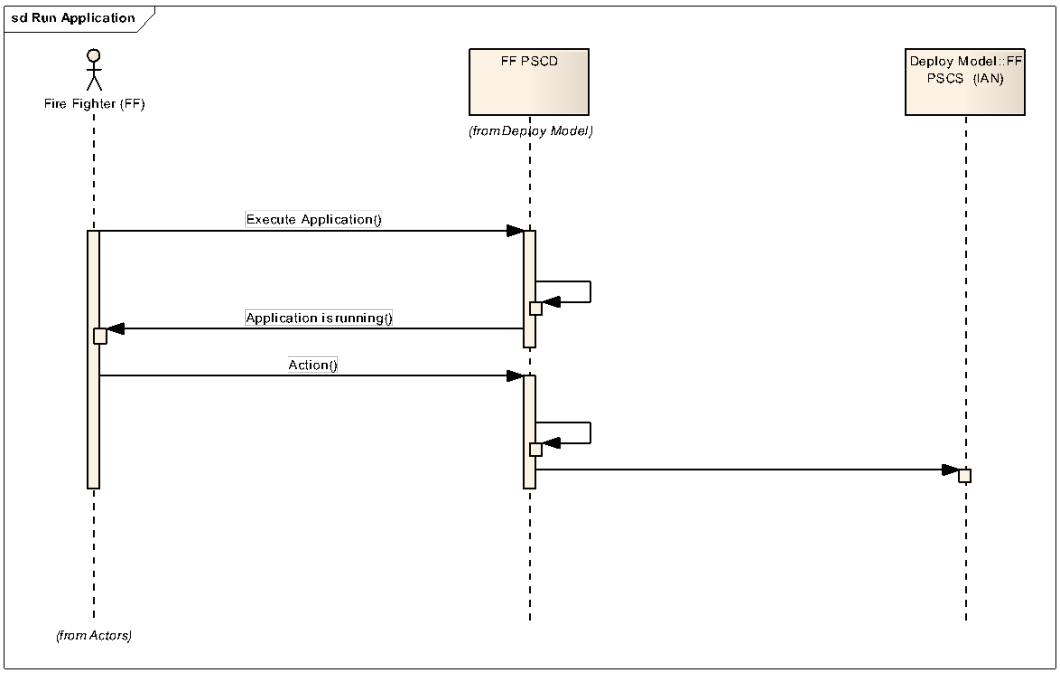

Figure 19: Run Application

3.1.6 Delete Application

When the application doesn't already exist on the PSCD, the user can delete it (see Figure 20).

