UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

SISTEMA EXPERTO DE CONFIGURACIÓN PARA SISTEMAS TELEFÓNICOS A TRAVÉS DE DIAGRAMAS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

FABIÁN ALEXIS BRAVO ABARCA

PROFESOR GUÍA:
SR. PATRICIO INOSTROZA FAJARDÍN

MIEMBROS DE LA COMISIÓN:
SR. JOSE MIGUEL PIQUER GARDNER
SR. RODRIGO ARENAS ANDRADE

SANTIAGO DE CHILE
ABRIL 2012

Resumen

Este trabajo se realizó en Sixbell Nekotec Solutions, empresa que provee de soluciones de voz y datos a través de toda América. Su software permite desde recibir llamadas hasta procesarlas, donde, por ejemplo, se puede implementar un sistema de prepago o un call center.

El software central para el trabajo es el SCE, herramienta escrita en JAVA que permite, gráficamente, programar aplicaciones CCXML/VXML, que son las que operan en los sistemas de Sixbell. Su diseño permite, aunque con algunas limitaciones, la introducción de nuevos lenguajes que hacen que este programa pueda cambiar su funcionalidad, pareciéndose más a un IDE de programación gráfica, mediante entidades y enlaces entre ellos, por lo que cualquier problema que pueda ser modelado como flujo o diagrama podría ser resuelto con el SCE.

El configurador entonces se empezó a desarrollar como un lenguaje del SCE, pero para poder ser ocupado completamente, el SCE tuvo que ser modificado y varias partes rediseñadas de manera de no solo se compatible con este nuevo lenguaje, sino que hacer frente desde ya a nuevos problemas que puedan ser modelados con este especie de IDE gráfico. Para ello se modificó la manera en que se dibujan los elementos en pantalla, pues antes solo existían flechas para unir las entidades y ahora se pueden poner lineas con flechas en ambos sentidos o sin punta o con punta cuadrada, se hizo un completo refactoring de cómo se comportan los enlaces y demás elementos gráficos y además se trabajó con la retrocompatibilidad para hacer que aplicaciones antiguas del SCE pudieran ser usadas en las nuevas versiones.

El configurador terminado permitió a la empresa cambiar su forma de trabajar con los sistemas que venden y usan, desde su creación todos empezaron a adoptarlo como primera opción al configurar un sistema y rápidamente ya se tenían templates de aplicaciones del SCE practicamente listas para distintos escenarios. Además comenzaron a usarlo en distintos países donde Sixbell está presente como México y Colombia, donde sus unidades de negocio dieron muy buenos comentarios sobre esta herramienta.

En conclusión, se logró la creación de una poderosa herramienta, a partir de software ya existente en la empresa lo que da dos resultados positivos para Sixbell : el SCE como un producto aparte que puede modelar y solucionar más problemas que involucren flujos o diagramas, y el configurador que permite a la empresa tener una ventaja frente a sus competidores dadas sus características que benefician a los operadores de los sistemas y ayudan a mejorar los tiempos de puesta en marcha y de reacción frente a escenarios negativos.

Fabián Alexis Bravo Abarca

Índice General

Índice de figuras

Capítulo 1

Introducción

En el mundo de las telecomunicaciones los servicios deben estar disponibles en todo momento y ésto no es negociable. Cada minuto de error es tiempo en que las personas no se podrán comunicar o que la empresa estará incurriendo en pérdidas, ambas situaciones, especialmente la primera, son críticas.

Sixbell es proveedora de los sistemas informáticos de telecomunicación que hacen posible la telefonía celular, fija e IP. Junto con la capacidad de tarificar, distribuir y dar valor agregado a tales servicios.

La empresa, originaria de Chile, con presencia en México, Colombia y Brasil ha creado software variado, que permite desde recibir llamadas hasta procesarlas, donde, por ejemplo, se podría implementar un prepago que se necesita descontar el saldo en una base de datos, conectar la llamada con el otro teléfono y verificar el momento en que se corta la llamada, todas acciones soportadas por los sistemas de Sixbell.

El software implicado en este trabajo es el Service Creation Environment (SCE), que se puede ver en la figura 1.1. Este programa hecho en Java permite, gráficamente, programar aplicaciones CCXML/VXML, que son las que operan en los sistemas para lograr lo mostrado en el ejemplo anterior.
Además, dada la manera en que está diseñado, soporta la introducción de nuevos lenguajes (programados en Java y XSLT) que hacen que el SCE pueda comportarse de una manera diferente y, a través de similares diagramas, generar cualquier archivo requerido como, por

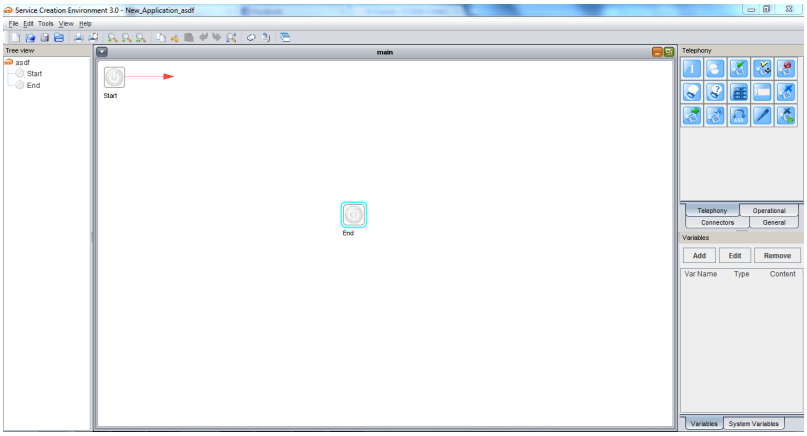

Figura 1.1: Pantalla principal del SCE, interfaz sobre la cual se montará el nuevo sistema de configuración.

ejemplo, los archivos de configuración de los sistemas informáticos de telecomunicación de Sixbell.

En el contexto de la flexibilidad del SCE nace la idea del Sistema configurador, aprovechando el trabajo realizado por los ingenieros creadores del SCE y considerando que la automatización es una parte crítica para el sector de las telecomunicaciones; la concentración de todo el conocimiento para levantar de manera correcta los servicios es un camino más que lógico y, pensando en el tema comercial, permite colocar a Sixbell en ventaja frente a su competencia.

Como solución a la necesidad detectada se creará una versión de configuración de SCE, aquí el usuario podrá construir esquemas de sistemas telefónicos, representando stacks de señalización, Media Resources, Tarjetas , Intérpretes CCXML/VXML, etc. Todo esto gracias al uso de iconos llamados *entidades*, que mediante sus conexiones proveen la información necesaria para que el sistema pueda generar la configuración de todos los programas. Con esta nueva capacidad del SCE, se pueden levantar los sistemas en un corto tiempo, mantener versionadas las configuraciones en caso de falla de un nuevo escenario y, por supuesto, todo de una manera automatizada.

Capítulo 2

Configuración de Sistemas, situación previa

Previo al trabajo realizado, la configuración de los sistemas telefónicos por parte de Sixbell se hacía de manera manual, ocupando en varias ocasiones archivos anteriores cambiando solo algunos parámetros. En cada una de las instalaciones realizadas se enviaban al menos dos ingenieros para poder realizar la tarea de configuración los cuáles tardaban dos o más días en terminar.
Cada uno de los componentes del sistema requiere gran cantidad de conocimiento específico y particular, que impide que personas externas, aun con el conocimiento técnico necesario, puedan ajustar la configuración de ellos.
A continuación se explicará la función de cada uno de los componentes y la forma en cómo se configuran.

2.1. TMP

El TMP o Telecom Media Portal de Sixbell, es un producto pensado para reemplazar a IVR convencionales tecnológicamente inferiores en cuanto a características y funcionalidades. Provee medios para generar una interacción con el usuario final, permite reproducir y grabar audio, reconocer opciones ingresadas por el usuario mediante DTMF o voz, enviar y recibir fax y permitir la creación y manejo de conferencias entre otras cosas.[1]

El TMP permite desarrollar aplicaciones en base a los lenguajes CCXML 1.0, VXML 2.1, NSL(Nekotec Scripting Languaje) y JavaScript, interactuando además con una serie de

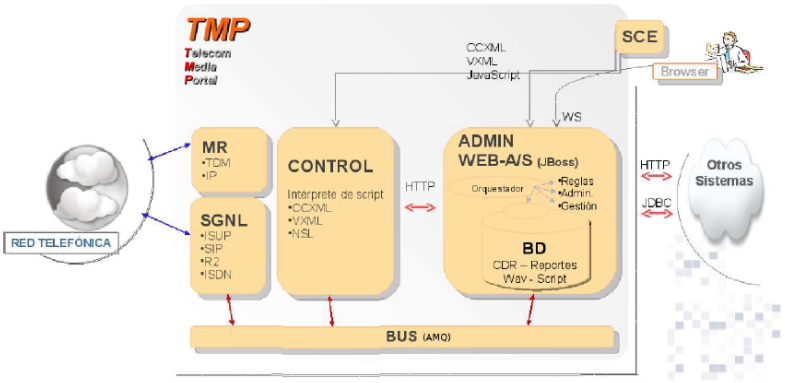

Figura 2.1: Principales módulos del TMP.

protocolos de comunicación, entre los cuáles se pueden encontrar :

- Protocolos de redes telefónicas tradicionales: ISUP, ISDN, CAS y otros
- Protocolos IMS: SIP, Diameter
- Protocolos sobre transporte IP: SMPP, SMTP, RTP y otros
- Habilitadores de servicios: conectores a Base de Datos Oracle, MySql y archivos de texto

Este producto está compuesto de varios módulos, los cuales pueden verse en el diagrama 2.1, y que serán explicados a continuación.

2.1.1. Control CCXML/VXML

El control es el proceso encargado de interpretar las aplicaciones escritas en los diferentes lenguajes soportados, como CCXML/VXML, destinadas a controlar el flujo de las llamadas telefónicas y a la creación de diálogos que permiten interacción con los usuarios finales de los sistemas, como por ejemplo pueden ser los clientes de un banco en el sistema que les informa su saldo en las cuentas. Esto se realiza mediante los diferentes módulos de la plataforma con los que se comunica, los cuales brindan acceso a redes telefónicas, bases de datos o servicios que proveen lógica de negocio.

El trabajo de éste módulo esta condicionado por los módulos externos, quienes reciben los diferentes eventos del sistema, como una llamada telefónica que viene ingresando, y luego comunican al control para que éste comience a interpretar la aplicación para atender el evento.

Su configuración se realiza a través de dos archivos, el sigas-control2.conf que es el principal y el denominado routing rules".

Archivo de configuración principal

El archivo sigas-control2.conf[2] contiene los parámetros de las características centrales del Control, es por esto que se recomienda usar una plantilla básica de configuración y modificar solo los valores estrictamente necesarios. Un archivo de ejemplo se puede encontrar en los anexos, ver la sección A en la página 66.
Se divide en diferentes secciones, cada una hace referencia a los diferentes módulos con que se debe comunicar el control, éstas son :

- **Stats** : que contiene información relativa a las estadísticas
- **Interpreter** : que tiene los parámetros relativos a los intérpretes CCXML y VXML
- **Gateway** : configura el comportamiento del control, para operación como gateway
- **Traps** : información relativa al servidor donde se dirige la información de traps
- **Monitor** : donde se coloca la información del servidor de monitoreo de la plataforma
- **Console** : información requerida para la consola del control, para que éste último se pueda operar remotamente
- **Logs** : localiza el archivo *log* del control
- **Debug** : configura el nivel de log que tendrá el control
- **Media** : configura el comportamiento y alcance de funcionalidades de media respecto a recursos de TTS y ASR
- **ActiveMQ** : configura la lista de sesiones a establecer con ActiveMQ, donde se inscriben las colas desde las que el control recibirá los mensajes de los diferentes módulos.

Archivo *Routing rules*

Este archivo contiene las reglas de ruteo, permiten al control tomar decisiones basadas en las reglas que son configuradas por el usuario.
Un archivo de ejemplo se puede encontrar en los anexos, ver la sección B en la página 68.
Aquí se pueden configurar el comportamiento que tendrá el control, de manera que dependiendo de qué eventos sean recibidos por la plataforma el sistema responda ejecutando una u otra aplicación.
Lo anterior se logra definiendo bloques, parecido a un archivo JSON, que son representados por la ruta al archivo CCXML que contiene la aplicación, para luego dentro de ellos definir las condiciones que al cumplirse harán que se interprete, las cuales pueden ser expresiones lógicas de las siguientes variables:

- Número de origen de la llamada entrante (ANI)
- Número de destino (DNIS)

Además pudiendo usar los siguientes operadores sobre dichas variables :

- **startswith** Verifica si el parámetro empieza con un valor
- **endswith** Verifica si el parámetro termina con un valor
- **contains** Verifica si el parámetro contiene un valor
- **equals** Verifica si el parámetro es igual a un valor, como cadena de texto
- **equalsnumeric** Verifica si el parámetro es igual a un valor, comparándolos como números
- **greaterthan greaterorequals** Verifica si el parámetro es mayor (o igual) numéricamente a un valor
- **lowerthan lowerorequals** Verifica si el parámetro es menor (o igual) numéricamente a un valor

Así por ejemplo una regla que establece que el ANI de la llamada debe comenzar con los dígitos 800, quedaría de la siguiente forma :

oper.ANI—startswith—800

2.2. Media Resource NMS

El Media Resource (MR) es el proceso encargado de ejecutar las funciones de audio del control, permitiendo así la interacción con los abonados mediante audio. Funciona en base a las peticiones enviadas por el control que tienen la forma *request/response.*[2]
El MR permite el manejo del audio para comunicarle al cliente el estado o situaciones de atención en el curso de la llamada. Ejemplos comunes son, el solicitar información al usuario, indicar el saldo que le queda al usuario o avisar sobre algún error en la llamada.
Entre sus características se pueden encontrar:

- Soporte de tarjetas NMS CG
- Comunicación con Control VXML/CCXML
- Integración con protocolos SIP, ISUP, ISDN, R2
- Play y record
- Say As
- Manejo de protocolo MRCP 2.0 para TTS y ASR
- Manejo de protocolo RPT, soporte de codec disponible para SIP
- Transferencia de llamadas

Su configuración se realiza a través de un archivo, el cuál se debe indicar al momento de levantar el servicio.
Además se debe tener en cuenta que parte del funcionamiento del MR depende de las tarjetas de audio (NMS), por lo que éstas deben estar debidamente configuradas y sus servicios arriba antes de iniciar el MR.

Archivo de configuración

A diferencia de otros archivos de configuración, éste se especifica al ejecutar el servicio, a través de un parámetro al ejecutar el binario. Esto cobra vital importancia después, al querer tratar de automatizar al configuración y carga de éste servicio, objetivo de este trabajo de título.
Un archivo de configuración estándar se puede encontrar en los anexos, para ello ver la sección C en la página 69. Ahí se encontrarán los siguientes elementos listados a continuación :

- **configuracion** en esta sección se ajustan los parámetros de las colas de ActiveMQ que serán usadas por el servicio, además de otros elementos que tienen que ver con intervalos de actualización y rutas de guardado para archivos traídos desde internet
- **ConsTabla** acá se configuran las rutas a los archivos de configuración de las tarjetas y de canales
- **Gestion** contiene la información de los servidores que recibirán alarmas desde este servicio (*traps*) y también la ruta al archivo que configura las alarmas que se enviarán
- **IP** acá se especifica si se ejecutará el servicio bajo NAT, se configura cómo se procesarán los DTMF y codecs de audio a usar
- **PSTN** aquí se definen los parámetros necesarios para que el MR se conecte con las tarjetas NMS CG, así también como las reglas de fecha y hora para los diferentes idiomas de las voces que se poseen
- **Trace** parámetros que definen el nivel de log que se quiere para este servicio

2.2.1. Señalizador ISUP

El objetivo del señalizador ISUP es proveer a la plataforma de un medio de acceso hacia una red SS7 de manera de poder interactuar con *signaling points* externos a través del protocolo ISUP, utilizando el Stack SS7.[2]
En términos funcionales la tarea de este señalizador es extraer la información relevante de la mensajería recibida desde la red y enviarla hacia el Control de manera que éste pueda tomar las decisiones correspondientes de acuerdo a la lógica de servicio implementada en los scripts

```
<library name="libengine-isup_hss.so" configfile="/opt/sixlabs/etc/
   engines/engine-conector-isup_hss.xml"/>
```

Figura 2.2: Entrada para inscribir el *engine* en la lista que tiene *Engines loader*

cargados en él. Acciones iniciadas desde el Control, como llamadas salientes, también son soportadas.

