Capítulo 1

Introducción

En los últimos años, se ha presentado una enorme demanda por servicios portátiles, a los que se les ha llamado tecnologías móviles, este repentino crecimiento de tecnologías ha traído consigo nuevas formas de hacer negocios y nuevos paradigmas para ofrecer servicios de información. Tal es el caso de los servicios de texto o *SMS* [1] [4] por sus siglas en inglés *Short Message Service* que permiten el intercambio de mensajes cortos (entre 140 y 160 caracteres como límite) entre teléfonos celulares o entre teléfonos celulares y una central de *SMS* (llamada *SMSC* [1] [4] por sus siglas en inglés *Short Message Service Center*) que a su vez, puede redirigir los mensajes a una entidad externa de mensajes cortos (llamada *ESME* [4] por sus siglas en inglés *External Short Message Entity*) donde pueden ser procesados para brindar servicios más complejos.

Para que un *ESME* sea capaz de hospedar servicios, éste debe comunicarse con el *SMSC* mediante un protocolo de propósito específico, generalmente *SMPP* [4] que permite el intercambio de mensajes a través de una conexión punto a punto, que puede realizarse usando protocolos de más bajo nivel como TCP [3], PPP [8] u otros. Este enlace debe ser establecido por un gateway (*SMS gateway*) que implemente dicho protocolo e integre el enlace a los servicios de información que se van a ofrecer. Los servicios son atendidos por un software que recibe el mensaje del cliente, lo procesa y envía una

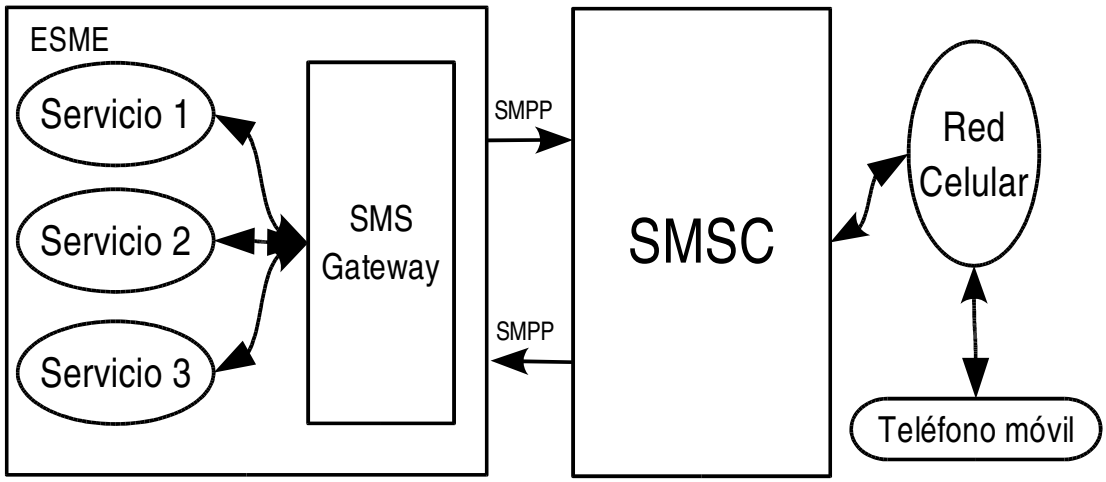

Figura 1.1: Trayectoria de los mensajes transmitidos entre un servicio y un teléfono móvil

respuesta de regreso en caso de ser requerido. A este software vamos a llamarlo *servicio* de ahora en adelante.

Uno de los *SMS* gateways más populares actualmente, es el proyecto *Kannel* [2] [9], que es un *WAP* [7] y *SMS* gateway que sigue la filosofía *OpenSource* [10]. *Kannel* posee, entre otras características, la capacidad para crear servicios muy sencillos mediante el uso de archivos de configuración, pero para una aplicación en el mundo real, esta capacidad es bastante reducida, por lo que posee características que le permiten comunicarse con otros procesos usando HTTP [6], de este modo, servicios más complejos pueden ser desarrollados como aplicaciones web.

1.1. Definición del problema

El protocolo que usa *Kannel* para comunicarse con otras aplicaciones (*HTTP*) funciona correctamente, pero tiene la desventaja de no haber sido diseñado específicamente

para este propósito, lo cual trae con sigo algunos aspectos a considerar.

1.1.1. Persistencia y fallas

HTTP no incorpora características para mantenimiento de persistencia, por lo que en caso de falla del sistema, pueden llegar a perderse mensajes, ya que no existe un método para recuperarlos. Tampoco existe el concepto de transacciones por lo que no se tiene un control de las acciones que son iniciadas y terminadas con éxito.

1.1.2. Componentes

Kannel puede ser configurado para funcionar de manera distribuida, pero sus componentes realizan demasiadas tareas que en la mayoría de los casos no son utilizadas. El funcionamiento de unos componentes depende del de otros de manera directa, por lo que podemos decir que los módulos de Kannel no son del todo autónomos.