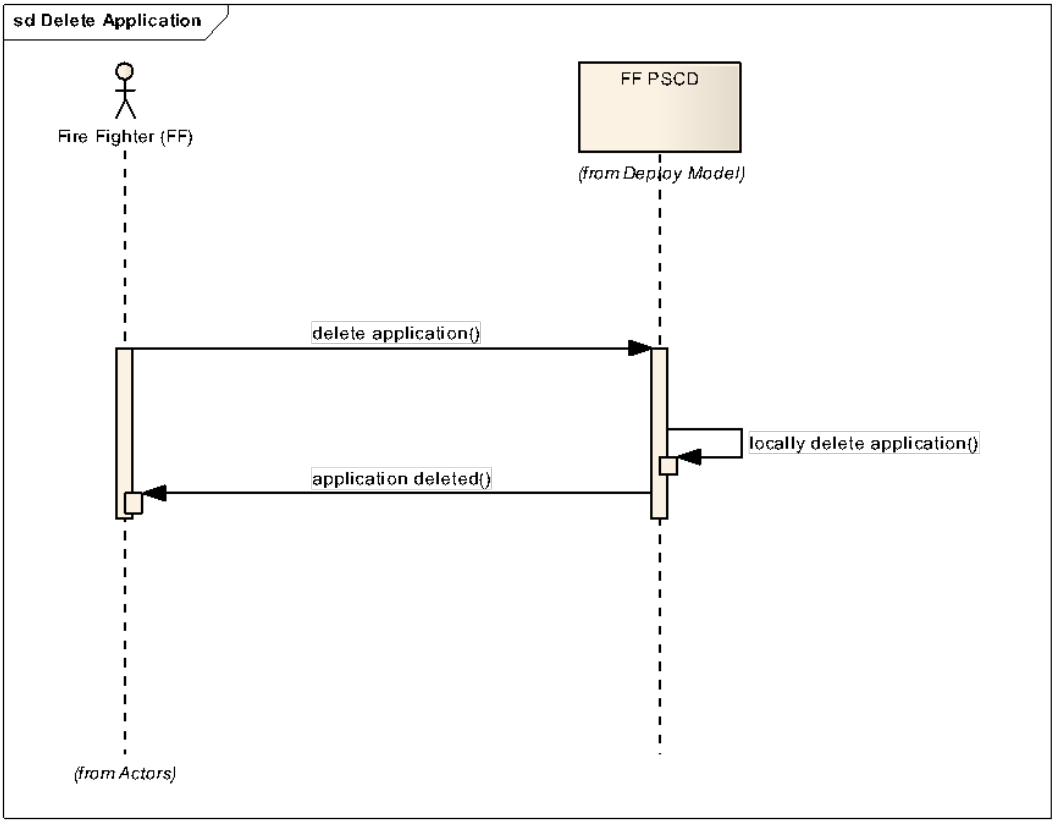

Figure 20: Delete Application

4 Prototype technologies

4.1 Operating system: Google Android

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The actual version of Android SDK is a beta version; it provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

The following diagram shows the major components of the Android operating system, each section is described in more detail below.

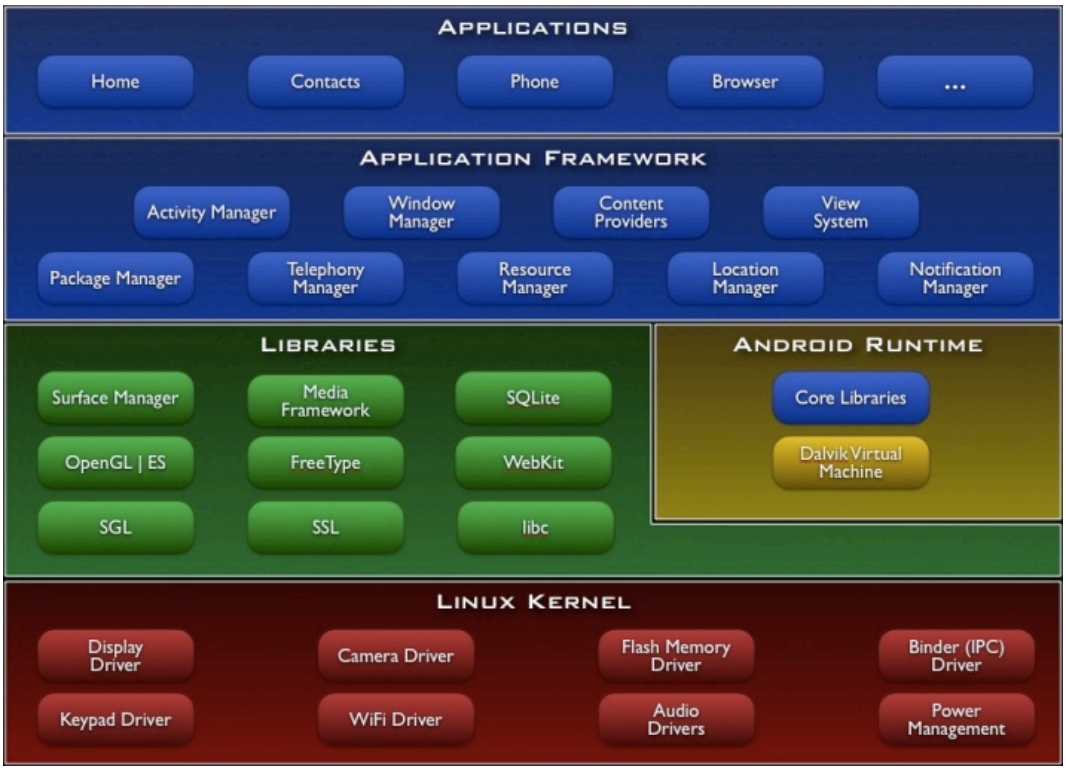

Figure 21: Google Android

4.2 Applications

Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

4.2.1 Application Framework

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security

constraints enforced by the framework). This same mechanism allows components to be replaced by the user. Underlying all applications is a set of services and systems, including:

