IST / Specific Targeted Research Project (STREP)

IST-2004-507217

IT2.3 Notification Service

Author(s): **Maximilian Schmidt**

Company(s): FOKUS

Version: Draft – V.01

Date: 18.02.05

Distribution: WP4 – D4.1

Code: IT2.3

Abstract:

This document describes the specification of a notification service for the eMayor platform.

The consortium consists of:

Partic. Role*	**Partic. no.**	**Participant name**	**Participant short name**	**Country**	**Date enter project****	**Date exit project****
CO	1	Deloitte & Touche	D&T	The Netherlands	1	26
CR	2	Expertnet S.A.	Expertnet	Greece	1	26
CR	3	University of Piraeus Research Centre	UPRC	Greece	1	26
CR	4	Fraunhofer Institute for open communication systems	FOKUS	Germany	1	26
CR	5	Ubizen N.V.	Ubizen	Belgium	1	26
CR	6	Advanced Encryption Technology Europe B.V.	AET	The Netherlands	1	26
CR	7	University of Zurich	UoZurich	Switcher land	1	26
CR	8	University of Siegen, Institute of Data Communication Systems	UoSiegen	Germany	1	26
CR	9	Municipality of Seville, Treasury and taxes delegation	MoSeville	Spain	1	26
CR	10	Municipality of Aachen	MoAachen	Germany	1	26
CR	11	Municipality of Siena	MoSiena	Italy	1	26
CR	12	Municipality of Bolzano	MoBolzano	Italy	1	26
CR	13	SADESI	SADESI	Spain	1	26
CR	14	Psychiko Municipal Development Corporation	MoPsychiko	Greece	1	26

Document Description

DOCUMENT REVISION HISTORY

Version	*Date*	*Modifications introduced*	
		Modification reason	*Modified by*
V.01		First draft	M. Schmidt

(to be removed in the version for submission)

Table of Contents

1 Introduction

This document contains the first draft presentation of the guidelines for using the implementation of the notification service for the eMayor platform.

2 eMayor Platform

As stated in D3.1 the eMayor platform will contain Notification and Printing Services controlled by a central service handling component (SH, see Figure 1).

