Capítulo 5

Pruebas

En este capítulo se analizará la correctitud de implementación y funcionamiento de algunos aspectos del software. Para tener un ambiente de pruebas apropiado, se debe tener en primer lugar un simulador de SMSC, se utilizó un simulador implementado en Java distribuido OpenSource por *Logica Mobile Networks Limited* llamado *OpenSMPP* (`http://opensmpp.logica.com`):

Simulador de SMSC

```
[tesis@mathilda simulator_1]$ java -cp ./smpp.jar:./smscsim.jar
   com.logica.smscsim.Simulator
Copyright (c) 1996-2001 Logica Mobile Networks Limited
This product includes software developed by Logica by whom
   copyright
and know-how are retained, all rights reserved.


- 1 start simulation
- 2 stop simulation
- 3 list clients
- 4 send message
- 5 list messages
- 6 reload users file
- 7 log to screen off
- 0 exit
```

```
> 1
Enter port number> 10001
Starting listener... started.

- 1 start simulation
- 2 stop simulation
- 3 list clients
- 4 send message
- 5 list messages
- 6 reload users file
- 7 log to screen off
- 0 exit
>
```

Como puede verse en el listado, éste ha sido iniciado en el puerto 10001, que es el puerto al que espera conectarse kannel. A continuación debe iniciarse el bearerbox de kannel:

bearerbox de Kannel

```
[tesis@mathilda Kannel]$ bin/bearerbox conf/test_00.conf
 ... [Debug info] ...
Hostname mathilda.micel.com, IP 127.0.0.1.
Libxml version 2.5.11.
Using OpenSSL 0.9.7a Feb 19 2003.
Using native malloc.
 ... [Debug info] ...
2005-05-02 07:20:12 [9764] [0] INFO:
   ----------------------------------------
2005-05-02 07:20:12 [9764] [0] INFO: Kannel bearerbox II version
    1.4.0 starting
2005-05-02 07:20:12 [9764] [0] INFO: MAIN: Start-up done,
   entering mainloop
2005-05-02 07:20:12 [9764] [7] DEBUG: sms_router: list_len = 0
```

El archivo de configuración usado para estas pruebas, se encuentra en el apéndice D.

Es en este momento cuando una conexión del bearerbox con el simulador es establecida:

Simulador de SMSC

```
Enter port number> 10001
Starting listener... started.

- 1 start simulation
- 2 stop simulation
- 3 list clients
- 4 send message
- 5 list messages
- 6 reload users file
- 7 log to screen off
- 0 exit
> 07:20:12 [sys] new connection accepted
07:20:12 [] client request: (bindreq: (pdu: 36 9 0 1) pavel
   pavel VMA 82 (addrrang: 0 0 ) )
07:20:12 [pavel] authenticated pavel
07:20:12 [pavel] server response: (bindresp: (pdu: 0 80000009 0
   1) Smsc Simulator)
```

Seguido de esto, el sniffer debe ser iniciado, para estas pruebas se utilizó *Ethereal - Network Protocol Analyzer v0.10.8* (`http://www.ethereal.com/`). Para nuestros fines, el análisis de los protocolos IRCd y SMPP deben ser deshabilitados en ethereal, debido a que por su similitud con este ambiente de pruebas, causarán distracción al software.

En último lugar, vamos a iniciar JBoss, que es un servidor de aplicaciones para J2EE. Estas pruebas fueron realizadas con las configuraciones por default de dicho ambiente:

JBoss

```
[tesis@mathilda jboss-4.0.1sp1]$ ./bin/run.sh
=========================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /home/tesis/jkjGW/JBoss/jboss-4.0.1sp1
```

```
  JAVA: /usr/java/j2sdk1.4.2_03/bin/java

  JAVA_OPTS: -server -Xms128m -Xmx128m -Dprogram.name=run.sh

  CLASSPATH: /home/tesis/jkjGW/JBoss/jboss-4.0.1sp1/bin/run.jar
     :/usr/java/j2sdk1.4.2_03/lib/tools.jar

=========================================================

07:57:00,152 INFO  [Server] Starting JBoss (MX MicroKernel)...
 ... [Startup info] ...
07:58:34,646 INFO  [JkMain] Jk running ID=0 time=1/213  config=
   null
07:58:34,823 INFO  [Server] JBoss (MX MicroKernel) [4.0.1sp1 (
   build: CVSTag=JBoss_4_0_1_SP1 date=200502160314)] Started in
   1m:31s:477ms
```