- A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser
- Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all applications to display custom alerts in the status bar
- An Activity Manager that manages the life cycle of applications and provides a common navigation back stack

4.2.2 Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

- System C library: a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
- Media Libraries: based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- Surface Manager: manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
- LibWebCore: a modern web browser engine which powers both the Android browser and an embeddable web view
- SGL: the underlying 2D graphics engine
- 3D libraries: an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- FreeType: bitmap and vector font rendering
- SQLite: a powerful and lightweight relational database engine available to all applications

4.2.3 Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

4.2.4 Linux Kernel

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

4.2.5 Other Features

Android supports a lot of useful functions, sometimes dependent on hardware capabilities, like the following:

- **Integrated browser** based on the open source WebKit engine
- **Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **SQLite** for structured data storage
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM Telephony** (hardware dependent)
- **Bluetooth, EDGE, 3G, and WiFi** (hardware dependent)
- **Camera, GPS, compass, and accelerometer** (hardware dependent)
- **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plug-in for the Eclipse IDE

4.2.6 Android Maps API

Google Android includes Google Maps between his own applications. But Android also allows creating personalized Google Map in other applications by the maps package.

The maps package allows applications to display and control a Google Map interface. To create a Google Map is necessary to extend MapActivity and implement a MapView in the layout.

This is not a standard package in the Android library. In order to use it, we must add a specific XML element, as a child of the `application` element, in AndroidManifest.xml file.

In order to use a MapView in an application, is needed to include the "android:apiKey" attribute in the MapView (or include a key in the MapView constructor).

4.3 IP Multimedia Subsystem

The following paragraphs provide an assessment of the most important Open Source IMS platforms, clients and API services. The analysis of all this software provides an overview of what could be useful for the creation of the IMS client for PICO demonstrator and running on portable devices.

This part of the document is divided in four parts: the first one is an introduction about all existing free IMS clients, the second one shows the most important free IMS platforms, the third and fourth

one describe two particular IMS client, the fifth one describes a set of API that implements several IMS services and the last one summarizes the conclusions about what could be useful to our project.

4.3.1 IMS platforms overview

In this paragraph with the word platform represents a set of elements and tools that interacts with an IMS application and supports it. We analyzed the Open Source IMS Core that supports several IMS clients (like UTC IMS client and IMS communicator), Mobicents platform and doubango framework.

The Open IMS Core, the first examined platform, is an implementation of IMS Call Session Control Functions (CSCFs) and a lightweight Home Subscriber Server (HSS), which together form the core elements of all IMS/NGN architectures as specified today within 3GPP, 3GPP2, ETSI TISPAN and the PacketCable initiative. The four components are all based upon Open Source software (e.g. the SIP Express Router (SER) or MySQL).

This platform is not created to product applications in a commercial context. Its sole purpose is to provide an IMS core reference implementation for IMS technology testing and IMS application prototyping for research purposes. The following image shows an overview of the architecture.