1.2. Objetivos

1.2.1. Objetivos generales

Crear un módulo para *Kannel* (*box*) que sirva como gateway entre el protocolo nativo/intenrno de *Kannel* y Java mediante el uso de *JMS* que permita persistencia de los mensajes y uso de transacciones.

1.2.2. Objetivos específicos

1. Habilitar la persistencia de mensajes transferidos entre Kannel y un servicio SMS mediante la integración con JMS que permite administrar esta característica.

2. Habilitar las características de JMS para uso de transacciones en las colas de mensajes compartidas por el gateway y una aplicación de servicio SMS.

3. Reducir el tamaño de los componentes de un Gateway basado en Kannel.

4. Aumentar la autonomía de la administración de los componentes.

5. Determinar si es necesario modificar el *bearerbox* para una conexión más estable.

1.3. Alcances y limitaciones

1.3.1. Alcances

1. Desarrollar un módulo (*box*) capaz de mantener una conexión con el *bearerbox* de *Kannel*.

2. El software resultante tendrá las características necesarias para la etapa de pruebas.

3. Crear una arquitectura que permita mayor autonomía de los componentes del sistema.

4. Realizar pruebas con el nuevo módulo:

   a) Que demuestren la funcionalidad básica de la implementación del protocolo.

   b) Con simulaciones de casos de tráfico extremo.

   c) Con un servicio de ejemplo.

1.3.2. Limitaciones

1. La manera de configurar el software resultante será principalmente mediante archivos de configuración.
2. El manual resultante tendrá sólo las características necesarias para que otras personas de la comunidad puedan experimentar con él.
3. La documentación del software deberá estar escrita en inglés con el objeto de facilitar la participación de la comunidad en el proyecto.
4. Las pruebas se limitarán al hardware, software y conexiones disponibles al momento de la experimentación.

1.4. Recursos a utilizar

Hardware

Los requerimientos de hardware dependerán de los requerimientos de las herramientas de software a utilizar.

Software

Las pruebas se realizarán usando la versión CVS de *Kannel* más reciente disponible y compatible al momento de realizar las pruebas. Se utilizará cualquier sistema operativo que soporte el uso de las herramientas requeridas:

1. Java como plataforma y lenguaje de programación principal
2. jEdit como interfaz de desarrollo
3. gcc Kannel está escrito en C asi que es probable llegar a utilizar algunas librerías o desarrollar partes en C.

4. ethereal/tcpdump software de monitoreo de bajo nivel para analizar la información transferida.

5. Kannel es preferente trabajar con la última versión cvs de Kannel.

6. JBoss para usar su implementación de JMS y otras características.

1.5. Estado actual del problema

1.5.1. Sistemas existentes

No hay sistemas disponibles en la red que resuelvan estos problemas en Kannel, lo más cercano a éste, son sistemas que reciben los mensajes de Kannel por HTTP y a continuación los traducen a una especificación de JMS para distribuirlos a sus servicios. Esto es, usando el *smsbox* incluido en Kannel que presenta las desventajas mostradas en la sección 1.1.

Componentes de Kannel

Los componentes en Kannel son llamados box´es en el lenguaje del proyecto, y existen tres tipos principales:

- bearerbox
- smsbox
- wapbox

El bearerbox o bearer es el encargado de realizar las conexiones con los SMSC's y realizar el ruteo entre las conexiones y los servicios, el smsbox se conecta al bearer mediante un protocolo interno y es el encargado de atender los servicios SMS, y el wapbox

se encarga del mismo modo de atender servicios WAP. De los problemas que se han expuesto en la sección 1.1, el único que ha intentado resolverse ha sido el de la persistencia, pero su solución hasta el momento no ha mostrado buenos resultados, actualmente (Octubre 2004) es la parte más discutida y modificada diariamente por los desarrolladores interesados. Para intentar resolver este problema, el bearerbox implementa una cola de mensajes interna que almacena los mensajes de tres modos posibles:

- Memoria RAM - Veloz pero no es persistente.
- Archivo - Persistente pero muy lento.
- Base de Datos - Promedio de los anteriores.

Estos métodos, en condiciones de alto tráfico o tráfico con ciertas características, suele fallar, resultando en la pérdida de algunos mensajes.

1.5.2. Aportación

La idea aquí es la implementación de un smsbox que en lugar de proveer sus servicios a través de HTTP, lo haga a través de JMS. Podríamos decir que se va a implementar un *jsmsbox* que tomará en cuenta los aspectos negativos del smsbox (Secc. 1.1) para corregirlos.

1.5.3. Técnicas y tecnologías a usar

- Reverse Engineering, para descifrar el protocolo nativo de Kannel
- Packet Sniffing, para descifrar el protocolo nativo de Kannel principalmente.
- J2EE, se usará como ambiente en algunos casos y pruebas para demostrar integración.

- JMS, especificación de MQ de Java.

- Los lenguajes que serán más probablemente utilizados son: Java, C, Bourne Shell.