Entre sus características está:

- Soporte para el protocolo ISUP
- Comunicación con el módulo Control CCXML/VXML y también NSL
- Comunicación con el módulo Stack de señalización SS7 o SIGTRAN
- Comunicación con el módulo SMIlodon a través de ActiveMQ
- Configuración vía archivo XML

Su configuración se realiza mediante dos archivos XML, el primero configura el módulo *Engines loader*, que es el responsable de cargar todas las bibliotecas dinámicas o plugins que implementan alguna funcionalidad de algún protocolo en particular. El segundo archivo contiene los parámetros configurables de este *engine* en particular.

Configuración de *Engines Loader*

En este caso solo se debe agregar este *engine* a la lista mediante la linea que aparece en la figura 2.2. En ella se especifica el módulo a cargar y la ruta hacia el archivo de configuración a ocupar para cargarlo.

Configuración del señalizador ISUP

Como se mencionó antes, se realiza a través de un archivo XML, del cuál se puede ver una muestra en los anexos (ver sección D en la página 70). Ahí se define cuáles serán las colas que usará el módulo para comunicarse a través de ActiveMQ, datos relativos a los recursos ISUP como *point codes* y *circuitos* asociados, así como también los relacionados con

los recursos de medios, requeridos por el módulo MR.[2]
Hay que notar en este punto que es importante que en la sección de *point codes* y circuitos a usar se tenga exactamente los mismos datos que se manejan en los demás módulos. Por lo tanto acá se hace conveniente la definición de los grupos de circuitos, rutas de conexión, *point codes* de origen y destino, estén previamente definidos y sean replicados en cada uno de los lugares que correspondan, como en este archivo de configuración y posteriormente la capa ISUP del Stack SS7 que se verá más adelante.

2.2.2. Señalizador NCC

El objetivo del señalizador NCC NMS es permitirle al Control poder manipular eventos de ISDN y protocolos CAS utilizando la API de **Natural Call Control**, tanto para eventos de entrada como de salida.[2]
Sus características son:

- Soporte de protocolo ISDN
- Soporte de protocolos CAS (R2/LS1)
- Comunicación con módulos Control CCXML/VXML y NSL
- Comunicación con stack de la tarjeta NMS
- Comunicación con el módulo SMIlodon a través de ActiveMQ
- Configuración en archivo XML

Al igual que el anterior señalizador, este se configura a través de dos archivos XML, uno que inscribe éste *engine* en el *Engines loader* y el otro que configura los parámetros que son propios del módulo.

Configuración de *Engines loader*

En el archivo de configuración de *Engines loader* es necesario inscribir el módulo agregando la linea que aparece en la figura 2.3.

```
<library name="libengine-ncc_nms.so" configfile="/opt/sixlabs/etc/
   engines/engine-conector-ncc_nms.xml"/>
```

Figura 2.3: Entrada para inscribir el *engine* en la lista que tiene *Engines loader*

Configuración del Señalizador NCC

En éste archivo es donde se configuran las colas que se usarán en ActiveMQ para poder comunicarse con los otros módulos, el canal donde se podrá comunicar con la tarjeta NMS, los datos relativos a los recursos de señalización, así como también los correspondientes a los recursos de medios, requeridos por el módulo MR.[2]
Un ejemplo de este archivo se puede encontrar en los anexos, ver la sección E en la página 71.

2.2.3. Señalizador SIP

El señalizador SIP tiene como objetivo permitir al Control poder manipular eventos SIP[5], tanto de entrada como de salida. Entre sus características están :

- Soporte para el protocolo SIP
- Comunicación con los módulos Control CCXML/VXML y NSL
- Comunicación con el módulo Stack de señalización SIP
- Comunicación con el módulo SMIlodon a través de ActiveMQ
- Archivo de configuración XML

Similarmente a los demás módulos de señalización, su configuración se realiza por medio de dos archivos de configuración, los cuales se describen a continuación.

Configuración de *Engines loader*

En el archivo de configuración de *Engines loader* es necesario inscribir el módulo agregando la linea que aparece en la figura 2.4.

```
<library name="libengine-sip_radvision.so" configfile="/opt/sixlabs/
   etc/engines/engine-conector-sip_radvision.xml"/>
```

Figura 2.4: Entrada para inscribir el *engine* en la lista que tiene *Engines loader*

Configuración del Señalizador SIP

En el caso de este señalizador, en la configuración se deben especificar dos buses ActiveMQ, uno para la comunicación con el Control y otro para la comunicación del Stack. Para cada uno de ellos se deben especificar el nombre de las colas desde donde se extraerá y colocará información.
Por otra parte también se configura una instancia de comunicación especial para recibir mensajes desde el SMIlodon.

2.2.4. Connector DBEnabler

El DBEnabler de Sixbell, ofrece la posibilidad de conectarse con diferentes bases de datos, locales o remotas, de distintos motores, como MySQL y Oracle, para ejecutar acciones de consulta, creación, actualización y eliminación de datos.

En su configuración participan varios archivos, pero la información relevante que es necesario definir es la de conexión a las bases de datos, donde hay que colocar la dirección donde se encuentra, usuario y contraseña. Además se debe configurar el bus ActiveMQ por el cuál se comunicará con el resto de los módulos.

2.2.5. Connector File

El *Connector File* es un módulo que provee funcionalidades de escritura y lectura sobre archivos de texto, ejecución de comandos de sistema o *shell scripts* a entidades que posean soporte para mensajería de Sixbell.[2]

Su configuración se realiza mediante dos archivos, al igual que los señalizadores, uno para *Engines Loader* y otro para parámetros propios del módulo.

```
<library name="libengine-file_sixlabs.so" configfile="/opt/sixlabs/
    etc/engines/engine-file_sixlabs.xml"/>
```

Figura 2.5: Entrada para inscribir el *engine* en la lista que tiene *Engines loader*

```
<library name="libengine-smpp_sixlabs.so" configfile="/opt/sixlabs/
    etc/engines/engine-smpp_sixlabs.xml"/>
```

Figura 2.6: Entrada para inscribir el *engine* en la lista que tiene *Engines loader*

Configuración de *Engines loader*

En el archivo de configuración de *Engines loader* es necesario inscribir el módulo agregando la linea que aparece en la figura 2.5.

Configuración de *Connector File*

Para este módulo tan solo se deben especificar las colas de ActiveMQ que se usarán, además de la forma, también a través de ActiveMQ, en como se conectará con la consola del SMIlodon.
Un ejemplo de este archivo se puede ver en los anexos, sección F en la página 72.

2.2.6. Connector SMPP

El objetivo del *Connector SMPP* es proporcionar conectividad entre el SigAS y un **Short Message Service Center** (SMSC) mediante protocolo SMPP, su principal funcionalidad es enviar y recibir mensajes cortos (SMS) de un SMSC.[2]

De forma análoga al *Connector File* usa dos archivos de configuración, uno para *Engines Loader* y otro para especificar sus parámetros de funcionamiento.

Configuración de *Engines Loader*

En el archivo de configuración de *Engines loader* es necesario inscribir el módulo agregando la linea que aparece en la figura 2.6.

```
<library name="libengine-smtp_sixlabs.so" configfile="/opt/sixlabs/
   etc/engines/engine-smtp_sixlabs.xml"/>
```

Figura 2.7: Entrada para inscribir el *engine* en la lista que tiene *Engines loader*

Configuración de *Connector SMPP*

En el archivo XML, del cuál se puede ver un ejemplo en los anexos (ver sección G en la página 73), se debe especificar la siguiente información:

- Parámetros SMPP, donde se debe incluir el nombre de la entidad que se conecta al SMSC, la versión del protocolo y datos exclusivos del protocolo como TON, NPI y rango de direcciones
- Datos del socket de conexión SMSC
- Colas de ActiveMQ por las cuales se comunicará con los demás módulos, en particular el Control
- Comunicación, a través de ActiveMQ, con la consola de SMIlodon

2.2.7. Connector SMTP

El *Connector SMTP* es un módulo que provee funcionalidades de cliente de envío de correos electrónicos vía el protocolo SMTP (Simple Mail Transport Protocol). Este módulo además posee soporte para MIME (Multipurpouse Internet Mail Extensions) con lo cual se habilita el envío de correos con archivos adjuntos.[2]

De forma análoga a los *connectors* anteriores, se debe inscribir el módulo en el *Engines loader* y además se deben configurar los parámetros propios en un archivo XML dispuesto para ello.

Configuración de *Engines Loader*

En el archivo de configuración de *Engines loader* es necesario inscribir el módulo agregando la linea que aparece en la figura 2.7.

```
<library name="libengine-socket_sixlabs.so" configfile="/opt/sixlabs
   /etc/engines/engine-socket_sixlabs.xml"/>
```

Figura 2.8: Entrada para inscribir el *engine* en la lista que tiene *Engines loader*

Configuración de *Connector SMTP*

En el archivo XML de configuración se debe especificar la dirección del servidor de correos que se usará, las colas ActiveMQ para comunicarse con los demás módulos así como también los medios por los que se comunicará con la consola de SMIlodon.
Un archivo de ejemplo se puede encontrar en los anexos, en la sección H , página 74.

2.2.8. Connector Socket

La finalidad del *Connector Socket* es proporcionar a la plataforma un medio para realizar transacciones síncronas del tipo *request/response* con alguna entidad externa, a través de una conexión TCP de propósito general.

Igual que los *connectors* anteriores se debe inscribir en el *Engines loader* y posee un archivo XML donde se configuran sus propios parámetros.

Configuración de *Engines Loader*

En el archivo de configuración de *Engines loader* es necesario inscribir el módulo agregando la linea que aparece en la figura 2.8.

Configuración de *Connector Socket*

En su archivo de configuración XML se debe colocar la cantidad de información máxima permitida para el socket, así como también las colas con las que se comunicará, a través de ActiveMQ, con los otros módulos y con la consola de SMIlodon. También se debe introducir el tiempo por el que este módulo esperará por una respuesta antes de rendirse en la conexión.

Un ejemplo de éste archivo de configuración puede encontrarse en los anexos, en la sección I página 75.

2.3. Stack SS7

El Stack de Señalización 7 de Sixbell está conformado por múltiples entidades de software relacionadas en forma lógica, de manera de brindar las funcionalidades de las capas de protocolo MTP2, MTP3, ISUP y/o SCCP, permitiendo que sobre ellas se puedan desarrollar todo tipo de aplicaciones que requieran realizar control de llamadas entrantes o salientes.[6] Básicamente el stack se compone de tres módulos que interactuan entre si, de manera de proporcionar las funcionalidades antes descritas :

- **MTP2** : Capa de convergencia que permite manejar ciertas tarjetas de señalización que implementan funcionalidades hasta el protocolo MTP2 (alineación, alarmas, envío y recepción de MSU, etc...), y que además se encarga de interactuar (mediante API de Sixbell) con el módulo siguiente, el cuál implementa MTP3 y superiores[7]

- **MTP3 más ISUP y/o SCCP** : Modulo que maneja estos protocolos y que los comunica entre si. Además este módulo se comunica con la capa de convergencia MTP2 (de manera de enviar MSU a la red) y hacia al usuario de SCCP y/o ISUP permitiendo de esa forma que pueda hacer control de llamadas[7]

- **SMT (SS7 Management Tool)** : Este módulo se encarga de configurar lo relacionado a los protocolos MTP3(rutas, point codes), ISUP (circuitos), SCCP (nodos adyacentes, SSN, etc...), además de permitir labores de operación y administración sobre algunas variables de los protocolos de señalización SS7 soportados[8]

Normalmente para la implementación de IVR solo es necesario un stack que implemente MTP3 y ISUP, pero si se quiere desarrollar una aplicación de redes inteligentes en general, será necesario ocupar un stack que incluya las capas MTP3 y SCCP.

Un diagrama de cómo se relaciones los diferentes módulos del stack SS7 puede verse en la figura 2.9.

También éste stack se puede instalar de forma redundante, del tipo *active-standby*, en cuyo caso se debe agregar un nuevo módulo llamado RD, el cuál realiza un monitoreo del estado del par de nodos que compondrían el stack (así como cada uno de sus procesos) y determina

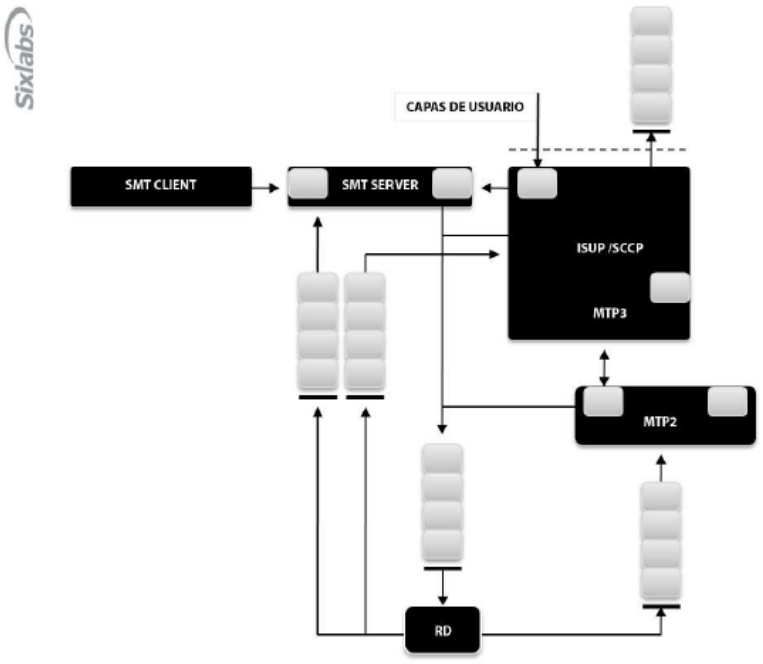

Figura 2.9: Diagrama de relación entre los módulos del Stack SS7.

las acciones a realizar en caso de fallos, como por ejemplo: traspasar el procesamiento del tráfico desde un nodo con problemas hacia el nodo en *standby*.

2.3.1. Configuración capa MTP2

La capa de convergencia MTP2 es la encargada de comunicarse directamente con las tarjetas de señalización SS7 soportadas, accediendo a funcionalidades básicas como : alienación de links, alarmas, envío y recepción de MSU[1], procedimientos de congestión y disponibilidad de links, etc. Además, este módulo se encarga de interactuar directamente con el módulo que implementa MTP3 a través de sockets TCP, utilizando API propietarias para el envío de comandos entre ellos.

Configuración del módulo MTP2

El módulo MTP2 se configura mediante el archivo mtp2_ph_conf (nombre por defecto para este archivo). En él se encuentra la información de rutas, ubicaciones físicas, timers

[1]Message Signal Unit (MSU) es una unidad de señal del protocolo SS7 que contiene información sobre el servicio y campos de señalización (como los *Point code* de origen y destino).

MTP2, información de IPC (valores de puertos TCP, direcciones IP, ID de colas Unix), valores físicos de links en las tarjetas de señalización, etc., lo cuál es configurado a través de múltiples parámetros, los cuales se pueden ver en el archivo de ejemplo puesto en la sección de los anexos J .