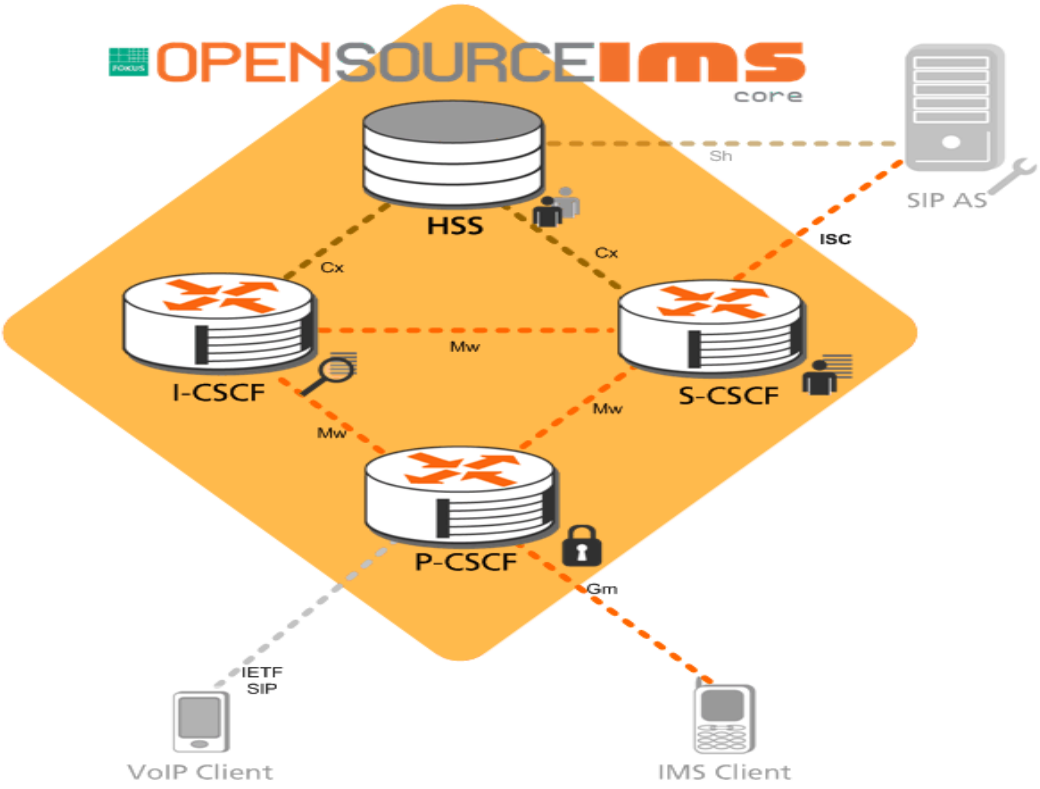

Figure 22: Open IMS Core overview.

The second platform considered in our analysis about free IMS platforms is Mobicents. It's a highly scalable event-driven application server. Mobicents is the first and only Open Source VoIP Platform

certified for JSLEE 1.0 compliance. It complements J2EE to enable convergence of voice, video, instant messaging and data in next generation applications.

In the scope of telecom Next Generation Intelligent Networks (NGIN), Mobicents fits in as a high-performance core engine for Service Delivery Platforms (SDP) and IP Multimedia Subsystem (IMS).

Mobicents enables the composition of Service Building Blocks (SBB) such as call control, billing, user provisioning, administration, and presence sensitive features. The JAIN SLEE specification allows popular protocol stacks such as SIP to be plugged in as resource adapters. The following figure describes the Mobicents architecture.

