Implémentation broker SMSU 1.x

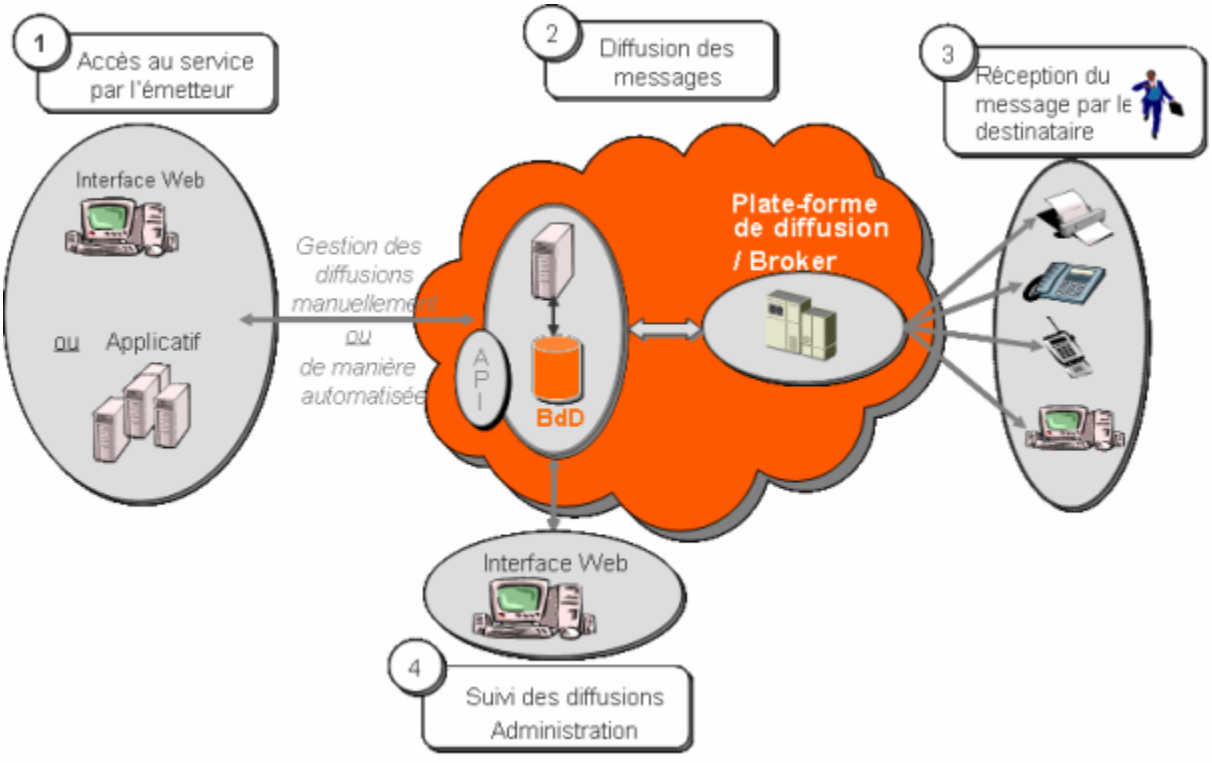
Documentation pour l'ancienne version de SMS-U

Contexte du projet

Rappel du contexte

L'Université Paris 1 Panthéon - Sorbonne souhaite mettre en place un service d'envoi de messages SMS intégré à l'Environnement Numérique de Travail. Ce service doit être :

- Ouvert aux applications du SI (bibliothèque, service administratif, partenaires, …)
- Mutualisable entre les différents adhérents de l'UNPIdF



Ce service doit permettre d'attribuer à l'université une image d'administration en mouvement, ayant réussi son passage à l'ère numérique, en utilisant un média de communication très apprécié des jeunes adultes.

Ce service doit respecter le cadre technique décrit dans le projet open source ESUP Portail en s'appuyant plus particulièrement sur le framework esup-commons et proposer une architecture d'API permettant un faible couplage vis-à-vis des opérateurs télécom.

Périmètre de l'étude

Ce document décrit la méthode à utiliser pour implémenter un broker autre que celui fourni par OLM.