En éste archivo de configuración se pueden ver las siguientes secciones importantes :

- **Log section** donde se puede cambiar el nivel de log que se quiere del módulo y donde se dejará guardado.
- **Queues section** lugar en el cual se configuran las colas que usará el módulo para comunicarse con los demás.
- **TCP/IP ports section** que contiene los puertos que ocupará el módulo.
- **Trunks section** sección en la que se especifica cada una de las bocas de las tarjetas de señalización.
- **MTP3 section** con datos de cómo el modulo MTP3 ve a las bocas de las tarjetas físicas. Si por ellas pasará una conexión nacional, internacional, etc.
- **MTP2 timers section**
- **MTP2 links section** contiene la configuración de cada una de las bocas físicas de las tarjetas de señalización. Es acá donde se indica el tipo de tarjeta, el driver que se ocupará y si la boca estará activa o no.
- **Secciones de cada boca** por cada una de las bocas de señalización hay que hacer una sección diferente, en donde se configura información de bajo nivel que será usada por la tarjeta para el funcionamiento del enlace en tal boca.

2.3.2. Configuración Capa MTP3 + ISUP

Éste es un módulo que puede manejar los protocolos ISUP y/o SCCP e incluso comunicar entre si. Además, se conecta con la capa MTP2 (de manera de enviar MSU a la red) y con el usuario de SCCP/ISUP permitiendo de esta forma que se pueda hacer el control de llamadas. Su configuración se realiza mediante un archivo usualmente nombrado como ss7_ph.conf, un ejemplo se puede ver en los anexos en la sección K , cuyas secciones son :

- **Stack log data** lugar en el que se especifica cuánta memoria ocupará el log del módulo y dónde se guardará el archivo de log.
- **MTP3 Standard and node type** donde se define cómo el stack actuará en la red.
- **IP address used** contiene los datos de IPC[1].
- **RD process communication** si se quiere tener un stack redundante entonces se debe ingresar la información para comunicar los procesos entre si en este bloque del archivo.
- **ISUP multiqueue** acá se configura cómo la capa MTP3 se conectará con la capa ISUP del TMP, si para ello usará colas de ActiveMQ o una cola UNIX. También se indican todas las conexiones de ISUP con las distintas centrales y cuales circuitos de las bocas de las tarjetas de señalización serán usados para cada una de ellas.
- **Ports** bloque en donde se listan los puertos que serán usados por todos los módulos del stack.
- **MTP2 conv** sección donde se clasifican las bocas de las tarjetas de señalización entre locales (de la máquina donde está corriendo el módulo que se está configurando) y remotas (las que pertenecen al stack redundante).

2.3.3. Rutas

Luego de que todos los módulos del stack están ejecutándose, se deben configurar las rutas que habrán desde el stack de señalización hasta las centrales. Para ello, a través de comandos de la consola del stack, se van ingresando los diferentes circuitos disponibles (los mismos que se especifican en el archivo de configuración de la capa MTP3), los diferentes DPC[2] que pueden ser alcanzados por el stack, el OPC[3] del sistema. Una vez terminado lo anterior se comienza a especificar cómo cada central, identificada por su DPC, será alcanzada, a través de qué circuitos, si se hará de manera directa o indirecta (pasando por STP[4] por ejemplo) y si es indirecta cuál es el STP preferido para llegar hasta la central.
Aparte a través de la consola se pueden configurar varios comportamientos del stack, como

[1]Interprocess Communication Protocol, es un protocolo de capa de transporte (capa 4) en el Stack del protocolo VINES, provee servicio de envío de mensajes y datagramas de forma confiable.
[2]Destination Point Code
[3]Origin Point Code
[4]Signaling Trasfer Point

alarmas y timers.

Un ejemplo de configuración de rutas puede verse en la figura 2.10.

```
ADD-SPC:SPC=333,TYPE=PRIMARY;
ADD-DPC:DPC=124,TYPE=ADJACENT;
ADD-DPC:DPC=125,TYPE=ADJACENT;
ADD-DPC:DPC=121,TYPE=REMOTE;
ADD-DPC:DPC=122,TYPE=REMOTE;
ADD-DPC:DPC=123,TYPE=REMOTE;
ADD-LINK:ID=1,OPC=333,DPC=125,SLC=0,MTP2-ID=1,LOG-DATA-LINK-ID=0,TYPE=A;
ADD-LINK:ID=2,OPC=333,DPC=124,SLC=1,MTP2-ID=1,LOG-DATA-LINK-ID=1,TYPE=A;
ADD-LINK:ID=3,OPC=333,DPC=125,SLC=1,MTP2-ID=2,LOG-DATA-LINK-ID=0,TYPE=A;
ADD-LINK:ID=4,OPC=333,DPC=124,SLC=0,MTP2-ID=2,LOG-DATA-LINK-ID=1,TYPE=A;
ADD-LINKSET:ID=1,OPC=333,DPC=124,NUM-NORMAL=1,BRDCAST=DISABLE,LINK-ID=2,STATE=ACTIVE;
ADD-LINK-LINKSET:LINKSET-ID=1,LINK-ID=4,LINK-PRIO=0,STATE=ACTIVE;
ADD-LINKSET:ID=2,OPC=333,DPC=125,NUM-NORMAL=1,BRDCAST=DISABLE,LINK-ID=1,STATE=ACTIVE;
ADD-LINK-LINKSET:LINKSET-ID=2,LINK-ID=3,LINK-PRIO=0,STATE=ACTIVE;
ADD-ROUTE:ID=1,OPC=333,DPC=121,ROUTE-TYPE=INDIRECT,LINKSET-ID=1,ROUTE-PRIO=3;
ADD-ROUTE:ID=2,OPC=333,DPC=122,ROUTE-TYPE=INDIRECT,LINKSET-ID=1,ROUTE-PRIO=3;
ADD-ROUTE:ID=3,OPC=333,DPC=123,ROUTE-TYPE=INDIRECT,LINKSET-ID=1,ROUTE-PRIO=3;
ADD-ROUTE:ID=4,OPC=333,DPC=121,ROUTE-TYPE=INDIRECT,LINKSET-ID=2,ROUTE-PRIO=2;
ADD-ROUTE:ID=5,OPC=333,DPC=122,ROUTE-TYPE=INDIRECT,LINKSET-ID=2,ROUTE-PRIO=2;
ADD-ROUTE:ID=6,OPC=333,DPC=123,ROUTE-TYPE=INDIRECT,LINKSET-ID=2,ROUTE-PRIO=2;
ADD-ROUTE: ID=7,OPC=333, DPC=124,ROUTE-TYPE=DIRECT, LINKSET-ID=1,ROUTE-PRIO=0;
ADD-ROUTE: ID=8,OPC=333, DPC=125,ROUTE-TYPE=DIRECT, LINKSET-ID=2,ROUTE-PRIO=0;
ADD-ISUPCCTGRP:OPC=333,DPC=123,START-CIC=1,NUM-CIC=30,PROP-DELAY=1,IS-SATELLITE=FALSE;
ADD-ISUPCCTGRP:OPC=333,DPC=123,START-CIC=32,NUM-CIC=30,PROP-DELAY=1,IS-SATELLITE=FALSE;
ADD-ISUPCCTGRP:OPC=333,DPC=121,START-CIC=1,NUM-CIC=30,PROP-DELAY=1,IS-SATELLITE=FALSE;
ADD-ISUPCCTGRP:OPC=333,DPC=122,START-CIC=1,NUM-CIC=30,PROP-DELAY=1,IS-SATELLITE=FALSE;
```

Figura 2.10: Ejemplo de comandos de la consola del Stack SS7

Capítulo 3

SCE 3.0

SCE, o Service Creation Environment, es el programa creado por Sixbell para facilitar la creación de aplicaciones de telefonía para su intérprete CCXML/VXML, llamado Sigas Control. Los usuarios en el SCE pueden modelar el flujo de las llamadas mediante entidades, las cuales, corresponden a pequeños cuadrados con una imagen característica, que además tienen propiedades que se editan a través de diálogos. Su unión se hace a través de flechas permitiendo colocar las acciones a seguir dada una determinada entidad. En la figura 3.1 se puede ver un ejemplo de aplicación simple, que recibe una llamada, ejecuta un mensaje y luego corta.[1]

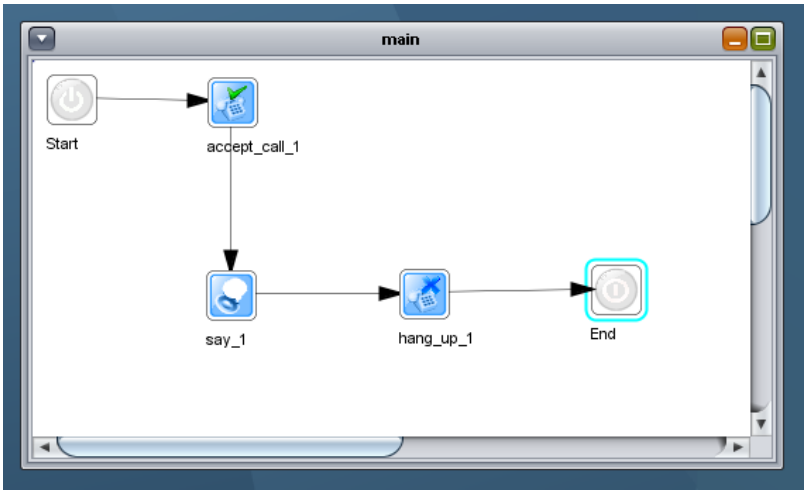

Figura 3.1: Aplicación de ejemplo cuya única función es reproducir un mensaje a la persona que llama para luego cortar.

Es así como el usuario a través del SCE puede crear que van desde simples menús de atención automática hasta complejos sistemas de prepago para compañías de telefonía celular. De manera gráfica, simple y sin necesidad de experiencia en CCXML y VXML, lenguajes

estándares para la creación de estas aplicaciones y que son usados por la mayoría de los programas disponibles para este tipo de soluciones telefónicas. El resultado final de las aplicaciones del SCE, en todo caso, es la generación de los archivos CCXML/VXML necesarios para su ejecución en el intérprete de Sixbell, llamado Sigas-Control.

Actualmente la interfaz del SCE está, en gran medida, separada de lo que es el lenguaje propiamente tal, que viene siendo la definición de las entidades, sus diálogos y forma de generar los archivos resultantes. Ésta característica resulta atractiva para su utilización con otros fines y es la que se explotará para crear un nuevo lenguaje, para facilitar la configuración de todo un sistema de telefonía armando su diagrama y fijando ciertas propiedades claves sin tener que editar grandes archivos de configuración.

3.1. Componentes de una aplicación

Toda aplicación, independiente el lenguaje que se este utilizando en el SCE, tiene componentes básicos, los cuáles se detallan a continuación:

- **Ventana principal** : En la figura 3.1 se puede ver la ventana principal de la aplicación, la cuál es llamada *main*. Posee dos entidades siempre, que no se pueden eliminar, que son el comienzo y final del flujo.

- **Entidad** : Una entidad es la unidad básica de una aplicación y se representa como un nodo. Tiene una imagen representativa que entrega cierta idea acerca de su funcionalidad, además de elementos generales como un nombre y un conjunto de propiedades. Provee de una interfaz de usuario para la manipulación del valor de sus propiedades llamada diálogo. También posee enlaces de salida que permiten la confección del flujo.

- **Entidad Contenedora** : Es una entidad pero que guarda una subaplicación dentro. Se usan para dar más orden a un flujo y encapsular ciertos comportamientos, como si fuera una función de algún lenguaje de programación conocido.

- **Subaplicación** : Es una ventana parecida a la principal de la aplicación que es representada por una entidad contenedora. También posee entidades de comienzo y fin, pero los enlaces que salen de la entidad de comienzo representan a los que llegan a la

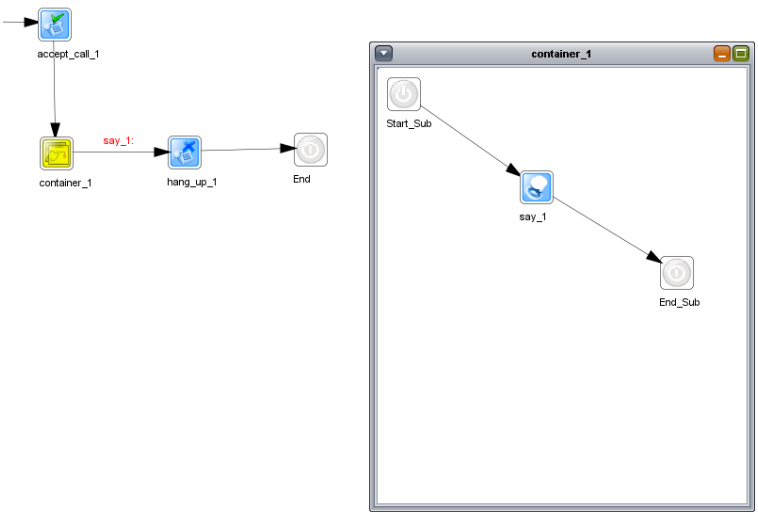

Figura 3.2: Entidad contenedora *container 1* y la subaplicación asociada cuyo título de ventana tiene el mismo nombre.

entidad contenedora y los que salen son representados en la entidad contenedora como si fueran sus propias flechas como se ve en la figura 3.2. Hay que notar que la ventana principal *main* al guardarse la aplicación también es tratada como una subaplicación, pero sin ninguna entidad asociada.

- **Enlace** : Es la flecha o conexión entre entidades y permite definir el flujo que tendrá una aplicación formando un grafo dirigido de entidades.

- **Diálogo** : Cada entidad posee un diálogo o ventana en donde se pueden editar propiedades de ella o aumentar o disminuir los enlaces salientes en algunas de ellas. En la figura 3.3.

- **Propiedad** : Valores que pueden ser cambiados en un diálogo para cada entidad y que son particulares para cada una de ellas, previamente definidos en el lenguaje.

3.2. Guardado de una aplicación

Toda aplicación hecha en SCE es guardada en un archivo XML que incluye desde la posición de cada una de las entidades hasta el valor de cada una de sus propiedades.
La definición del árbol es importante para luego crear los lenguajes, ya que es a través de

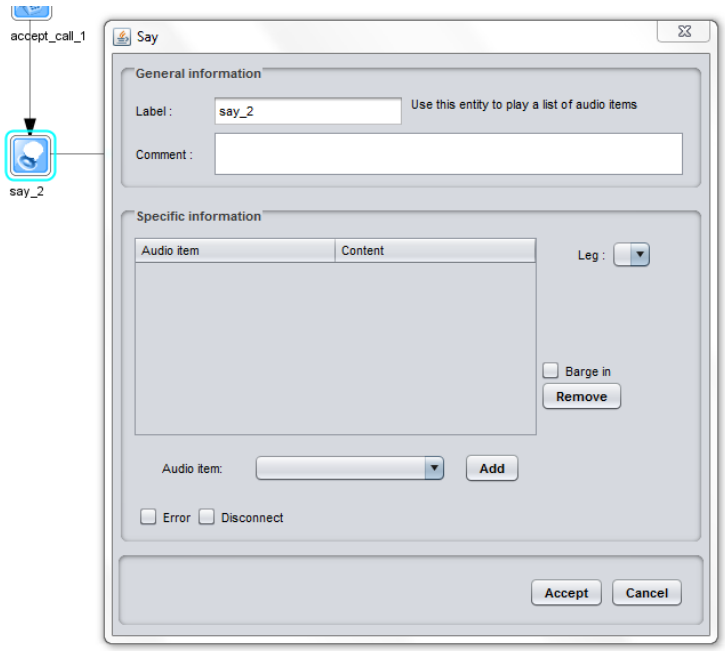

Figura 3.3: Entidad *say* y su diálogo.

este archivo que se podrán generar, con el lenguaje para aplicaciones telefónicas por ejemplo, archivos CCXML/VXML o los archivos de configuración de las máquinas y servicios, en el caso del objetivo de este trabajo.