Este servidor de aplicaciones contiene al proveedor de JMS, *JBossMQ*, que ya está incluido en la configuración por default del mismo y que ofrecerá sus servicios a nuestra aplicación.

5.1. Funcionamiento del protocolo

Debido a que se describió el protocolo usando sniffers y el código fuente no documentado de Kannel, hubo que hacer pruebas en ambas implementaciones para comparar los resultados y ajustar satisfactoriamente esta versión.

El primer paso de validación de los paquetes durante la etapa de desarrollo fue hacer muestras hexadecimales de las cadenas de bytes generadas por cada tipo de paquete para asegurar que se ajustaran a la descripción del protocolo en el apéndice A.3.

kjGateway

```
[tesis@mathilda kjGateway]$ ./launchGW
Kannel - JMS Gateway v0.1
```

```
 by: Oscar Medina Duarte
 moscar[at]gmail.com
out:
0000000C 00000001  00000003 FFFFFFFF  | ................

AckThread starting
Reading thread started.
out:
00000008 00000000  00000000  | ............

in:
000000A7 00000002  00000006 32333432  |  ........ ....2342
33340000 00063233  34323334 FFFFFFFF  |  34....23 4234....
00000004 686F6C61  4276428B 00000007  |  ....hola BvB.....
74657374 5F3030FF  FFFFFFFF FFFFFF00  |  test_00. ........
00002439 35363430  6133632D 37656564  |  ..$95640 a3c-7eed
2D343130 362D3863  36382D65 36633836  |  -4106-8c 68-e6c86
66353935 34663100  000000FF FFFFFFFF  |  f5954f1. ........
FFFFFF00 00000000  000000FF FFFFFFFF  |  ........ ........
FFFFFFFF FFFFFFFF  FFFFFF00 00000000  |  ........ ........
000000FF FFFFFFFF  FFFFFFFF FFFFFFFF  |  ........ ........
FFFFFFFF FFFFFF00  000000  | ............

writing ack
ack written
out:
000000A8 00000002  00000006 32333432  |  ........ ....2342
33340000 00063233  34323334 FFFFFFFF  |  34....23 4234....
0000000C 486F6C61  206D756E 646F2021  |  ....Hola .mundo.!
4276428B FFFFFFFF  FFFFFFFF FFFFFFFF  |  BvB..... ........
00000024 31643062  63376661 2D626231  |  ...$1d0b c7fa-bb1
632D3131 64392D61  3564352D 33313565  |  c-11d9-a 5d5-315e
39343439 65626133  00000001 FFFFFFFF  |  9449eba3 ........
FFFFFFFF 00000000  00000000 FFFFFFFF  |  ........ ........
FFFFFFFF FFFFFFFF  FFFFFFFF 00000000  |  ........ ........
00000000 FFFFFFFF  FFFFFFFF FFFFFFFF  |  ........ ........
FFFFFFFF FFFFFFFF  00000000  | ............

in:
00000034 00000003  00000000 4276428B  |  ...4.... ....BvB.
00000024 31643062  63376661 2D626231  |  ...$1d0b c7fa-bb1
632D3131 64392D61  3564352D 33313565  |  c-11d9-a 5d5-315e
39343439 00000000  | 9449....
```

Disección del último paquete de salida:

Disección

```
1er cuarteto de bytes: longitud = 000000A8 = 168
2do cuarteto de bytes: tipo = 00000002 = SMS
```

Ya que sabemos que el paquete de longitud 168 bytes contiene un mensaje tipo SMS, comparamos el resto del mensaje con su especificación BNF:

Disección

```
<sms_packet> := <sender><receiver><udhdata><msgdata><time>\
                <smsc_id><service><account><uuid><sms_type>\
                <mclass><mwi><coding><compress><validity>\
                <deferred><dlr_mask><dlr_url><pid><alt_dcs>\
                <rpi><charset><boxc_id><binfo><msg_left><priority>
```

Disección

```
<sender> = 00000006 32333432 3334 = 234234
<receiver> = 00000006 323334323334 = 234234
<udhdata> = FFFFFFFF = cadena vacia
<msgdata> = 0000000C 486F6C61  206D756E 646F2021 = Hola mundo !
<time> = 4276428B = 69690408 secs
<smsc_id> = FFFFFFFF = cadena vacia
<service> = FFFFFFFF = cadena vacia
<account> = FFFFFFFF = cadena vacia
<uuid> = 00000024 31643062  63376661 2D626231
632D3131 64392D61  3564352D 33313565
39343439 65626133
<sms_type> = 00000001
<mclass> = FFFFFFFF = null
<mwi> = FFFFFFFF = null
<coding> = 00000000
<compress> = 00000000
<validity> = FFFFFFFF = null
```

```
<deferred> = FFFFFFFF = null
<dlr_mask> = FFFFFFFF = null
<dlr_url> =  FFFFFFFF = cadena vacia
<pid> = 00000000
<alt_dcs> = 00000000
<rpi> = FFFFFFFF = null
<charset> = FFFFFFFF = cadena vacia
<box_id> = FFFFFFFF = cadena vacia
<binfo> = FFFFFFFF = cadena vacia
<msg_left> = FFFFFFFF = null
<priority> = 00000000
```

Del mismo modo, fueron analizados todos los paquetes que genera el API.

El siguiente paso fue verificar la comunicación y el funcionamiento del protocolo usando un sniffer (Apéndice E).
Por ejemplo, la comparación del contenido de un mensaje de identificación generado por smsbox:

Disección

```
...
0040  ca 38 00 00 00 0c 00 00 00 01 00 00 00 03 ff ff   .8..............
0050  ff ff                                            ..
```

Comparado con uno creado por kjGateway:

Disección

```
 ...
0040  4a f2 00 00 00 0c 00 00 00 01 00 00 00 03 ff ff   J...............
0050  ff ff                                            ..
```

Nota: Los paquetes empiezan en la sección de datos del paquete TCP, es decir, el primer byte del paquete está en el offset 0x0042.
Podemos ver que ambos paquetes son idénticos. De hecho, la prueba más importante

en este caso, y que nos puede decir definitivamente que el protocolo se implementó correctamente, es el hecho de poder mantener conexiones entre el bearerbox y kjGateway sin problemas.

Una vez validado el protocolo, se realizaron pruebas de funcionalidad conectándose directamente al bearerbox, y comparando dichas sesiones con sesiones realizadas por un smsbox de kannel usando un sniffer.

Estas pruebas fueron las más tardadas dada la circunstancia que se explicó en la sección 4.1, ya que el sniffer no fue útil para encontrar y diagnosticar dicha situación.

5.2. Funcionamiento del gateway

Para probar el funcionamiento del gateway, se implementó un servicio de ejemplo que cuando recibe mensajes de texto cuyo contenido en mayúsculas es igual a un keyword de contenido en un archivo de propiedades, devuelve dicho contenido, por ejemplo, con un archivo de configuración:

```
PRUEBA = Prueba exitosa !
HOLA = Hola mundo !
ADIOS = Good bye !
```

Si un cliente móvil envía un mensaje a nuestro sistema, con contenido: hola, el servicio deberá responder con : Hola mundo !