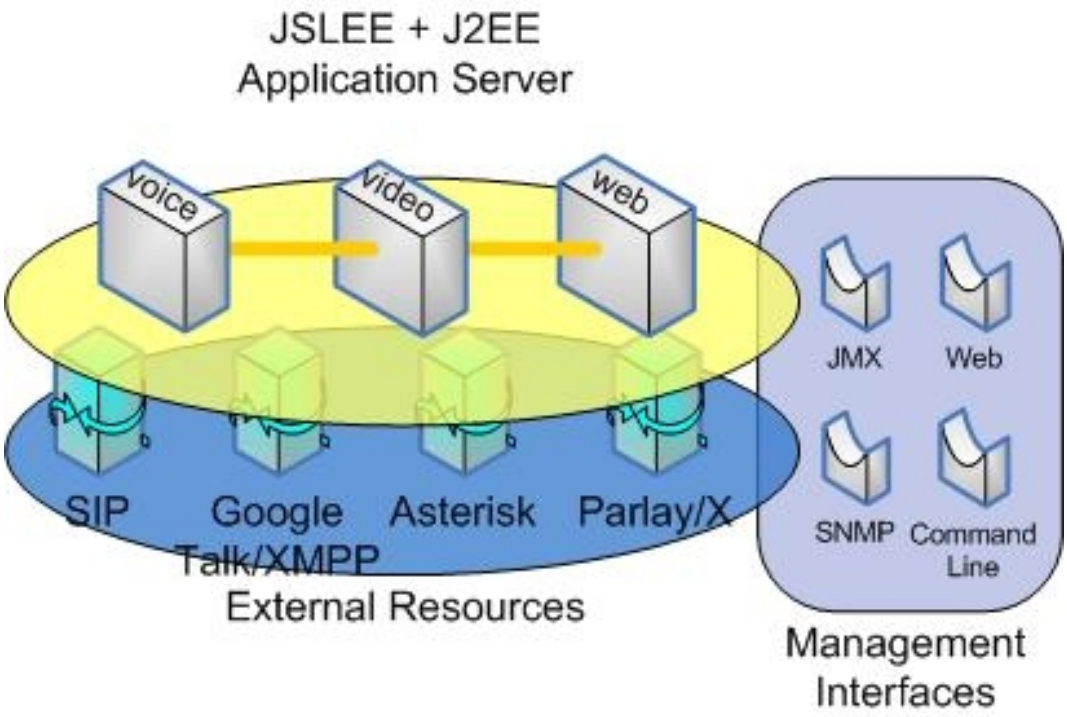

Figure 23: Mobicents solution overview

4.3.2 IMS clients

Over the last years, with the emergent effort on IMS Core Network development and NGN services, comes the need to have an IMS Client able to use and test all the new services and convergence scenarios made possible with SIP and the IMS architecture. Here we have identified some IMS clients that can be a good starting point for the construction of the PICO client:

- The Mercuro IMS client is freely available (closed-source)
- The UCT IMS client is available under the GPL (open source)
- The IMS Communicator is available under the GPL (open source)

- Open IMS client is available only commercially. Yet, there is also a free binary OpenIC_Lite version available right here

Mercuro IMS Client is the most complete IMS free client but it is closed source and the use of Mercuro IMS Client as framework (C++, C# or VB.NET) to develop other IMS services or clients is only for payment.

The Open IMS Client (available only commercially) is a framework that offers a programmable interface for development of various IMS applications. The word "Open" means extendible and should not be confused with "Open Source".

The Open IMS Client (Lite version) is a free IMS based soft phone. Also in this case, the word "Open" means extendible and should not be confused with "Open Source". Based on the architecture of OpenIC, OpenIC Lite is designed to highlight some of the rich features available with commercial IMS client. The use of the client that offers a framework for development of various IMS applications is not developed in Open IMS Client Lite.

For these reasons we focus our attention about the two GPL (open source client). Both clients are based on Open IMS Core.

4.3.2.1 UCT IMS Client

Regarding IMS client technologies, we report like first open source IMS client the UCT IMS Client. It is designed to work in conjunction with the Fraunhofer FOKUS Open IMS Core. At present the client is still in active development and there are several known bugs.

The client supports a system for authentication called AKA, and emulates IMS signaling as far as possible. The current version supports voice and video calls (numerous codecs), pager-mode instant messaging, Presence, an IPTV viewer and an XCAP client. The following list shows all the modules of UCT IMS Client:

- Methods to handle both SIP and IMS eXosip events
- Methods to handle both SIP and IMS interface events
- Methods to handle IMS eXosip events
- Methods to handle IMS interface events
- Main
- Media
- Preferences
- Presence
- SDP body helpers
- Methods to handle SIP eXosip events
- Methods to handle SIP interface events
- Sound converter
- Useful methods
- Watchers
- XCAP

Every module could contain several methods. For example the module “Methods to handle both SIP and IMS interface events” implements all the following methods:

- ***void set_mode()***: Sets the mode into either IMS or SIP mode, used only on startup
- ***void terminate_call()***: Terminates a call, method is the same for SIP and IMS mode
- ***void reject_call()***: Rejects a call, method is the same for SIP and IMS mode
- ***void common_start_im_session(constgchar *chat_uri_entry)***: Starts an Instant Messaging session as a result of sending a MESSAGE Opens an IM Window and brings correct tab to the foreground Method is the same for IMS and SIP mode. The parameter "chat_uri_entry" is URI to start the IM session with
- ***void common_send_dtmf(int val)***: Sends DTMF tones in an NFO message. The parameter "val" is the value of the DTMF tone to send

This client, written in C and based on eXosip (a library that hides the complexity of using the SIP protocol for mutlimedia session establishement), requires a Debian OS or a Linux-based Operating System.

Several related projects have resulted from the UCT IMS Client:

- **UCT Advanced IPTV**: This solution involves a SIP based Indirection server that facilitates an RTSP session between the UCT IMS Client and any RTSP supported media server. A 3rd Party RTSP supported media server is required.
- **UCT Policy Control Framework**: The UCT Policy Control Framework incorporates Policy and Charging Rule Function (PCRF) and Policy and Charging Enforcement Function (PCEF) functionality into the FOKUS Open Source IMS Core. XML Network level control policies are defined - the PCRF combines these policies with service information from the service control layer (e.g. P-CSCF) and creates policy enforcement rules to be enforced in the transport layer at the PCEF.
- **UCT Back-to-back User Agent**: The UCT Back-to-back agent is a simple server that sets up a call between two registered IMS clients. When incorporated with a web-page the user agent can be used as a click-to-dial server.
- **UCT IPtv Streaming Server**: The UCT IPtv Streaming Server allows broadcast video streaming over an IMS network. The project is no longer under active development, and users are directed to UCT Advanced IPTv for a more comprehensive and standards compliant IPTv implementation. However the Streaming server is still operational and remains for compatibility purposes.

The first version of this client, released in December 2006, was the 1.0.0. The last version, released in July 2008, is the 1.0.12. All software is released under the GNU General Public License version 3.

4.3.2.2 IMS Communicator

Through the evaluation work we analyze, like second solution, the IMS-Communicator. This project is based on the old version of the SIP-Communicator softphone. It is built on top of the JAIN-SIP RI (a full implementation of RFC 3261), in which contributions were also made to support the IMS SIP extensions defined by 3GPP and IETF. The IMS-Communicator media stack is provided by the Java Media Framework (JMF) API. Despite being written in Java and accomplishing multiplatform compliance, there are no plans for porting IMS-Communicator to the J2ME CDC environment.

The main development efforts made to extend SIP-Communicators' IMS-conformance focused in the IMS Registration and Authentication, and the IMS Session Establishment. For the IMS Registration and Authentication, the support of an IMPI (IP Multimedia Private Identity), the authentication

algorithm AKAv1 (with the MILENAGE 3GPP reference algorithm), the subscription to the “reg” event package and the Security Agreement mechanism were implemented. Regarding the IMS Session Establishment, the Precondition Mechanism was implemented, as also added the support of Early Media and Call Transfer. The IMS-Communicator features also include a Setup wizard, Voice and Video calls, Dial history, Contact list, IM and Presence support. The interoperability with the Open IMS Core is considered an important feature, so the IMS-Communicator project will continue to support it, as it does since its release.

IMS Registration and Authentication

- **support of IMPI**
- authentication algorithm AKAv1
- subscription to the “reg” event package
- Security Agreement mechanism (no IPSec though)

IMS Session Establishment

- Precondition Mechanism
- Early Media
- Call transfer

Other IMS features:

- Setup wizard
- Voice and Video calls
- Dial history
- Contact list
- Instant Messaging
- Presence support

The IMS Communicator runs on Windows XP OS and on Linux OS. The development status of the project is 4-Beta. IMS-Communicator is an open-source project, licensed with the Apache Software License and the GNU Lesser General Public License (LGPL).