3.2.1. Árbol XML

A continuación se presenta el detalle de las etiquetas usadas para guardar una aplicación SCE en un archivo XML, desde la raíz a las hojas :

ApplicationGUI

Raíz del árbol, marca el comienzo de una aplicación del SCE. Guarda información como : nombre de la aplicación, fecha de guardado, comentarios, lenguaje usado para crear la aplicación, versión del SCE.

SubApplicationGUI

Cada subaplicación, incluida la antes mencionada main, se guarda bajo esta etiqueta. Dentro de ella se agregan todas las entidades incluidas en la subaplicación con la etiqueta *EntityGUI*. Sus atributos indican el nombre de la subaplicación, la entidad contenedora asociada a ella (o un 0 si es que es la ventana principal y, por lo tanto, no tiene asociación).

EntityGUI

Cada una de las entidades de la aplicación, incluso las contenedoras, es descrita con una de estas etiquetas. A través de sus atributos se guardan : un identificador único, el nombre de la aplicación, el tipo de entidad (según el lenguaje), comentarios y la posición dentro de la ventana de la subaplicación.
Todo *EntityGUI* posee siempre dos ramas, *Dest* y *Props*, las cuales describen los destinos de los enlaces que posee y los nombres y valores de las propiedades que contiene, respectivamente.

Dest

Es la rama hija de una etiqueta *EntityGUI* en donde se colocan, mediante la etiqueta *LinkGUI*, los enlaces salientes que posee la entidad.

LinkGUI

Esta etiqueta se usa para guardar un enlace saliente de la entidad. En sus atributos se guarda la posición de inicio y el final de la flecha, la entidad de destino o null si es que no está conectada la flecha, la posición del texto, si tiene, y la referencia a otro enlace si es que éste es el resultado de una flecha que salía de una subaplicación para ser representado fuera, como si fuera una flecha de la entidad contenedora.

Props

Otra rama hija de la etiqueta *EntityGUI* que contiene, mediante la etiqueta *PropertyGUI*, las propiedades de la entidad.

PropertyGUI

Las propiedades de las entidades se guardan a través de ésta etiqueta. Sus atributos guardan : el nombre de la propiedad, el tipo (si es de texto, un número o verdadero y falso), si es singular o múltiple, si genera flechas y, por supuesto, el valor que se le ha asignado a la propiedad.

MultiPropValueGUI

En caso de que una variable sea múltiple, algo parecido a un arreglo, se usa ésta etiqueta por cada uno de los valores que contiene la propiedad y se dejan como hojas bajo *Property-*

GUI. En sus atributos están el identificador y el valor.

Un ejemplo de archivo XML del SCE puede encontrarse en los anexos, en el cuál se encuentra la aplicación mostrada en la figura 3.1.

3.3. Lenguaje

Un lenguaje para el SCE es la definición de las entidades, iconos, conexiones, diálogos y generador de archivos. Tal como un lenguaje de programación y su entorno de desarrollo, tal definición entrega solo las herramientas para crear las aplicaciones y es el SCE el que las integra para que el usuario pueda lograr sus objetivos.

3.3.1. Definición de un lenguaje

El lenguaje empieza a definirse en una clase de JAVA llamada *CreateLanguage*, que usa a la clase *Language* proveída por SCE, en donde se especifican las entidades, sus propiedades, las clases que contienen los diálogos y los iconos que tendrán los cuadrados que representarán las entidades.
El objeto de la clase *Language*, después completada la definición, es serializado y guardado en el archivo que finalmente se carga al SCE, el que, para diferenciarlo de los demás, tiene por extensión *.lang*.

3.3.2. Entidades

La definición de una entidad comienza creando un nuevo objeto de la clase *Entity*, el cuál es proveído por SCE (en una especie de API para crear nuevos lenguajes). Se le debe dar un identificador único, el cuál será el tipo, ya que el usuario puede poner varias entidades del mismo tipo, por ejemplo *say* de la figura 3.3.

También se deben especificar las propiedades que podrá tener una entidad del tipo que se está definiendo acá. Para ello se crea un objeto de la clase *EntityProperty* del SCE, en el cuál se deben especificar el identificador de la propiedad, su nombre, si generará nuevas flechas, si se requiere que el usuario forzosamente especifique su valor, su multiplicidad, mediante

expresiones regulares qué cadenas se aceptan y cuáles no, y valor por omisión que tendrá. Éste objeto después se debe ingresar en la lista de propiedades del objeto *Entity* que se estaba creando.

3.3.3. Diálogos

Los diálogos son clases SWING de JAVA, que extienden de *LangDialog*, cuya finalidad es permitir al usuario editar las propiedades de las entidades al hacer doble click sobre ellas. Son ventanas modales que vienen, por defecto, con un campo para editar el nombre de la entidad y botones para aceptar o cancelar la acción, lo demás lo define el creador del lenguaje.
Para permitir el cambio de los valores de las propiedades a través de esta ventana, SCE provee dos métodos, el primero *initialize* con el cuál se carga el estado previo de las propiedades al diálogo, que dentro de éste método debe usar esa información para llenar los diferentes componentes de la ventana como *checkboxes* o *textfields* donde el usuario interactuará. El segundo es donde, luego de que el usuario acepte en el diálogo, se guardan los valores de las propiedades y, se transmiten al SCE para ser guardadas a través de un objeto especialmente diseñado para esta tarea, el cuál es *Element*.

3.3.4. Plantilla de transformación

Solamente diálogos, propiedades, entidades y conexiones no son útiles para el usuario. Lo que se busca al hacer una aplicación en SCE es un resultado concreto, que el flujo que se ve en pantalla mediante iconos y flechas se traduzca a, por ejemplo, una aplicación telefónica lista para ser ejecutada en algún intérprete CCXML/VXML. Para obtener tal producto es necesaria, en el SCE, una plantilla de transformación, la cuál puede ser escrita en el lenguaje que desee el programador del lenguaje, pero que siempre debe tomar como entrada el XML de la aplicación SCE.
En el caso del lenguaje para crear aplicaciones telefónicas, la plantilla de transformación esta escrita en XSLT, un lenguaje XML especial para la conversión de códigos XML en textos, HTML, o incluso código de otros lenguajes, como en este caso CCXML/VXML.

Las aplicaciones son procesadas en una parte especial del SCE llamada **generador**, el

cuál toma la plantilla XSLT del lenguaje (en el caso del lenguaje para hacer aplicaciones telefónicas) y transforma el XML de la aplicación en curso con ella, produciendo así, por ejemplo, archivos CCXML/VXML o, en el caso de lo que se hará durante el trabajo de título, los archivos de configuración de los servicios creados y usados por Sixbell.
En el futuro se espera que este proceso de transformación no lo haga el SCE por defecto, sino que sea relegado al programador la responsabilidad de hacer el generador a través del sistema de plug-ins de SCE. De esta manera se da más libertad al programador de nuevos lenguajes para elegir cómo transformar el XML de la aplicación, y mayor flexibilidad en la generación de los productos que en estos momentos está fuertemente ligada a la creación de archivos CCXML/VXML.

3.4. Plug -In

El SCE permite extender su poder y flexibilidad en la creación de lenguajes mediante la posibilidad de agregar más funcionalidades a través de *Plug-ins*.
Un *plug-in* es una aplicación JAVA prácticamente independiente del SCE, que se comunica a través de una interfaz común. A través de ella es capaz de modificar la aplicación, o también, hacer la generación de archivos a partir de ella con el uso de una plantilla de transformación.

La interfaz común es proveída por la clase *CoreInterface*, de la cuál, todos los *plug-ins* deben extender para ser llamados a través del único método a implementar que es *init*. El método, al ser llamado por el SCE, provee por medio de sus parámetros la aplicación, gracias al objeto *ApplicationGUI*, y del lenguaje que se está usando, usando el objeto *Language*.

Para el trabajo de título se creará un *plug-in* que generará a partir de una aplicación del SCE los archivos de configuración de los servicios de Sixbell, usando una plantilla de transformación XSLT, dado que en la empresa se tiene experiencia usando ese lenguaje y se puede usar parte del código de la generación de CCXML y VXML.

Capítulo 4

Problemas detectados

Antes de empezar a crear el nuevo lenguaje para el SCE, es necesario analizar posibles limitaciones del software de Sixbell para realizar la tarea encomendada en este trabajo de título.
Hay que notar acá que, desde sus inicios, el SCE fue creado para la creación de aplicaciones telefónicas, por lo que muchas de sus características aun siguen ligadas a esa función. A medida que salieron nuevas versiones se vio que tenía potencial para cumplir otras funciones y se fue separando la interfaz de lo que se llamó ahora el lenguaje.

Entre las principales limitaciones se puede encontrar que el usuario, a través de las flechas, solo puede ver un flujo, un grafo dirigido y, para lograr hacer el configurador de los sistemas se necesita hacer solamente un diagrama, donde no tiene sentido una flecha, pero si una conexión entre dos entidades.
Por otra parte, debido a constantes cambios y el tiempo que lleva evolucionando la aplicación, se nota desgaste en el diseño y ciertas partes que aun están ligadas a la generación de aplicaciones telefónicas, impidiendo la generalización del SCE que se busca con el configurador.

4.1. Flechas

Las flechas tienen mucho sentido cuando se trata de un flujo, como en las aplicaciones telefónicas, en donde es natural el modelo pues CCXML trabaja con estados y transiciones. En un diagrama de configuración no tiene sentido una flecha, mas si una conexión entre dos entidades, pero sin un sentido.

En el SCE las flechas están arraigadas profundamente en el código, siendo la única posibilidad que tiene el programador al hacer un nuevo lenguaje. La clase responsable es *LinkGUI*, tiene 1333 lineas lo que hace un poco complicada su modificación e indtica problemas serios de diseño y extensibilidad. Lo anterior será tratado más adelante en la subsección de diseño.

Lo que se propone es un cambio en el sistema de enlaces entre entidades, que permita al programador de nuevos lenguajes seleccionar entre distintos tipos de conexiones, en donde se debe incluir, por supuesto, la flecha que ya usa el lenguaje de aplicaciones telefónicas. Con esto se solucionará la limitante gráfica para el configurador y, de paso, se podrá mejorar el diseño orientado a objetos del SCE.

4.2. Diseño y extensibilidad

Hasta el momento el SCE nunca se ha enfrentado a la creación de otro lenguaje, solo se ha desarrollado su función con aplicaciones telefónicas y esta será su primera prueba de cuán flexible y cuán posible es la creación de otros lenguajes. Las falencias detectadas hasta el momento, antes del desarrollo del configurador son expuestas a continuación.

4.2.1. Generador

Aun quedan partes atadas a la producción de CCXML/VXML, como las antes mencionadas flechas, pero en puntos más serios. Es el caso del generador de ese lenguaje, que está puesto no como *plug-in* sino que como característica del SCE propiamente tal. Para este punto, en conversación con los ingenieros de Sixbell se acordó su eliminación del SCE y transformación a *plug-in* de modo de generalizar la herramienta y hacerla apta para el nuevo lenguaje.

4.2.2. Cambios de nombres

Cuando se cambia un nombre, la acción no tiene mayor repercusión usando el lenguaje de aplicaciones telefónicas. Para el configurador los nombres son parte critica pues muchas propiedades dependerán de ellos, especialmente la parte que tiene que ver con modelar las rutas de señalización de telefonía para protocolo ISUP a través del Stack SS7, en donde se tendrán que relacionar varias futuras entidades y cualquier cambio de nombre puede producir un montón de vicios al escribirse los valores en el XML de guardado. Esto, dado que los únicos momentos en que se hacen cambios en los valores de las propiedades son, cuando se abre el diálogo de la entidad y cuando se cierra, lo que hace que ante cualquier cambio no se pueda crear código que pueda reaccionar a tal evento.

Se propone para este punto agregar un sistema de eventos, con el cual, el programador del nuevo lenguaje pueda escribir sus propias clases que puedan hacer frente a los cambios, sin tener que modificar el SCE. Lo mismo aplica para desconexiones de entidades, y otros eventos que puedan causar mal comportamiento del SCE.

4.2.3. Análisis de complejidad del sistema

Dado que se necesitan hacer varios cambios en el diseño y código del SCE para que el lenguaje del configurador sea exitoso, se han hecho algunas mediciones para ver la dificultad que estos trabajos tienen y cuánto impacto tendrán finalmente sobre el producto.

El primer análisis es el de complejidad del sistema, que fue realizado con *Moose* una herramienta de análisis de software bastante completa. En la figura 4.1 se puede apreciar el resultado del examen hecho por Moose sobre SCE, en donde se identifican de inmediato tres rectángulos negros indicando donde se concentra la mayor parte de las lineas de código, así como también la falta de estructura de muchas de las clases, aunque varias de ellas extienden de paquetes externos y cumplen ciertos patrones de diseño.

Las clases que tienen mayor peso en cuanto lineas de código son : *SubAppGUI*, *LinkGUI* y *SubAppGUIListener*. La primera, con más de 1300 lineas de código, tiene que ver con el

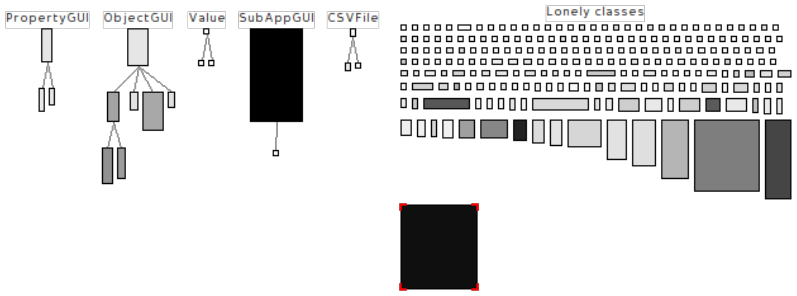

Figura 4.1: Complejidad del sistema visualizada en Moose. Las clases de representan con un recuadro, cuyo color indica las lineas de código, ancho el número de atributos y largo el número de métodos.

modelo detrás de las ventanas de las subaplicaciones, que manejan y contienen las entidades que dentro de ellas están. La última, con 913 lineas, va fuertemente ligada a la anterior, dado que es la encargada de conectar tal modelo con las acciones que tienen el usuario sobre tales ventanas. Y *LinkGUI*, con 1321 lineas, es la clase que maneja las conexiones entre entidades y el dibujado de las fechas sobre las ventanas y que se debe cambiar.
El que esté toda esta funcionalidad sobre estas 3 clases hace necesario cambiar el diseño de la aplicación para permitir los cambios que se necesitan, pero por sobre todo, para hacer del SCE un software que pueda seguir siendo mantenido y siga evolucionando a una herramienta flexible y multipropósito que tenga valor por si misma, independiente del lenguaje.