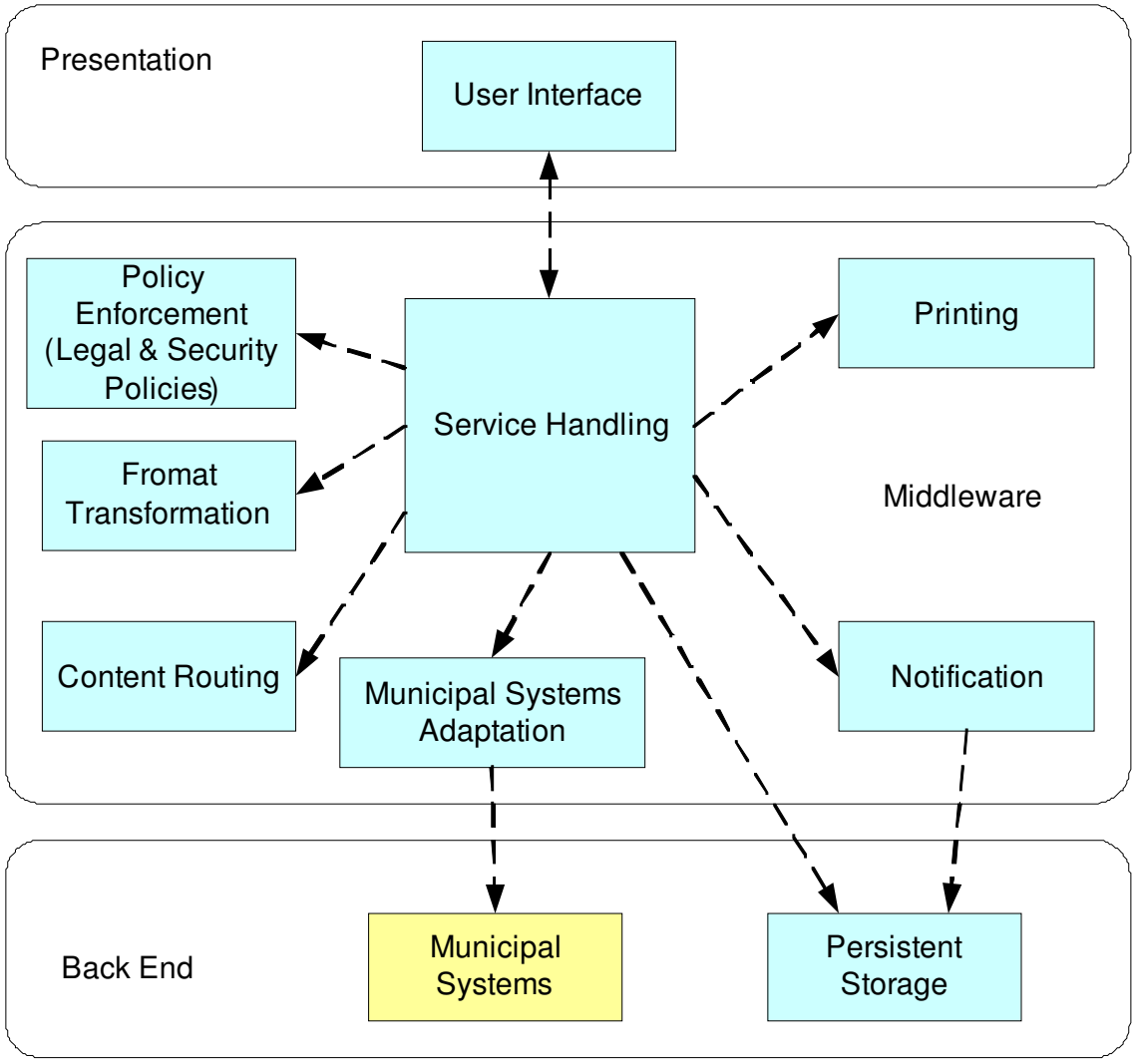

Figure 1: Top-level view of the eMayor system as packages of certain groups of functionalities

3 Notification Service

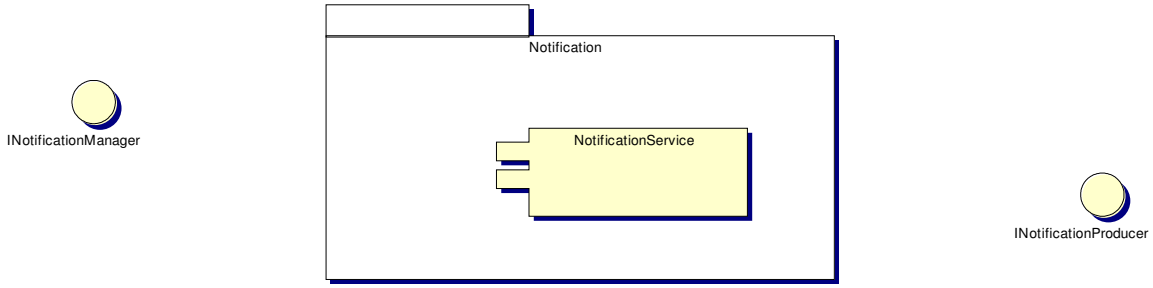

Figure 2: Interfaces of the Notification Service

The Notification Service as defined in D3.1 is integrated in the eMayor platform to provide the user with notification messages at different points of an ongoing process. To provide the user with such notifications a desired notification channel (called producer) including a valid configuration has to be implemented (eMail,Voice,SMS). To manage multiple notification producers, there also has to be a manager for storing and maintaining issues. Apart from session handling issues the notification manager also can provide a persistant storage mechanism for the producers by itself.
So we can divide the notification service in three main components:

- NotificationProducer - providing a notification channel for messages
- NotificationManager - managing multiple notification producers
- ProducerRepository - a persistant storage for notification producers