4.4 Selected PICO technologies

In the previous chapters several technologies have been analyzed and discussed. Below there is a brief description of the final technologies used in PICO.

4.4.1 PICO Server (PSCDS)

PICO server is a JAVA EE application. It uses PostgreSQL as database to collect all information related to the PSCD Users and Emergencies for a specific area. As Rule engine, PICO uses Drools from JBOSS. It takes some rules as input and provides the best action to do for each user, for example, a relevant application in case of emergency or an audio/video call to other PSCD User etc.

As IMS framework (server side), PICO uses doubango which is a 3GPP IMS framework for both embedded and desktop system. It is written in ANSI-C and so is very powerful. It exposes a Java wrapper to allow communication with the PICO JAVA EE application.

4.4.2 PICO Client (PSCD)

PICO client is the mobile application. It is written in JAVA and converted by Dalvik machine for Android system. As IMS framework (client side), PICO uses imsdroid which in turn is based on doubango as well. PICO extends its capabilities for full application support.

4.4.3 LoST: A protocol for mapping Geographic locations to public safety answering points

PICO architecture can take advantage also using LoST protocol for the answering point management. LoST has been designed to serve as a mapping protocol for handing PSAPs (Public Safety Answering Points). It allows end systems and VoIP proxies to map location data into URLs representing either PSAPs or other SIP proxies that perform a more fine-grained mapping. LoST is designed to operate globally, with a highly-distributed authority.

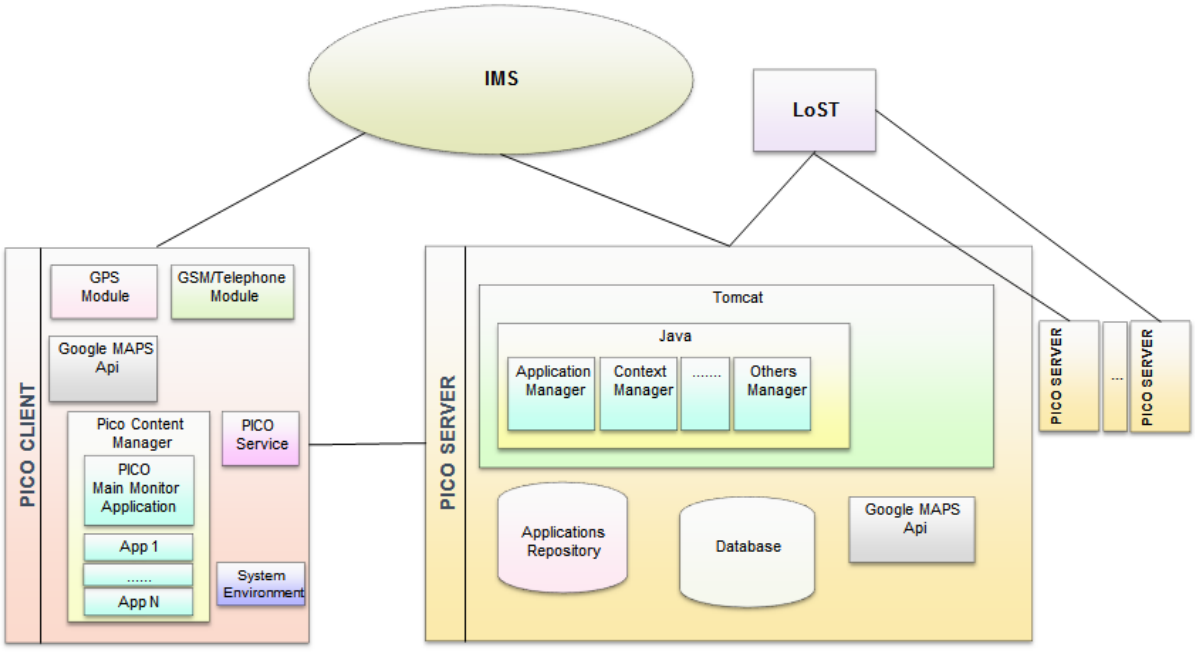

Figure 245: Lost: Draft Architecture

LoST is integrated into other components of an IP-based communications architecture for emergency calls. We are generally assuming that emergency calls use SIP, and in our case, use PICO Server, for setting up and terminating calls, as this is the most widely-used standards-based VoIP protocol.