También es necesario ver cuán acopladas están las clases y paquetes entre si, siempre pensando en la posibilidad de flexibilizar la herramienta y permitir los cambios necesarios. Para ello es que *Moose* también tiene una herramienta, el DSM, que entrega un diagrama donde muestra, con color rojo, los paquetes fuertemente acoplados con otros.
En la figura 4.2 se puede ver cómo el paquete SCE (donde se encuentra la clase SCE, portadora del método *main* que ejecuta la aplicación), marcado con esquinas rojas, es el más acoplado puesto que cumple con el patrón de diseño *Singleton* y es usado en prácticamente todas las clases del SCE. Su sobreutilización puede ser un mal indicio y puede que complique las cosas al tratar de modificar el SCE y corregir su diseño.
Otros paquetes acoplados son el de las acciones (según patrón Command) donde se juntan todos los comportamientos del SCE, desde la apertura de una aplicación hasta el copiado, pegado y eliminación de entidades. También el del modelo donde se encuentran las clases que

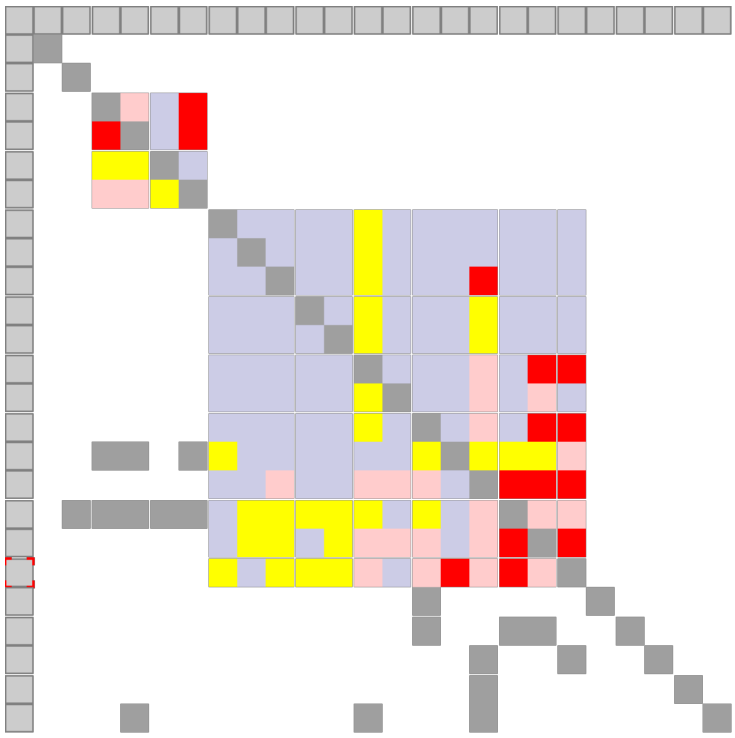

Figura 4.2: Diagrama de acoplamiento entre paquetes(matriz de todos versus todos), mientras más rojo se presente un cuadrado esta más acoplado con el paquete representante de la columna.

soportan internamente los datos de la aplicación y entidades.

Capítulo 5

SCE 3.1

Durante el desarrollo del nuevo lenguaje de configuración de sistemas, fue necesario hacer gran cantidad de cambios de manera de aumentar la extensibilidad y flexibilidad del SCE camino a una generalización de la herramienta, quitándole las amarras al lenguaje de creación de aplicaciones telefónicas, finalidad que se le dio desde sus inicios.

Cada nueva funcionalidad y cada rediseño que se hizo sobre el SCE, se hizo no solo pensando en la configuración de sistemas sino que también en que éste programa pudiera ser usado para resolver cualquier otro problema que tenga que ver con diagramas o flujo. Es por esto que se hizo un completo análisis de la estructura de clases y se modificaron puntos sensibles, como el dibujado de las conexiones y entidades, la eliminación de los llamados çodos"(entidad especial creada para ordenar las flechas) por un sistema de enlaces que se pueden moldear (usado en varios programas de diagramas), y comportamientos modificables para hacer plugins más poderosos y que puedan interactuar de manera más cercana con la aplicación que está creando el usuario.

El resultado esperado de todo este trabajo es un IDE preparado para aceptar las exigencias del nuevo lenguaje de configuración, y también para nuevos problemas que se le pongan en el futuro, haciendo que el SCE sea un producto por si mismo utilizable por otras empresas.

5.1. Nuevas funcionalidades

5.1.1. Bibliotecas

Frecuentemente los usuarios de SCE para aplicaciones telefónicas se encontraban copiando y pegando entidades de una aplicación a otra. Muchas funcionalidades se repetían de aplicación en aplicación y el SCE no ofrecía un mecanismo para poder aprovechar secciones de aplicaciones antiguas, más que el copiado y pegado.

Por otra parte, se notó que los contenedores podían servir como encapsuladores de nuevas funcionalidades, dado que, para alguien externo que no ha creado la aplicación, lucen como cualquier otra entidad. La desventaja que poseen es que no tienen una imagen distintiva que haga a un contenedor diferenciarse de los demás.

Es por esto que se hizo una extensión de los contenedores, dotándolos de nuevas funcionalidades, y la creación de una nueva estructura llamada biblioteca, la cuál puede albergar contenedores y además es posible guardarla y compartirla con otros usuarios del SCE que estén usando el mismo lenguaje. Así, cualquier funcionalidad que tenga potencial de ser reutilizada se puede crear dentro de un contenedor para posteriormente ser guardada en una biblioteca y luego ser compartida con los demás usuarios.

Creación de bibliotecas

Para el usuario, desde ahora en adelante siempre existirá una biblioteca llamada *Default*, la cuál se puede apreciar en la esquina inferior izquierda de la pantalla, como se ve en la figura 5.1. Esto es solo para que siempre el usuario sea capaz de guardar contenedores, posteriormente podrá cambiarle el nombre y usarla como cualquier otra.

Para crear una nueva biblioteca el usuario tan solo debe hacer click en el icono de la barra de herramientas dispuesto para ello, luego aparecerá un diálogo preguntando por el nombre de la nueva biblioteca, tal como aparece en la figura 5.2.

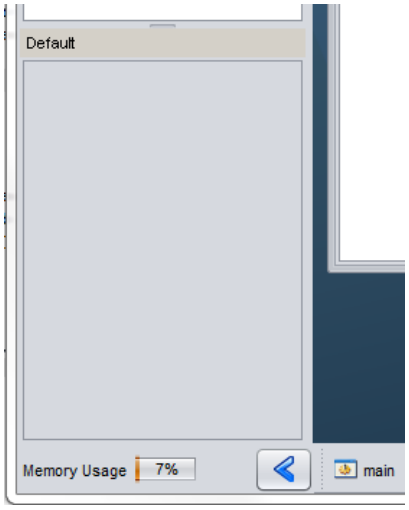

Figura 5.1: Vista de la biblioteca creada por defecto

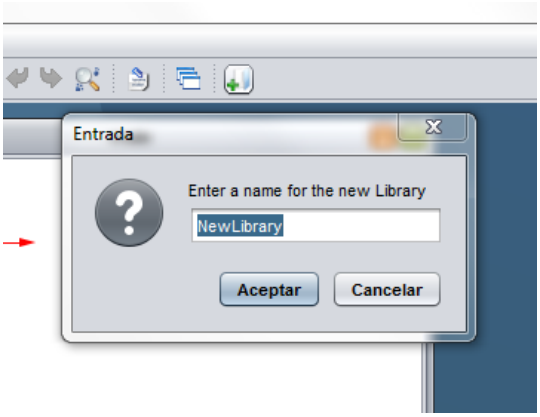

Figura 5.2: Diálogo para ingresar el nombre de la nueva biblioteca

Guardado de un Contenedor

Una vez creada una biblioteca, o incluso usando la que se crea por defecto, el usuario podrá guardar contenedores dentro de ella. La idea detrás de esto es que en los proyectos se puedan reusar funcionalidades encapsuladas en estos contenedores, en los cuales se guarda toda su estructura interna, y solo sus enlaces externos. Para guardar un contenedor, sólo se debe hacer click con el botón derecho sobre él y luego seleccionar *"Add container to library"*, el icono del contenedor ahora aparecerá en la biblioteca actual, tal como aparece en la figura 5.3.

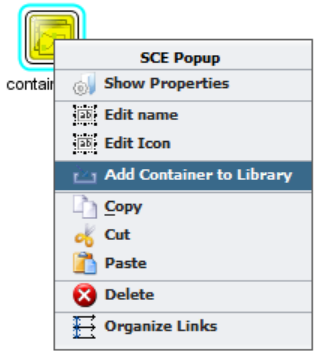

Figura 5.3: Menú del contenedor y opción para guardarlo en una biblioteca

Características nuevas del contenedor

Internamente el contenedor se cambió para poder soportar esta nueva característica, desde ahora soporta cambio en su imagen distintiva, cosa de hacer más fácil el uso del sistema de bibliotecas. Con este cambio, el usuario puede especificar cualquier imagen para el contenedor, y esta será guardada para la aplicación, aunque el contenedor no sea añadido a una biblioteca.

También se hizo otra clase contenedor para soportar el guardado de los elementos interiores de una manera coherente para el uso de las bibliotecas, esto es, que aparte de que se puedan crear y llenar las bibliotecas de contenedores, también otro usuario pueda hacer uso de éstas y que al colocarlas dentro de alguna aplicación, ésta última pueda ser abierta en otro SCE, a pesar de no tener la biblioteca desde donde se extrajo el contenedor. Análogamente también hay una clase biblioteca para mantener a estos nuevos contenedores en su lugar y permitir una serialización ordenada.

5.1.2. Enlaces

Los enlaces son parte importante de las aplicaciones del SCE, son los que definen el flujo de las aplicaciones y permiten apreciar al usuario fácilmente lo que hará su programa.

Los diferentes cambios hechos apuntan a flexibilizar el uso de enlaces para diferentes tipos de aplicaciones, ya no solo telefónicas, apuntando a una generalización del SCE, asi como también proveer de mejoras al usuario para que el uso de SCE sea más fácil e intuitivo. A continuación se explicarán las nuevas características de los enlaces, así como también aspectos técnicos que fueron corregidos internamente para facilitar el desarrollo de otros lenguajes y hacer el mantenimiento del SCE más llevadero.

Nodos

Uno de los cambios más importantes a nivel de usuario es la introducción de los nodos en los enlaces. En la versión anterior del SCE, el desarrollador tenía que usar una entidad especial (que no tenía efecto sobre la aplicación final) llamada *Codo* cada vez que quería quebrar el enlace en dos, para mantener ordenado el diagrama. Al final con lo anterior la aplicación se empezaba a ensuciar rápidamente con entidades sin significado y que agregaban ruido al ser interpretado el diagrama para su transformación a los archivos del programa final.

La solución se hizo sobre los mismos enlaces, los que ahora son un conjunto de nodos que marcan la trayectoria del enlace. Un enlace común y corriente, como los de la versión anterior, consta de tan solo dos nodos, el de inicio y el final, y el usuario puede seguir agregando tantos nodos necesite para demarcar la linea que representa el enlace, como se puede ver en la figura 5.4.

Para crear nodos, se le ha facilitado la tarea al usuario permitiéndole quebrar el enlace en cualquier parte, simplemente arrastrando la linea. De la misma forma se puede cambiar un nodo de posición o incluso borrarlo, arrastrándolo hasta que forme una linea recta con otros dos nodos. También se pueden borrar nodos haciendo click con el botón derecho y luego seleccionando la opción *"Delete node"*.

Inicio y final de flecha intercambiables

En la versión anterior un enlace era una linea que empezaba en una entidad y terminaba en una punta de flecha, eso para cualquier lenguaje. Tal limitación hacía que un lenguaje

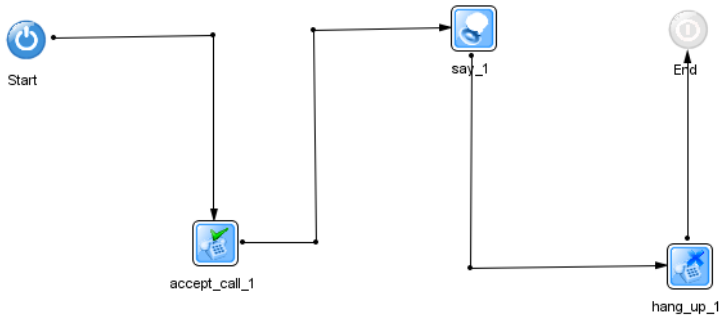

Figura 5.4: Enlaces quebrados por nodos, los cuales se representan como pequeños puntos negros.

como el que se creó para configurar los sistemas telefónicos fuera imposible, dado que en ese caso se hace un diagrama donde la flecha no tiene sentido, pues solo se quiere representar la unión de dos entidades y no un flujo.

El cambio que se hizo fue agregar la posibilidad, a los desarrolladores de lenguajes para el SCE, de definir para cada lenguaje cuál será la clase que se encargue de dibujar el inicio y el final del enlace, pudiendo elegir del stock de clases (en donde se encuentra la punta de flecha, punta redondeada, sin punta) o crear su propia clase dibujadora, extendiendo de la interfaz proporcionada por la biblioteca SCE.

Para este cambio se usó el patrón de diseño *Strategy*, logrando ordenar el código responsable de los enlaces, el cuál estaba condensado en tan solo una clase llamada *LinkGUI*. Ahora el dibujado se ha puesto en clases externas, contribuyendo a que la clase sea más extensible y mantenible en el tiempo.

Linea intercambiable

Así como el inicio y final de un enlace pueden ser intercambiados, el cómo se dibuja la linea también, pero acá la responsabilidad de definir el estilo de linea a ocupar pasa al usuario, el cuál puede en cualquier momento hacer click con el botón derecho sobre la linea y elegir el estilo que desea, tal como se muestra en la figura 5.5.

Internamente este comportamiento es manejado de similar forma que los inicios y finales

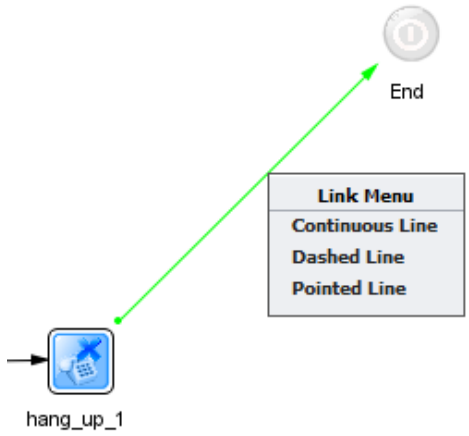

Figura 5.5: Menú de selección de estilo de linea para los enlaces.

de enlace intercambiables, solo que acá se agregan los elementos para que el usuario pueda también hacer sus modificaciones. Hay que señalar también que los lenguajes pueden definir sus propios estilos de linea y especificar cuál de todos se usará por defecto.

5.1.3. Correctores de aplicaciones antiguas

Debido a los múltiples cambios y arreglos que se han hecho en el SCE, y también por la mantención y cambios que se van haciendo a los lenguajes a través del tiempo, es que se hace necesario un mecanismo para asegurar la retrocompatibilidad de las aplicaciones creadas con el SCE.

Ahora el desarrollador de un lenguaje, en cada actualización puede agregar una clase especial que extienda de *AppFixer* con la cuál, mediante el método *fixOlderVersions*, recibe la aplicación y puede modificarla a su antojo. La idea es que se compruebe dentro de este método algún problema de compatibilidad y se corrija solo en caso de ser necesario, puesto que estas clases se ejecutarán en cada carga de la aplicación.

Ejemplo de ello ha sido la remoción de algunas entidades debido a que su función se ha incluido en alguna otra entidad, por lo que a través de estas clases correctoras se puede extraer las entidades el tipo que ya no existe y poner sus datos en las nuevas entidades, actualizando la aplicación del usuario a las nuevas versiones del SCE.

5.1.4. Comportamientos reemplazables

Para mejorar la extensibilidad del SCE, y permitir la creación de plug-ins que acompañen a cada lenguaje, es que ahora el comportamiento de las entidades puede ser modificado.
El desarrollador del lenguaje puede ahora extender desde *MouseObjectGUIBehaviour*, interfaz que da la posibilidad de cambiar cómo la entidad o el objeto gráfico (como los comentarios, codos, atajos, etc...) debe responder si recibe un click, un doble click, un click derecho, si se posan encima, si entran en su espacio, si salen de su área o incluso como actuar frente a un arrastre con el mouse. Luego tal comportamiento simplemente debe introducirlo a cualquier clase que extienda de *ObjectGUI* mediante el método *setMouseObjectGUIBehaviour*.
Es así que, por ejemplo, un plug-in podría desactivar todo evento del mouse en el SCE para así entrar en un modo de debug y que el usuario sea capaz solamente de ver cómo la llamada va progresando (en el lenguaje de aplicaciones telefónicas), sin posibilidad de modificar las entidades.