Por ejemplo, enviamos un mensaje desde el simulador:

Simulador de SMSC

```
- 1 start simulation
- 2 stop simulation
- 3 list clients
```

```
- 4 send message
- 5 list messages
- 6 reload users file
- 7 log to screen off
- 8 Stress Test
- 0 exit
> 4
pavel
Type the message> hola
From> 222111
  to> 444
Message sent.
```

Enseguida podemos ver justo el momento en que el mensaje de respuesta es recibido y a continuación mostrar el mensaje de la lista de mensajes recibidos del simulador:

Simulador de SMSC

```
> 11:37:32 [pavel] client request: (submit: (pdu: 54 4 0 2) (
   addr: 0 1 444)  (addr: 2 1 222111)  (sm: msg: Hola mundo !)
   (opt: ) )
11:37:32 [pavel] putting message into message store
- 1 start simulation
- 2 stop simulation
- 3 list clients
- 4 send message
- 5 list messages
- 6 reload users file
- 7 log to screen off
- 8 Stress Test
- 0 exit
> 5
-------------------------------------------------------------
   -----------
| Msg Id   |Sender     |ServT|Source address |Dest address   |
   Message
-------------------------------------------------------------
   -----------
- Smsc2001 |pavel      |null |444            |222111         |
   Hola mundo !
```

En ese mismo momento, podemos verificar en la terminal de kjGateway que un

mensaje llegó y que otro fue enviado:

kjGateway

```
Kannel - JMS Gateway v0.1
 by: Oscar Medina Duarte
 moscar[at]gmail.com
 ...
AckThread starting
Reading thread started.
 ...
in:
000000A4 00000002  00000006 32323231  |  ........ ....2221
31310000 00033434  34FFFFFF FF000000  |  11....44 4.......
04686F6C 61427657  4B000000 07746573  |  .holaBvW K....tes
745F3030 FFFFFFFF  FFFFFFFF 00000024  |  t_00.... .......$
37633561 34366266  2D346131 352D3437  |  7c5a46bf -4a15-47
33342D38 3563632D  38613037 37666533  |  34-85cc- 8a077fe3
64663339 00000000  FFFFFFFF FFFFFFFF  |  df39.... ........
00000000 00000000  FFFFFFFF FFFFFFFF  |  ........ ........
FFFFFFFF FFFFFFFF  00000000 00000000  |  ........ ........
FFFFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF  |  ........ ........
FFFFFFFF 00000000  | ........

writing ack
ack written
out:
000000A5 00000002  00000003 34343400  |  ........ ....444.
00000632 32323131  31FFFFFF FF000000  |  ...22211 1.......
0C486F6C 61206D75  6E646F20 21427657  |  .Hola.mu ndo.!BvW
4CFFFFFF FFFFFFFF  FFFFFFFF FF000000  |  L....... ........
24376233 65613831  352D6262 32382D31  |  $7b3ea81 5-bb28-1
3164392D 39386262  2D373937 33363336  |  1d9-98bb -7973636
61396430 32000000  01FFFFFF FFFFFFFF  |  a9d02... ........
FF000000 00000000  00FFFFFF FFFFFFFF  |  ........ ........
FFFFFFFF FFFFFFFF  FF000000 00000000  |  ........ ........
00FFFFFF FFFFFFFF  FFFFFFFF FFFFFFFF  |  ........ ........
FFFFFFFF FF000000  00  | .........

in:
00000034 00000003  00000000 4276574C  |  ...4.... ....BvWL
00000024 37623365  61383135 2D626232  |  ...$7b3e a815-bb2
382D3131 64392D39  3862622D 37393733  |  8-11d9-9 8bb-7973
36333661 00000000  | 636a....
```

Y del mismo modo, la terminal que muestra la salida del servicio de ejemplo, refleja la llegada de dicho mensaje y su respuesta:

SimpleService

```
[tesis@mathilda kjGateway]$ ./launchSimpleService
Starting SimpleService
-- listing properties --
ADIOS=Good bye !
PRUEBA=Prueba exitosa !
HOLA=Hola mundo !
Para terminar presiona *  ENTER
Contenido recibido: HOLA
Enviando contenido: Hola mundo !
```

Este servicio minimalista ha sido probado con éxito, lo que nos indica que el gateway funciona de correctamente.