The objective is to use LoST not only to serve calls but also to obtain the nearest PICO Server available for a PSCD user for instance to offer a relevant application to the context.

Table of Acronyms

AJAX	Asynchronous JavaScript and XML
CLDC	Connected Limited Device Configuration
CSS	Cascading Style Sheets
DHCP	Dynamic Host Configuration Protocol
DHTML	Dynamic HyperText Markup Language
DOM	Disk On Module
EMS	Emergency Medical Services
EOC	Emergency Operations Center
FF	Fire Fighters
GPL	General Public License
GUI	Graphic User Interface
HTML	HyperText Markup Language
IA	Intel Architecture
ICA	Independent Computing Architecture
IMS	IP Multimedia Subsystem
LE	Law Enforcement
J2ME	Java 2 Platform Micro Edition
JSON	JavaScript Object Notation
LAMP	Linux Apache MySQL Perl Python PHP
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MySQL	Multi-user Structured Query Language
NFS	Network File System
OpenVZ	Operating system-level VirtualiZation
PSCD	Public Safety Communications Device
PSCS	Public Safety Communication Server
PXE	Preboot eXecution Environment
RDC	Remote Desktop Connection
RDP	Remote Desktop Protocol

RFB	Remote Frame Buffer
RIA	Rich Internet Application
SAN	Storage Area Network
TFTP	Trivial File Transfer Protocol
VMM	Virtual Machine Monitor
VMWare	Virtual Machine softWare
VNC	Virtual Network Computing
WAN	Wild Area Network
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Index of Figures

5 Bibliography

[1] CEFRIEL, DELIVERABLE 1.1 “Application Scenarios and User-Provider Requirements”, PICO project

[2] CEFRIEL, DELIVERABLE 1.2 “Enabling Technologies”, PICO project

[3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, “SIP: Session Initiation Protocol”, RFC 3261, June 2002

[4] Roach, A., “Session Initiation Protocol (SIP) – Specific Event Notification”, RFC3265, June 2002

[5] A. Niemi, “SIP Extension for Event State Publication “,RFC 3903, October 2004

[6] Rosenberg, J., “A Presence Event Package for the Session Initiation Protocol (SIP)”, RFC 3856, August 2004.

[7] Day, M., Rosenberg, J., and H. Sugano, “A Model for Presence and Instant Messaging”, RFC 2778, February 2000.

[8] M. Lonnfors, E. Leppanen, H. Khartabil, J. Urpalainen, “Presence Information Data Format (PIDF) Extension for Partial Presence”, RFC 5262, September 2008

[9] Berners-Lee, T., Fielding, R. and L. Masinter, “Uniform Resource Identifiers (URI): Generic Syntax”, RFC 2396, August 1998

[10] J. Rosenberg, “A Data Model for Presence”, RFC 4479, July 2006

[11] H. Schulzrinne, “Timed Presence Extensions to the Presence Information Data Format (PIDF) to Indicate Status Information for Past and Future Time Intervals”, RFC4481, July 2006

[12] http://www.ietf.org/html.charters/simple-charter.html

[13] http://code.google.com/android/

[14] www.appstream.com

[15] www.endeavors.com

[16] http://ieeexplore.ieee.org

[17] http://portal.acm.org

[18] http://springerlink.metapress.com

[19] www.microsoft.com/systemcenter/softgrid/default.mspx

[20] http://openthinclient.org/home

[21] http://www.realvnc.com/index.html

[22] http://www.citrix.com/lang/English/home.asp

[23] www.vmware.com

[24] http://www.xen.org/

[25] http://openvz.org/

[26] http://www.adobe.com/products/flash/

[27] http://silverlight.net/

[28] http://www.sun.com/software/javafx/index.jsp