5.1.5. Dibujadores reemplazables

Con el mismo fin de los comportamientos reemplazables, el desarrollador ahora es capaz de modificar el cómo se dibujarán las entidades y objetos gráficos del SCE, dando la posibilidad, por ejemplo, de colorear las entidades o incluso reemplazarlas por otros iconos o por último simplemente no dibujarlas.
El desarrollador de lenguajes debe crear una clase que extienda a *DefaultObjectGUIDrawer*, luego para reemplazar el dibujado, el único método que debe sobreescribir es *draw*, el cual recibe el objeto *Graphics2D* preparado para dibujar.
Hay que notar acá, que esta característica en combinación con la de los comportamientos reemplazables puede ser usada para crear plug-ins realmente poderosos, pudiendo hacer, por ejemplo, entidades que puedan cambiar de color según las acciones que el usuario haga, o implementar estados simbolizados gráficamente y con comportamientos distintos según esos estados.

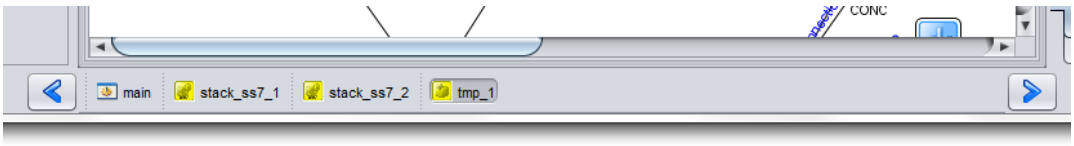

Figura 5.6: Barra de ventanas.

5.1.6. Barra de tareas

Para aplicaciones muy grandes, el estar cambiando de ventanas se hacía molesto y a veces muy complicado para los usuarios. Por tal razón se creó una barra de tareas con todas las ventanas interiores, la cuál se puede ver en la figura 5.6.

En la barra de tareas el usuario puede ver todos los nombres de las ventanas de la aplicación, con botones para cada una que muestran el nombre del contenedor al que referencian y el icono que las distingue. También se puede notar la ventana activa, mostrada como un botón presionado.

En caso de que hayan tantas ventanas que ya no se pueden mostrar en la barra, existen dos botones laterales que permiten desplazar la barra de tareas para ver las otras ventanas disponibles.

5.2. Diseño

En el diseño de la aplicación, el SCE tenía demasiados problemas. Esto principalmente por la evolución que tuvo el producto durante su vida antes de que llegara la idea de implementar el configurador de sistemas. Desde su versión 1 se ha pensado el SCE como un entorno para desarrollar aplicaciones telefónicas que siguen un flujo, por ello es que el SCE 3.0 solo tenía flechas como enlaces y siempre en todas sus aplicaciones aparecían dos entidades, una de comienzo y otra de final para marcar el inicio y termino de la aplicación telefónica, y también del flujo.

A medida que pasó el tiempo y nuevos requerimientos comenzaron a gestarse, la aplicación fue creciendo sin un diseño claro, sino que más bien una programación para lograr objetivos al corto y mediano plazo. No fue hasta la versión 3 del SCE en que se pensó que este en-

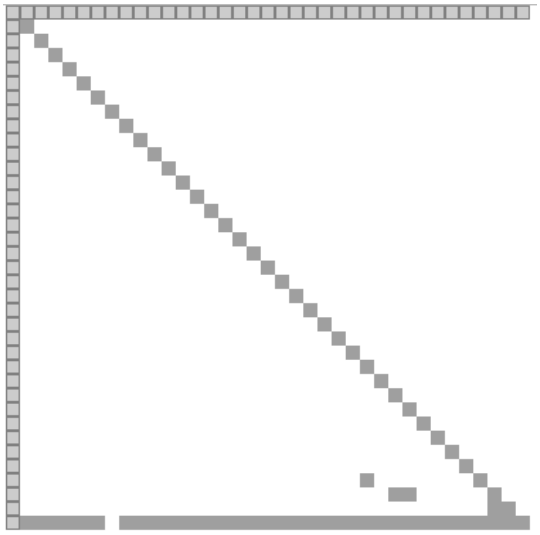

Figura 5.7: Diagrama de acoplamiento entre paquetes de la nueva versión del SCE.

torno gráfico podía ser usado para otros fines y con ello se separó lo que es una aplicación y su diagrama, del resultado que se quería obtener y nacieron los llamados lenguajes y plug-ins.

Es en esta versión 3.1 en que esa idea se consolida y se piensa ya en el SCE como un entorno independiente del lenguaje que se quiera usar. Se comienza a desarrollar en pos de una aplicación que permita ser usada en distintos casos de uso y que sea extremadamente flexible para convertirse en un producto por si sola y que las empresas creen sus propios lenguajes para resolver problemas internos que tengan.

El diseño de la aplicación ahora tiene ese sentido y es de donde se soportan todas las nuevas características mostradas anteriormente, y, gracias a los ingenieros de Sixbell que apoyaron el proyecto, se consiguió una total reestructuración de los paquetes y clases del SCE, logrando el diagrama de acoplamiento de paquetes ahora luzca como sale en la figura 5.7. Aunque el diagrama puede que tenga fallas dada la nueva versión de la herramienta que se usó, pero aun así se muestra un gran avance con respecto al anterior diagrama obtenido por la versión SCE 3.0 .

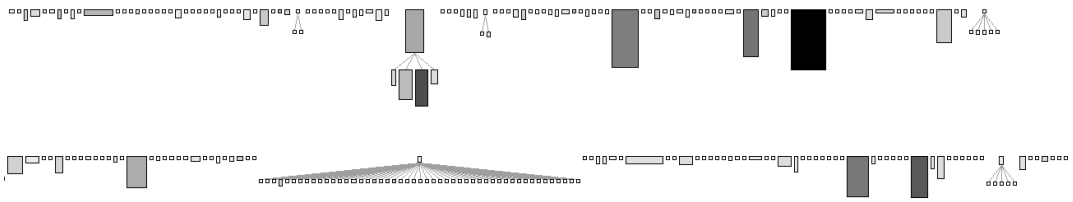

Figura 5.8: Análisis de complejidad sobre la nueva versión, donde las clases son mostradas como rectángulos, cuyo color es más negro si tiene más lineas de código, su ancho depende de los atributos que tiene y su alto del número de métodos.

5.2.1. Arquitectura de clases

Con respecto a la arquitectura de clases, profundos cambios se hicieron en diferentes clases para que estas pudieran reducir su cantidad de lineas y permitieran mayor flexibilidad a cambios futuros y a nuevos requerimientos a medida que surgieran nuevos problemas que pudieran ser modelados con el SCE. Al final se logró que el análisis de complejidad realizado al SCE 3.0 ahora muestre lo que aparece en la figura 5.8, con menos clases sobresalientes por su gran cantidad de lineas de código y se presenta una estructura con jerarquías más marcadas, dado que ahora se implementaron más patrones de diseño útiles para lograr el objetivo como el patrón *Strategy* usado en enlaces y entidades, tal como se verá en seguida en detalle por cada una de las principales clases involucradas.

LinkGUI

La clase LinkGUI es la que se encarga del dibujado de los enlaces y de su representación dentro del código. En la primera versión esta clase tenía 1333 lineas de código, las cuáles fueron ahora reducidas a 1000, pero con código más limpio y muchos más comentarios.
Además se aplicó un patrón *Strategy* y *Factory* para lograr desacoplar lo que tiene que ver con el dibujado del enlace, de lo que es su representación en el modelo. Es así que ahora los programadores de lenguajes para el SCE pueden hacer sus propios dibujadores de enlaces, o pueden definir su dibujador preferido de los que hay en stock. Para crear un nuevo dibujador solo deben extender de la interfaz ObjectGUIDrawer.

En la misma linea se hizo la división entre lo que es un dibujador del enlace en si y lo

que es un dibujador de sus puntas, por lo que también existe una interfaz y estrategias para dibujar finales de enlaces.

ObjectGUI

ObjectGUI es el padre de todos los objetos gráficos, como entidades, comentarios, codos y atajos. Como tal, encerraba mucho código que después no iba a ser usado, especialmente respecto a los eventos del mouse que recibía y cómo debía actuar frente a ellos. Para cambiar esto se hizo una limpieza de código, sumado a un refactor de los comportamiento separándolos de lo que es el modelo y permitiendo que el programador de un plug-in o de un lenguaje pueda cambiar el comportamiento de los objetos gráficos, agregando mucho poder y flexibilidad al SCE.

EventSystem

Se agregó un sistema de eventos mediante el patrón de diseño *Observer*, para que así siempre que un elemento se elimine, agregue, o se quite la conexión de alguno de sus enlaces notifique a clases que el programador del lenguaje puede hacer, de manera de mantener la coherencia con características o parámetros de una entidad que se vean afectados por tales acciones del usuario.
El sistema de eventos tiene un controlador central, donde se agregan los manejadores de eventos, e interfaces para cada una de las acciones soportadas hasta el momento para que tengan manejadores.

AppFixer

De manera de mantener la retrocompatibilidad se creó esta interfaz y se agregó en el cargador de aplicaciones código para ejecutar limpiadores de aplicaciones antiguas, las cuales el programador del lenguaje del SCE puede crear y ejecutar en serie. Cada una de estas clases recibe la aplicación completa del usuario y le permite modificarla a su antojo de manera de dejarla lista para la última versión del lenguaje en caso de que hayan cambios que entren en conflicto con versiones anteriores.

Actions

Muchas de las acciones del SCE (como cortar, copiar, eliminar) dependen del tipo de objeto gráfico con el que trabajar. Por ello es que antes se usaba mucho el método *instanceof* de JAVA para averiguar con qué clase se estaba intentando trabajar, teniendo código muy confuso muchas veces y muy poco extensible. Para solucionar esto se implementó el patrón *Visitor* combinado con sobrecarga de métodos para lograr que desde la clase abstracta AbstractSCEAction existan métodos separados para cada tipo de objeto gráfico, permitiendo que cada clase que herede de ella pueda escribir en estos métodos cómo reaccionará en caso de que reciba cierto tipo de objeto gráfico.

Capítulo 6

Lenguaje de Configuración

Dado el gran poder del SCE para la creación de diagramas es que se pensó en hacer este trabajo, más aun con todos los cambios hechos en el SCE mismo. Por otro lado, la gran cantidad de archivos de configuración y alta complejidad de algunos de ellos hacían urgente una herramienta que facilitara la configuración de los sistemas que la empresa vende e instala. Y es por estas dos grandes razones que se decidió la creación de este lenguaje, al principio solo para parte de los sistemas, de forma experimental, luego al ver su potencial se extendió a la mayoría de ellos llegando ahora a ser la manera oficial de configuración, además de estar el SCE versión configurador embebido en la herramienta de monitoreo y gestión *Smilodon* que acompaña a cada proyecto de la empresa.

El lenguaje creado consta de dos partes fácilmente reconocibles, una interna que es la plantilla de transformación donde se procesa el diagrama que el usuario crea en el SCE y las entidades en si y sus diálogos, parte gráfica con la cual el usuario interactúa, que se tratarán por separado en las secciones que vienen a continuación. Además se agrega una sección más para el plug-in creado que acompaña a este lenguaje, desde el cuál se ejecuta la plantilla de transformación y se empaquetan los archivos resultantes, los que después repartirá el Smilodon entre las diferentes máquinas y servicios del sistema a configurar.

6.1. Plantilla de transformación

La plantilla de transformación, escrita en el lenguaje XSLT, es donde está toda la lógica detrás de los diagramas. Sus más de 5100 lineas contienen todo el conocimiento de los inge-

nieros de Sixbell sobre las configuraciones de los sistemas. Por esta razón es que se le da el apodo de *Sistema experto* al trabajo realizado, no porque pueda aprender más por si solo, sino que encierra toda la experiencia acumulada en proyectos anteriores de Sixbell. Ejemplo de esto es que se usaron configuraciones reales de centrales telefónicas como bases para la creación de la plantilla, y como parámetro de comparación para los resultados que durante el desarrollo se fueran obteniendo.

6.1.1. Estructura

La plantilla, para mantener el orden, sigue una estructura ordenada por *templates* que se aplican a cada una de las entidades que aparecen en la aplicación SCE que está creando el usuario. XSLT solo puede generar un archivo de texto por ejecución, y como se usa solo una plantilla se decidió seguir la convención de que cada archivo que se genere debe empezar con un header como el siguiente :

¶*nombre_de_archivo.conf*####################

De esta forma el programa que procesa la plantilla y empaqueta los archivos de configuración resultantes sabe donde empieza y termina cada archivo, además del nombre que debe usar al escribirlo a disco.

Internamente en la plantilla, aparte de todos los archivos que se generan por las entidades, crea otro archivo más al principio, llamado *conf_master.xml*, el cual lista todos los archivos que se generarán y extrae, según cada entidad, la información de la máquina y ruta a la que debe ser cargado, también posee los comandos que deben ejecutarse antes y después de copiar el archivo, de manera de que los servicios continúen su ejecución aplicando la nueva configuración. En la figura 6.1 se puede ver un ejemplo de uno de los elementos de la lista.

Además en el archivo conf_master.xml se agregan dos campos, con etiquetas llamadas *globalpre* y *globalpost*, que contienen los comandos que se deben ejecutar **siempre** antes y después de cada injección de archivo de configuración, en general son comandos que matan todo proceso que pueda dificultar el copiado de los archivos.

```
<config name="Control" ip="192.168.2.165" file="sigas-control226.conf"
    realname="sigas-control226.conf" route="~/control/">
        <pre>
                <command cmd="mkdir ~/control"/>
                <command cmd="ps ux | awk '/sigas-control226.conf/ &amp;&amp;
                    !/awk/ {print $2}' | xargs kill -9"/>
        </pre>
        <post>
                <command cmd="cd /opt/sixlabs/bin"/>
                <command cmd="nohup ./sigas-control2 -x ~/control/sigas-
                    control226.conf &gt; /dev/null &amp;"/>
        </post>
</config>
```

Figura 6.1: Elemento de la lista en el archivo conf_master.xml

6.2. Principales entidades

La sección que ve el usuario es la interfaz del SCE, es ahí donde se arma el diagrama a través de entidades y conexiones entre ellas, sin el cual la plantilla no tiene sentido. En el caso del configurador cada entidad modela un servicio, capa o sección del sistema telefónico a instalar, de forma que un operador, que conoce los elementos técnicos involucrados, pueda fácilmente identificar cada entidad y a través de sus diálogos llenar los datos necesarios, sin tener que recorrer la larga lista de archivos de configuración de todo el sistema.

A continuación se hará un recorrido por las principales entidades, donde se verá su icono distintivo, cómo se usan y en las más complejas, su diálogo donde el usuario ingresa los datos necesarios para la configuración.

6.2.1. Entidades generales

Bajo esta categoría se pueden encontrar las entidades que pertenecen al grupo de máquinas o programas de más alto nivel, desde la perspectiva de configuración, o las más comunes, tales como **ActiveMQ**. Es en esta sección donde se encuentran entidades contenedoras, como el **TMP** o el **Stack SS7** que se detallarán más adelante.

TMP

Para configurar un TMP se debe usar el contenedor que tiene el icono mostrado a la derecha, el cual se coloca en la ventana *main*. Dentro de él se colocan los diferentes componentes, tales como el control, tarjetas de *media* y diferentes conectores o capas que soportará el TMP.

En la sección **??** se explicarán todos los componentes de un TMP.

Stack SS7

Para colocar un Stack SS7 en el sistema a configurar, se debe usar el contenedor identificado con el icono de la derecha. Dentro de él se pueden colocar sus componentes modelados con las entidades clasificadas bajo su mismo nombre.

En la sección **??** se explicarán las diferentes entidades que pueden formar parte del Stack SS7.