Architecture de l'implémentation broker

Fichiers concernés dans smsuapi

- l'implémentation doit être réalisée dans

  ```
  src/org/esupportail/smsuapi/services/sms/impl/le_nom_du_broker/
  ```

- la configuration doit être mise dans :

  ```
  properties/broker/le_nom_du_broker/
  ```

  et notamment

  ```
  properties/broker/le_nom_du_broker/le_nom_du_broker.xml
  ```

  qui doit contenir un bean id="smsSenderImpl".
  Regarder properties/broker/olm/olm.xml et properties/proxy/proxy.xml pour avoir des exemples.
- enfin, pour utiliser le nouveau broker, il faut modifier properties/config.properties :

  ```
  sms.connector.name=le_nom_du_broker
  ```

Description de la gestion de l'envoi de SMS

Toute implémentation de broker doit répondre à l'interface ISMSSender.java. Celle ci ne contient qu'une méthode :

```
public interface ISMSSender {
    /**
     * Send the specified message.
     * @param smsMessage
     */
    void sendMessage(SMSBroker smsMessage);
}
```

Cette méthode *sendMessage* est appelée par la couche métier du Back Office du service SMS-U. Elle ne prend qu'un unique paramètre, un objet de type SMSBroker qui est décrit comme suit :

```
public class SMSBroker implements Serializable {
    /**
     * The unique identifier message.
     */
    private int id;
    /**
     * The message recipient.
     */
    private String recipient;
    /**
     * The message itself.
     */
    private String message;
    ...
}
```

Ce « pojo » est composé de 3 attributs (et de leurs accesseurs) décrivant un SMS à envoyer avec :

- id : l'identifiant unique du SMS à envoyer
- recipient : une chaine de caractère représentant le numéro de téléphone du destinataires
- message : le message en lui même.

Description de la gestion des acquittements

La gestion des acquittements reçus par le broker passe la classe nommée AckManager.

```
public class AckManager {
    /**
     *
     * @param acknowledgment
     */
    public void manageAck(final Acknowledgment acknowledgment) {
        final AckManagerThread ackManagerThread = new AckManagerThread();
        ackManagerThread.setAcknowledgment(acknowledgment);
        ackManagerThread.run();
    }
}
```

Elle gère les acquittements sous la forme d'un objet Acknowledgment.

```
public class Acknowledgment {
    /**
     * The sms unique identifier.
     */
    private int smsId;
    /**
     * the sms ack status.
     */
    private SmsStatus smsStatus;
    public int getSmsId() {
        return smsId;
    }
    public void setSmsId(final int smsId) {
        this.smsId = smsId;
    }
    public SmsStatus getSmsStatus() {
        return smsStatus;
    }
    public void setSmsStatus(final SmsStatus ackStatus) {
        this.smsStatus = ackStatus;
    }
}
```

Ce « pojo » contient deux attributs :

- smsId : qui est l'identifiant unique du message auquel est associé l'acquittement (le pendant de l'attribut id de la classe SMSBroker).
- smsStatus : qui est la valeur de l'acquittement, cette valeur représente l'état dans lequel est le sms

La classe SmsStatus est une énumération qui reprend les différents états que peut prendre le sms.

```
public enum SmsStatus {
    // Message saved in BD
    CREATED,
    // Message sent by the broker
    IN_PROGRESS,
    // Message successfully sent to the user mobile phone
    DELIVERED,
    // Message not sent due to quota error
    ERROR_QUOTA,
    // Message not sent because phone number is already in black list
    ERROR_PRE_BL,
    // Message not sent because phone number is invalid
    ERROR_POST_BL,
    // other error
    ERROR
}
```

Pour remplacer l'actuelle implémentation, il faut donc implémenter un gestionnaire d'acquittement spécifique au broker (comme cela est fait pour OLM dans le package org.esupportail.smsuapi.services.sms.impl.olm) dans lequel sera construit un objet Acknowledgment qui sera envoyé à la couche métier par le biais de la classe AckManager.

NB : le code OlmAckManager utilise notamment fr.cvf.util.mgs.mode.sgs.Manager qui gère lui-même un cron qui interroge régulièrement le broker pour savoir s'il y a des acks. A défaut, on peut faire comme le broker "proxy" : utiliser quartz (cf id="smsuapiStatusJobTrigger" dans properties/broker/proxy /proxy.xml).

NB2 : le broker "proxy" ayant des besoins spécifiques n'utilise pas AckManager. L'équivalent à AckManagerBusiness.manageAck est AckStatusProxy. updateStatus

Glossaire

Glossaire des manuels du service SMS-U