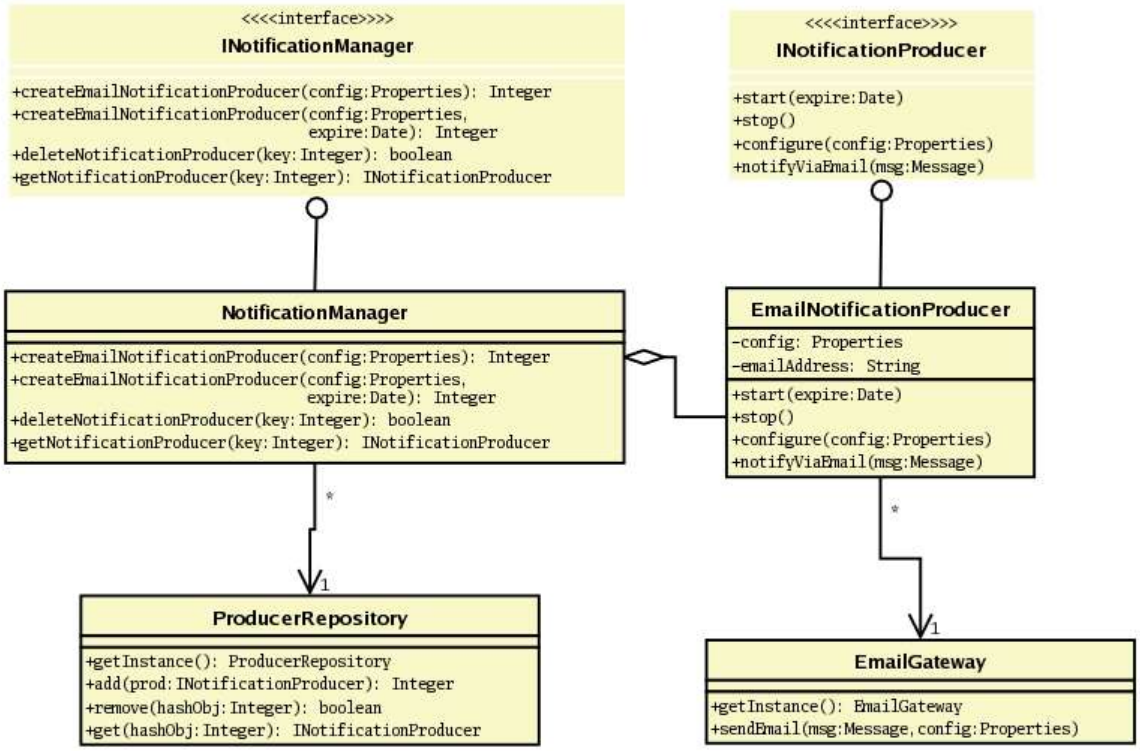

Figure 3: Breakdown of the Notification Service

3.1 INotificationProducer (interface)

The notification producer supports the following basic actions:

- ***void configure(java.util.Properties config)***
  throws java.rmi.RemoteException, NotificationException
  a notification producer can be configured using properties
- ***void start(java.util.Date expire)***
  throws java.rmi.RemoteException, NotificationException
  a notification producer can be started with a given date it should expire on
- ***void stop()***
  throws java.rmi.RemoteException, NotificationException
  a notification producer can be stopped
- ***void notifyViaMail(javax.mail.Message msg)***
  throws java.rmi.RemoteException, NotificationException
  there will be one notification method for each notification producer – up to now, only mail is provided using a message that is ready to be send out

See section “Example” below to get an idea how an actual implementation of an eMail producer can look like.
Here is a simple chart on the lifecycle of a notification producer:

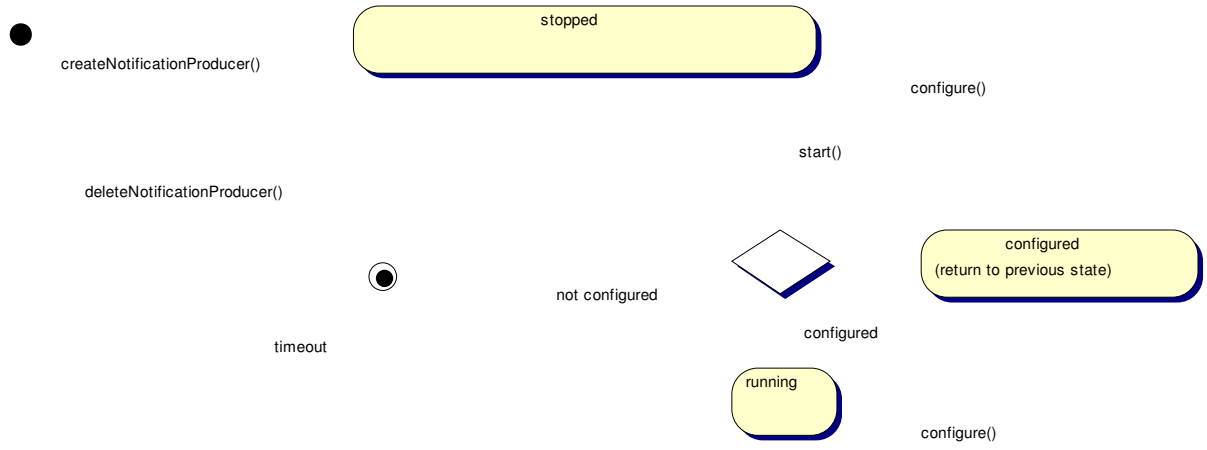

Figure 4: Lifecycle of a Notification Producer

3.2 INotificationManager (interface)

The notification manager acts as an interface to the outside and provides the following actions:

- ***Integer createEmailNotificationProducer(java.util.Properties config)***
  throws java.rmi.RemoteException, NotificationException
  - ➔ creates a producer for email-notifications, that expires after a default time of one day and will be configured usind the given properties
  - ➔ the returned integer (which is actually the hashCode of the producer) has to be stored by the client, so it can be used as a back-reference for access to that producer
  - ➔ the producer will be stored in an persistant repository, which will be available the next time a manager is created
- ***Integer createEmailNotificationProducer(java.util.Properties config, java.util.Date expire)***
  throws java.rmi.RemoteException, NotificationException
  same as above, but expires at a given time
- ***boolean deleteNotificationProducer(Integer key)***
  throws java.rmi.RemoteException, NotificationException
  will delete the producer referenced by the given key
- ***INotificationProducer getNotificationProducer(Integer key)***
  throws java.rmi.RemoteException
  return the producer referenced by the given key

3.4 Example (EmailNotificationProducer)

As default an eMail notification producer is avaiable.

3.3.1 Implementation

The client should be able to make transparent use of the producer.
Therefore every producer has to implement the INotificationProducer interface, which for EJB issues extends javax.ejb.EJBObject. This is necessary because the remote interface of the implemented EJB has to extend INotificationProducer, therefore allowing the client to use the notification methods.
(This concept if far from perfect, because now each producer has to implement dummy methods for all notification methods used by every other producer that will ever be implemented. In fact we only need this for casting issues to present a clean interface to the client. An abstract class won´t do the job, because it has to be extended by the remote interface of the producer´s EJB.)
As specified in D3.1 we also provide an integrated EmailGateway that will actually do the notification by sending an eMail out using the given properties.

Here you can see a list of properties that can (and in some cases should) be provided to the notification producer:

Property	***Description***
mail.smtp.host	the host used for sending mails (required)
mail.smtp.auth	whether authentication for sending is necessary (optional)
mail.smtp.user	name of the user for authentication on the mailhost (optional, requires auth)
mail.smtp.pass	password for authentication (optional, requires user)
emailAddress	recipient address for the producer (required, if not provided NotificationException is thrown)

3.3.2 Usage

Here is a small example showing how to use the NotificationManager along with the standard EmailNotificationProducer.

First of all, look up and create a new NotificationManager:

```
/* get context & lookup for service */
Context context = new InitialContext();
Object ref = context.lookup("ejb/NotificationManager");
NotificationManagerHome home =
        (NotificationManagerHome) javax.rmi.PortableRemoteObject.narrow
```

```
                    (ref,NotificationManagerHome.class);

    /* set up the manager */
    INotificationManager manager = home.create();
```

Then, create a message that should be send as notification:

```
    /* set up properties */
    String host  = “localhost”;
    String emailAddress = “root@”+host;

    /* set properties according to local mail-system settings */
    Properties prop = System.getProperties();
    prop.put("mail.smtp.host",host);

    /* get a new session for the message */
    Session sess = Session.getDefaultInstance(prop,null);


    /* create a new message out of the values read */
    Message msg = new MimeMessage(sess);
    try {
            msg.setFrom(new InternetAddress(emailAddress));
            msg.setSubject(“subject”);
            msg.setText(“bodypart”);
    } catch (Exception e) {}
```

Now, create a producer using the previously defined properties and store the reference:

```
    Integer key = manager.createEmailNotificationProducer(prop);
```

Push a notification using the producers notify-method:

```
    (manager.getNotificationProducer(key)).notifyViaMail(msg);
```

Delete the producer using its reference-key:

```
    manager.deleteNotificationProducer(key);
```

Finally, remove the manager:

```
    manager.remove();
```

3.3.2 Sequence chart

Here you can see a sequence chart including details for the actions taken in the example above.