MSC

La entidad MSC sirve para indicar un *Point Code* de destino. Como tal, puede recibir múltiples enlaces, ya sean provenientes de una entidad STP o desde un Stack SS7. Esta entidad solo puede ir en la ventana *main*.

STP

Mediante esta entidad se representa un STP, el cual puede ser conectado con otros STP de cualquier forma, así como también a varios MSC. En el diálogo que se despliega al hacer doble click sobre ella, se puede seleccionar el número de conexiones que tendrá.

En la figura 6.2 se muestra un ejemplo de una red con STP y MSC correctamente conectados.

Smilodon

Con esta entidad se representa al servicio Smilodon de la empresa, cuyo diálogo permite configurar la IP de la máquina y el servicio de mensajería ActiveMQ de ella.

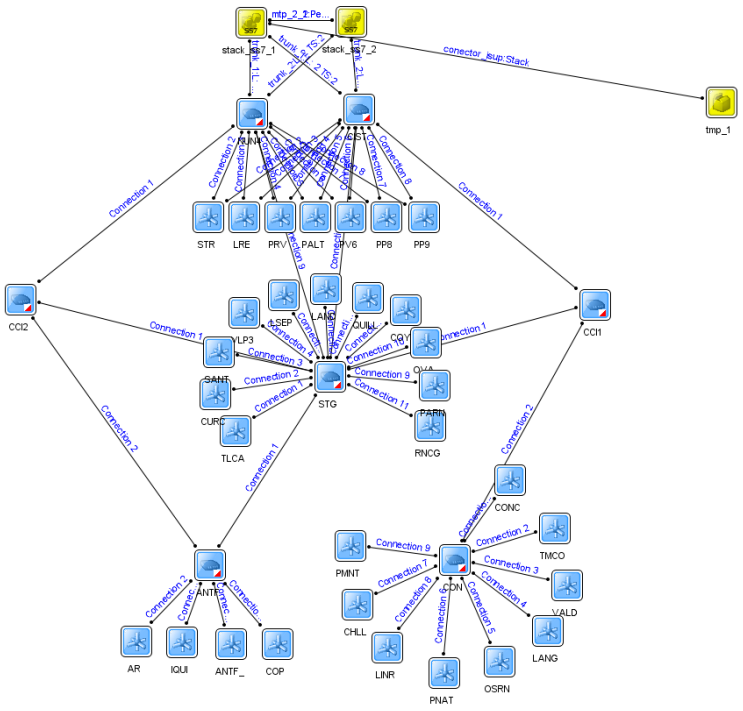

Figura 6.2: Ejemplo de una red de MSC y STP con Stack SS7 redundante modelada en el SCE-Configurador.

6.2.2. Entidades del TMP

En el SCE se clasifican las entidades por pestañas, desde donde se seleccionan según sus iconos distintivos. En la pestaña del TMP se encuentran todos sus componentes, donde el Control es la entidad principal. También se ha agregado a la sección del TMP la entidad Stack SIP, debido a su conexión casi directa con el Control (pero pasando por un conector SIP) y su fácil configuración. A continuación se detallan todas las entidades y cómo se relacionan entre si y con las demás.

Control

Es la entidad principal y representa al software Sigas Control, al cual se le conectan las demás entidades del TMP. La información que se introduce en su diálogo es muy importante, pues en la plantilla de transformación se usa para identificar dónde irán los archivos de configuración de las entidades conectadas a él.

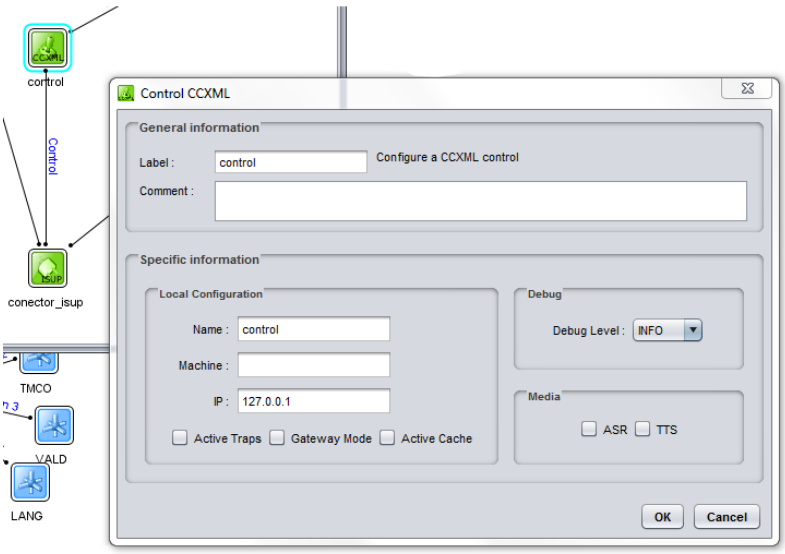

Figura 6.3: Diálogo de la entidad Control del TMP.

Su diálogo es simple y limpio, como se puede apreciar en la figura 6.3, y es muestra del poder del SCE en cuanto a la reducción de la complejidad del archivo de configuración en si del control, que tiene más de 40 parámetros configurables, ahora convertidos en no más de 9, ya que los demás se infieren o se extraen de la relación que tiene el control con las demás entidades.

Hay que notar también la flexibilidad que entrega el SCE en cuanto ésta entidad puede activar opciones como *"Active Traps"* que generarán un nuevo enlace para poder comunicar el control con el conector que provee de *Traps*.

Stack SIP

Solo basta con agregar esta entidad al TMP, ingresar sus direcciones UDP y TCP/IP y conectarla a un *Señalizador SIP* para configurar un Stack SIP. En la figura 6.4 se puede ver un ejemplo de un Stack SIP dentro de un TMP.

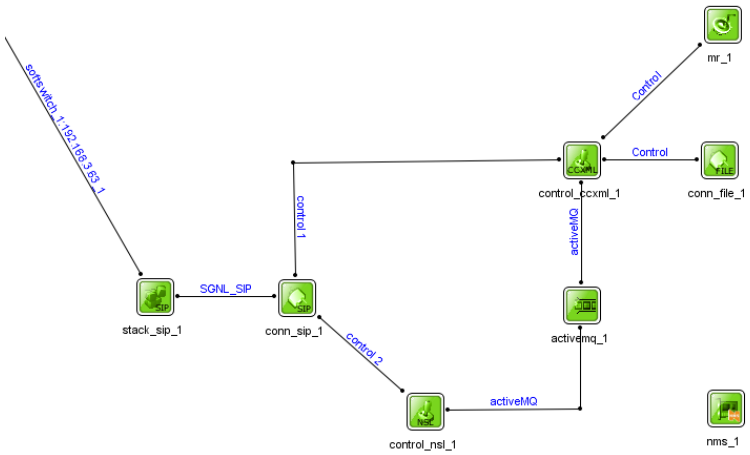

Figura 6.4: Ejemplo de configuración de un Stack SIP.

Señalizador SIP

Permite la conexión del TMP con un Stack SIP, además su ventana de configuración permite la conexión de este conector con varios controles.

Conector ISUP

Permite la conexión del TMP con un Stack SS7, sin requerir más configuración que enlazarlo con la entidad que representa la capa ISUP del Stack, y por supuesto, con el control respectivo que recibirá los eventos de este conector.
En la figura 6.5 se puede ver un ejemplo de configuración de un Stack SS7 con ISUP, conectado a un TMP.

Media Resource HMP

Con esta entidad se representa a un Media Resource HMP que necesita solamente ser conectada a un control. En su diálogo se pueden establecer los parámetros básicos de configuración, tales como los codecs, las rutas a archivos de audio

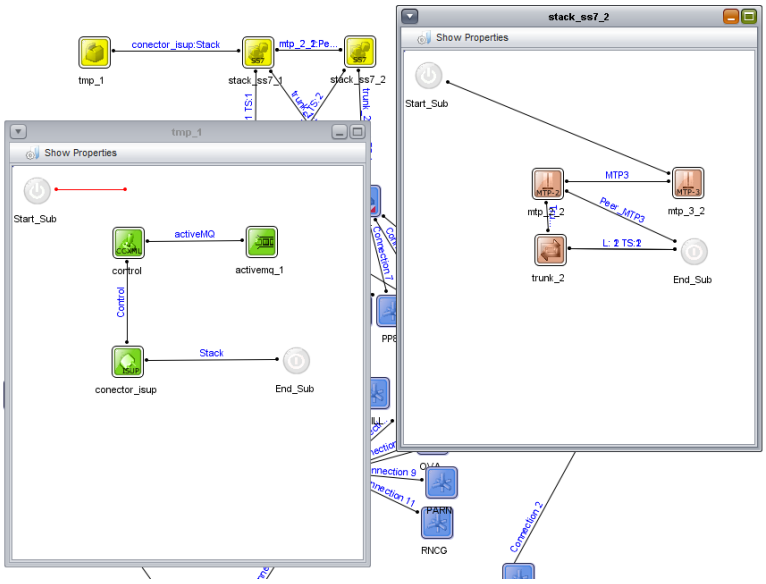

Figura 6.5: Ejemplo de configuración de un Stack SS7 con ISUP conectado a un TMP.

que usará, entre otras opciones.

Para configurar correctamente los codecs, éstos deben agregarse a la lista que aparece en el diálogo en orden, donde el primero será el de mayor prioridad.

NMS

Una de las entidades más complejas, pues la configuración de una de estas tarjetas es sumamente compleja y requiere de gran conocimiento de cómo funciona a bajo nivel. Estas tarjetas se ocupan de reproducir y grabar voz, detectar y generar tonos, cancelar eco, compresión de voz, procesar conferencias, fax y VoIP.

En su diálogo se pueden configurar la IP que usará, información de la tarjeta y puertos que ocupa en el servidor más datos de bajo nivel, los relojes, los enlaces E1 que recibe y si soporta conferencia y cancelación de eco. También posee una pestaña especial para ingresar código propio, donde de todas maneras se guía al usuario, quien puede usar templates o generar una configuración a partir de los datos ya ingresados. Se puede ver ésta última pestaña en la figura 6.6

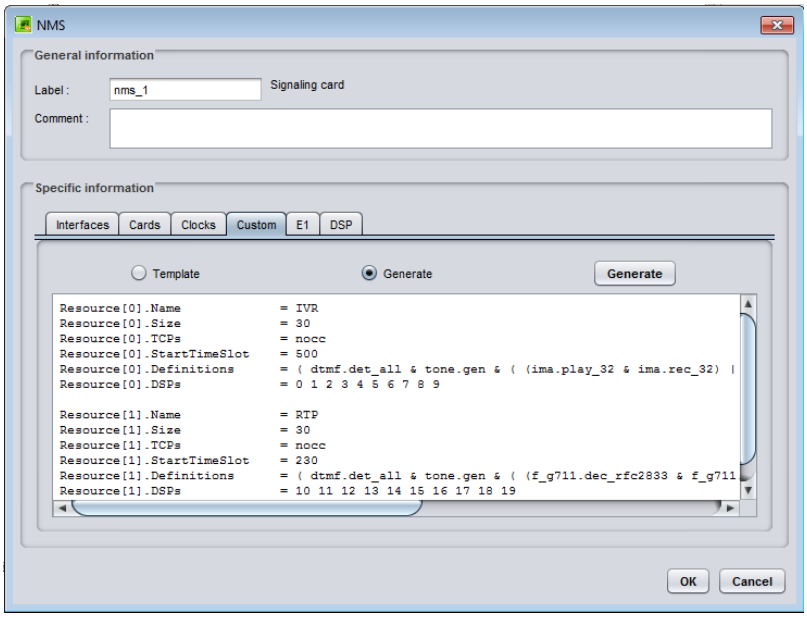

Figura 6.6: Diálogo de la entidad NMS, mostrando la pestaña para ingresar código propio.

6.2.3. Stack SS7

Para configurar un Stack SS7 se han separado sus capas en entidades, de manera de facilitar la configuración. Además, en los diálogos de pueden especificar fácilmente todos los parámetros correspondientes a links, timeslots, grupos de circuitos, incluso la prioridad con que se eligen los STP por los que se debe cruzar hasta las centrales.

MTP 2

Para la capa MTP 2 se ha dispuesto un diálogo muy completo de configuración, en el cual se pueden especificar lo que se guardará en el log, también la ruta para acceder a este, los timers, el tipo de comunicación que tendrá, el número de trunks a usar y si tendrá un Stack SS7 de redundancia.

Al elegir el número de trunks a conectar, desde esta entidad saldrán los enlaces para unir los trunks. Tales enlaces pueden o no ser usados, ya que en caso de ser dejados sin conectar, en el archivo de configuración generado se pondrá como desactivado tal trunk (se debe tomar en cuenta la numeración de los enlaces, pues indican el trunk que se usará).

En la figura 6.7 se observa a una entidad MTP 2 que tiene espacio para 4 trunks.
Por otra parte, la entidad MTP 2 posee otro enlace, que posee la etiqueta "MTP 3" el cual deberá unir con la entidad MTP 3 del Stack y, si se ha seleccionado la opción de MTP 3 redundante, se dispondrá de un enlace adicional para conectarlo al MTP 3 del Stack redundante.

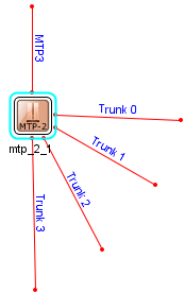

Figura 6.7: Enlaces para conectar trunks saliendo de la entidad MTP 2.

MTP 3

En el caso de la entidad MTP 3, la ventana de configuración permite seleccionar el *Point Code* y la IP del Stack SS7, además de ciertos parámetros de log y concernientes a las conexiones que manejará.
Tamobién se puede manejar la prioridad de los STP adyacentes a usar para conectarse a cada uno de los MSC que alcanza el Stack SS7 en el cuál se encuentra esta entidad.

Trunk

Esta entidad representa a una de las bocas que puede tener una tarjeta de señalización conectada a la máquina donde corre la capa MTP 2 del Stack.
En su diálogo de configuración se deben especificar el tipo de tarjeta en la que se encuentra, el dispositivo UNIX que se ocupará y otros parámetros necesarios. Por otra parte, existe una tabla en la que se pueden agregar o quitar links, quienes son los que finalmente llegarán a los STP o centrales (MSC). Es muy importante acá elegir bien el tipo de trunk usado (E1 o T1) ya que se modificará el límite de timeslots disponibles.

Una vez que se ha introducido toda la información requerida, cada uno de los links que aparecen en la lista aparecerá como un enlace, el que se debe conectar a un STP o un MSC, tal como aparece en el ejemplo de la figura 6.8.

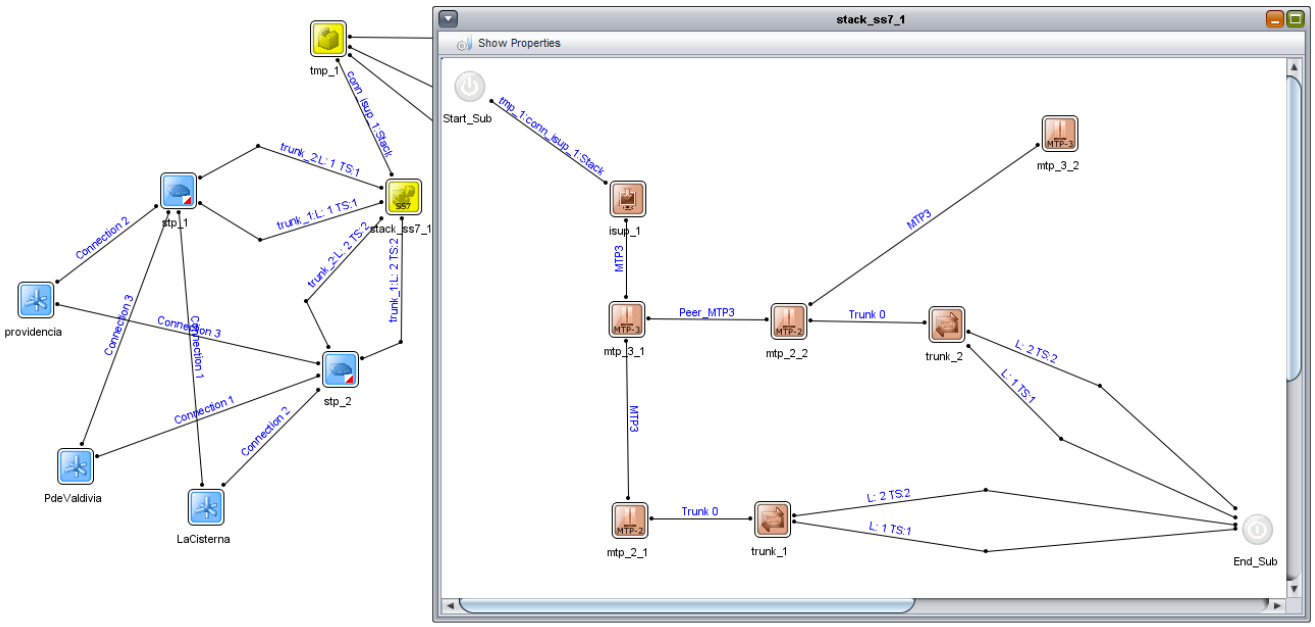

Figura 6.8: Ejemplo de configuración de Stack SS7, con dos trunks, cada uno con dos links.