5.3. Transacciones y persistencia

Estos aspectos no son controlados de manera explícita por la aplicación, sino que son administrados por el proveedor de JMS, quien típicamente usa una base de datos para almacenar los mensajes encolados.

El uso de transacciones en JMS debe ser solicitado al momento de crear la sesión de JMS:

.

```
...
QueueConnectionFactory queueConnectionFactory = null;
QueueConnection queueConnection = null;
QueueSession queueSession = null;
```

```
...
queueConnectionFactory = (QueueConnectionFactory)jndiContext
                                  .lookup("UIL2ConnectionFactory"));
queueConnection =
   queueConnectionFactory.createQueueConnection();
queueSession =
 queueConnection.createQueueSession(true,Session.AUTO_ACKNOWLEDGE);
//                                   ^
//                                   |
// Transacciones activadas = true >-+
...
```

Este tipo de sesiones usan un modelo de transacciones encadenadas. En dicho modelo, la aplicación no inicia explícitamente una transacción, ya que al iniciar una sesion, se inicia automaticamente la primera transacción. Al llamar a los métodos `commit` o `rollback`, la aplicación inicia automáticamente una transacción. Se puede decir entonces, que siempre hay una transacción presente y disponible.

La persistencia también es administrada por el proveedor de JMS, una prueba que podemos hacer para mostrar su funcionamiento, es enviar varios mensajes al gateway cuando el servicio de ejemplo no esté corriendo y a continuación ejecutar el servicio. Al realizar esta prueba, el sistema se comportó del modo esperado, es decir que cuando el servicio fue iniciado, pudo recuperar los mensajes enviados a él sin problemas.
La siguiente prueba que se realizó fue enviar mensajes estando el servicio desactivado, a continuación reiniciar el servidor de aplicaciones (JBoss) y por último tratar de recuperar los mensajes iniciando nuevamente el servicio de ejemplo. En este caso, el sistema también funcionó del modo esperado recuperando los mensajes encolados, y respondiendo a dichas peticiones satisfactoriamente.

5.4. Comentarios

La realización de pruebas de estrés apropiadas debería haber consistido en la creación de un simulador de bearerbox para enviar mensajes al gateway y un simulador de JMS para recibir los mensajes sin el overhead de ejecutar un servidor de J2EE ni un bearerbox completo. A cambio, se ha utilizado un simulador de SMSC que simula estar conectado una red de telefonía celular, éste envía mensajes SMPP a través de una conexión TCP al bearerbox, quien envía los mensajes al gateway, y éste a continuación a un servicio de prueba.

La velocidad medida del simulador para enviar mensajes SMS solamente, varía entre los 100 y los 250 mensajes por segundo, este tiempo varía según el alojamiento de memoria del JVM del simulador, es decir, que entre más pruebas se hacen, la velocidad varía positivamente y se estabiliza alrededor de los 250 mensajes por segundo. Otra variable que afecta la velocidad es el tamaño del mensaje, estas pruebas se realizaron enviando mensajes de 160 caracteres que es el tamaño máximo de un mensaje SMS.

La velocidad de recepción y reenvío del Simulador de servicio varió alrededor de 1 mensaje por segundo, lo que parece extremadamente lento, la solución a ello es hacer un deploy del proveedor de JMS en un ambiente con mayor memoria disponible y configurado con el menor número de componentes habilitados. Estas pruebas fueron realizadas usando JBossMQ como message provider.

5.5. Evaluación del sistema

Este software logra eliminar el método tradicional de transferir mensajes SMS entre Kannel y aplicaciones de servicio, se probaron con éxito las características buscadas en JMS dentro de la aplicación, es decir, el uso de transacciones y la persistencia de

los mensajes. El diseño del software siempre fue distribuido y las conexiones entre los módulos: SMSC, bearerbox, kjGateway y la aplicación de ejemplo se realizó siempre usando la interfaz de red de loopback del sistema operativo mediante el uso de protocolos basados en TCP. Se logró escribir un gateway que no depende fuertemente del bearerbox para funcionar ni para ser administrado, es decir, autónomo.