ISUP

Mediante esta entidad se puede configurar la capa ISUP del Stack SS7, en donde en su diálogo aparece una tabla en la cual se pueden especificar los diferentes filtros de circuitos que se aplicarán a cada uno de los destinos (MSC) que son alcanzados por el Stack.

Además se puede especificar cuál sera el TMP que recibirá la conexión, siempre y cuando se encuentre unido a esta entidad (Se pueden conectar varios TMP a esta entidad).

También está la opción de no aplicar ningún filtro, por lo que al agregar el MSC a la tabla solo se selecciona el TMP que recibirá la conexión.

En la figura 6.9 se puede apreciar el diálogo, con la tabla de filtros y los TMP seleccionados.

6.3. Plug-in de generación

El plug-in de generación es una pieza fundamental en este programa, pues une el mundo del diagrama con el de la plantilla de transformación. Como resultado se obtienen finalmente los archivos de configuración de todas las máquinas involucradas en el sistema que el usuario diagramó y el archivo maestro que contiene la información necesaria para instalar cada uno

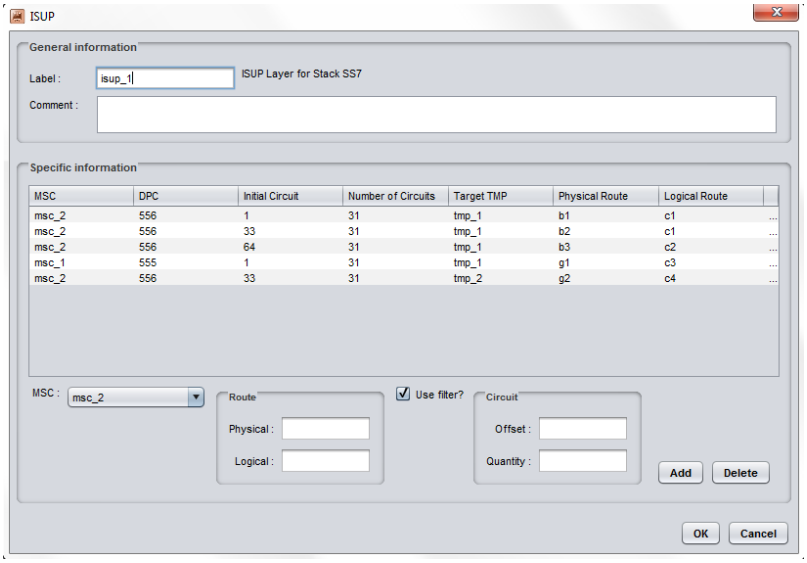

Figura 6.9: Dialogo de la entidad ISUP del Stack SS7.

de ellos.

Este plug-in primero toma la aplicación del SCE y fuerza su guardado a disco, de manera de obtener el archivo XML en el cuál está contenida toda la aplicación. Después le ofrece al usuario elegir cuáles módulos serán los que se procesarán (entidades contenedoras como el TMP o el Stack SS7), con la ventana que se muestra en la figura 6.10.

Una vez que el usuario aceptó los módulos de los que desea generar su configuración, el extractor pasa a otra ventana, en la cuál se muestra el progreso de la generación. En esta etapa el plug-in procesa el archivo XML de la aplicación aplicando la plantilla de transformación XSLT, operación que retornará un gran archivo, el cuál se corta con las lineas de división mencionadas en la sección 6.1.1. Después todo se empaqueta en un archivo ZIP para luego ser cargado al Smilodon, producto de Sixbell que entre sus variadas funcionalidades se encarga de inyectar los archivos de configuración a las diferentes máquinas según el archivo maestro generado.

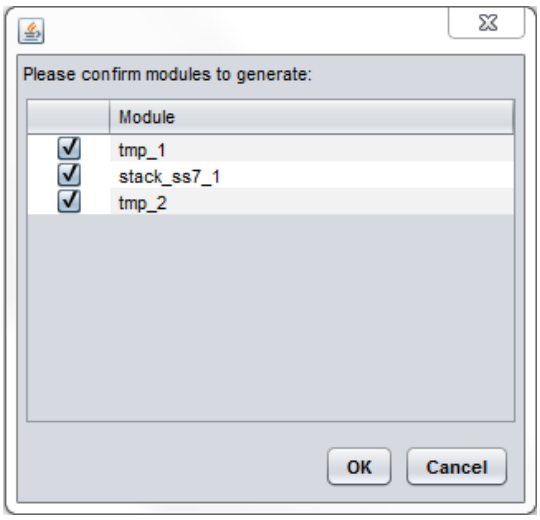

Figura 6.10: Ventana inicial del extractor.

6.4. Configurador dentro de Smilodon

Smilodon es una herramienta de monitoreo, provisionamiento y control centralizado de los productos que ofrece Sixbell. Su interfaz web hace simple todas las tareas de administración del sistema a los operadores de las compañías clientes, reduciendo los tiempos de puesta en marcha y facilitando la agregación de nuevas aplicaciones y la mantención de éstas.

Con la llegada del SCE en su versión configurador, se le agregó una nueva sección, en donde se pueden cargar configuraciones hechas en el SCE para luego propagarlas a través del sistema de manera automática. Incluso soporta la carga de varias configuraciones, dándole la opción al usuario de tener su sistema preparado para varios escenarios, o de tener una configuración segura antes de efectuar un cambio en el sistema. Esto es una de las características más fuertes del proyecto, ya que permite una respuesta rápida frente a cambios en los sistemas, o solución probada y de veloz aplicación en caso de fallas al aplicar nuevas configuraciones.

La inyección de archivos se realiza mediante SSH y usuarios especialmente creados durante la instalación de los paquetes de los programas de Sixbell. Esto es un riesgo de seguridad grande, pero hasta el momento es la única solución para lograr una conexión remota y poder copiar archivos además de ejecutar comandos para bajar y reestablecer servicios. Por suerte

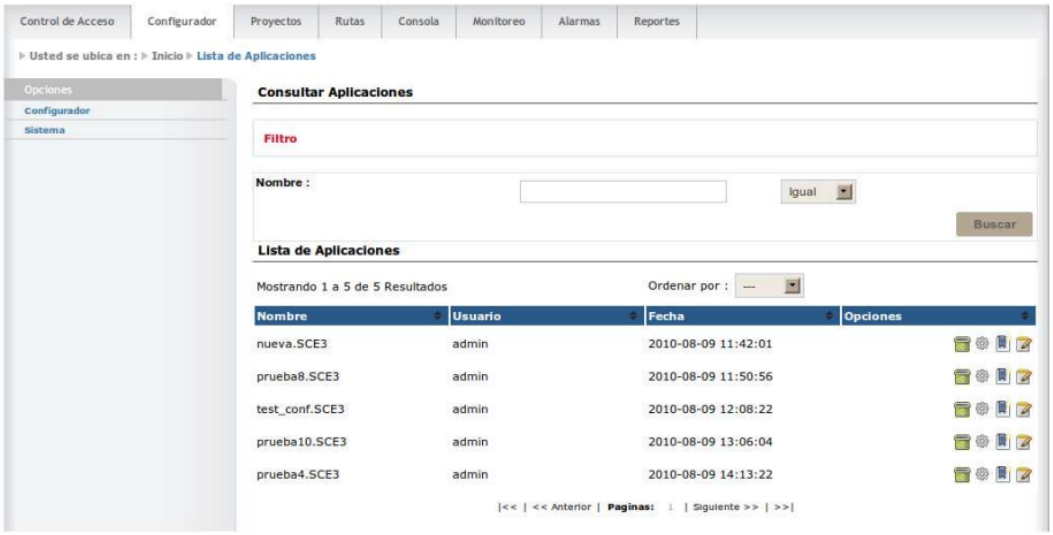

Figura 6.11: Ventana inicial del extractor.

la mayoría de estos sistemas se encuentran en una red cerrada, sin salida a internet, salvo contadas excepciones.

En la figura 6.11 se puede ver un pantallazo de la sección de configuración del Smilodon, donde se pueden apreciar varias configuraciones cargadas y la opción de aplicar cada una o modificarla, en cuyo caso se carga, mediante Java WebStart, el SCE versión configurador, con la aplicación lista para editar.

Capítulo 7

Puesta en producción

Desde que se pasó a la etapa de producción de este proyecto, estos son los comentarios de los principales usuarios.

7.1. Sixbell - Unidad de Negocio de Brasil

El configurador es una herramienta que facilita la interpretación por parte del cliente de la topología de red y su interconexión con el sistema implementado. En especial provee una abstracción importante sobre la vasta gama de archivos de configuración, todos en distintos formatos y de gran complejidad.

7.2. Sixbell - Unidad de Negocio de México

En México, el configurador nos permitió cumplir con el requerimiento comercial y técnico de incorporar un Element Manager (EM) en la solución[1], el cual debía permitir realizar la configuración y visualización de esta gráficamente.
Lo que le resta a este sistema para cumplir cabalmente con el requerimiento de EM, radica en combinar la parte gráfica del configurador, con el sistema de monitoreo y alarmas.
Por ejemplo: si un enlace de señalización sufre problemas, en la configuración se debería apreciar un cambio de color en la línea que representa el enlace con dichos problemas. Si se hace doble click sobre el enlace se debería observar más información asociada a la falla. Una vez que se restablezca el enlace este debe retornar a su color normal.

[1] http://en.wikipedia.org/wiki/Element_Management

7.3. Desde la perspectiva de Desarrollo

Permite realizar modificaciones a los archivos de configuración:

- Sumando parámetros nuevos
- Cambiando formatos y comportamientos
- Adicionar nuevos elementos

Todo esto se puede realizar sin alterar la configuración gráfica, y en el caso que alguna funcionalidad deba ser conocida por el usuario final, el impacto es menor modificando levemente la interfaz de alguna entidad involucrada.

7.4. Necesidades detectadas para el futuro

- Interfaz full web: Esto surge producto que el sistema se ejecuta localmente, genera una serie de inconvenientes (bloqueos de firewall, incompatibilidad con versiones de java instaladas, la solución limita el desarrollo de sistemas hosteados).
- Mejoras en la usabilidad de las entidades.
- Soporte para nuevas entidades y arquitecturas.
- Generar a partir de la información contenida en las entidades de la configuración aplicada un reporte del inventario.

Capítulo 8

Conclusión

Con el SCE versión configurador Sixbell ahora tiene dos muy buenos productos, ya que permitió, por una parte, la separación total del SCE de lo que es el lenguaje de creación de aplicaciones telefónicas, y aun más importante, la creación de esta nueva herramienta que permite tener una manera más sencilla y rápida de tener en marcha los sistemas frente a distintos escenarios. El configurador puede ser presentado como una gran ventaja frente a la competencia para los clientes.

La separación del SCE de lo que siempre fue su función, la creación de aplicaciones telefónicas, marca un gran hito, pues ahora el SCE se convierte en un producto en si. De hecho, durante el final del trabajo ya se estaba intentando vender a empresas con problemas que podían ser modelados mediante el SCE y que necesitaban solo la creación del lenguaje respectivo, para ello se crearon manuales y lenguajes pequeños a modo de tutorial.

El configurador, por su parte, trajo todo un cambio cultural en la empresa, dado que desde su puesta en marcha ahora las configuraciones se manejan solo a través del SCE y solo en contados casos se necesitan cambios hechos a mano, los cuales son rápidamente reportados para agregarlos a la plantilla de transformación del lenguaje. Los comentarios que están en el capítulo de puesta en marcha dan prueba de ello.

Los beneficios para las empresas que usan los sistemas de Sixbell están a la vista, y en combinación a la herramienta Smilodon, que permite que se tengan disponibles varias configuraciones hacen de todo este trabajo una poderosa herramienta que reduce los tiempos de

puesta en marcha de los sistemas, provee alternativas en caso de fallas y da la posibilidad de hacer frente de una buena manera a ampliaciones o cambios de los sistemas. Todos objetivos logrados y propuestos al empezar con el trabajo.

Capítulo 9

Bibliografía

[1] TELECOM MEDIA PORTAL, Manual de Descripción, Revisión 1.2 del 26 de Febrero 2010 [En linea], Sixbell, <Intranet de la empresa - www.sixbell.cl> [Consulta : 22 Agosto 2011]

[2] TELECOM MEDIA PORTAL, Manual de Instalación y Configuración, Revisión 1.3 del 26 de Febrero 2010 [En linea], Sixbell, <Intranet de la empresa - www.sixbell.cl> [Consulta : 22 Agosto 2011]

[3] TELECOM MEDIA PORTAL, Manual de Operación, Revisión 1.3 15 de Junio 2010 [En linea], Sixbell, <Intranet de la empresa - www.sixbell.cl> [Consulta : 22 Agosto 2011]

[4] STACK SIP, Manual de Descripción, Revisión 2.0 del 31 de Julio 2009 [En linea], Sixbell, <Intranet de la empresa - www.sixbell.cl> [Consulta : 28 Septiembre 2011]

[5] STACK SIP, Manual de Instalación y Configuración, Revisión 2.0 del 31 de Julio 2009 [En linea], Sixbell, <Intranet de la empresa - www.sixbell.cl> [Consulta : 28 Septiembre 2011]

[6] STACK SS7, Manual de Descripción, Revisión 2.1 del 31 de Agosto 2010 [En linea], Sixbell, <Intranet de la empresa - www.sixbell.cl> [Consulta : 11 Octubre 2011]

[7] STACK SS7, Manual de Instalación y Configuración, Revisión 2.0 del 31 de Agosto 2010 [En linea], Sixbell, <Intranet de la empresa - www.sixbell.cl> [Consulta : 11 Octubre 2011]

[8] STACK SS7, Manual de Operación, Revisión 2.1 del 31 de Agosto 2010 [En linea], Sixbell, <Intranet de la empresa - www.sixbell.cl> [Consulta : 11 Octubre 2011]

Apéndices

A . Archivo de configuración Control CCXML/VXML

```
#(c) Copyrights 2009 Consorcio Tecnologico Sixlabs Ltda.
#SCE version  : remote (remote)
#---------sigas_control2.conf---------
configuration:{
        #CONTROLS
        control:{
                name          : "control";
                machine       : "lleuque";
                ivr_mode      : "true";
                benchmarking_enabled   : "false";
                http_timeout  : "4";          # Tiempo maximo que debiera demorar el fetch
                http_maxage   : "5";          # Edad del Documento
                http_maxstale : "5";          # Validez del Documento
                http_verify_cert       : "disabled"; # Verificacion de Certificados
                use_hostName  : "disabled";   # Agrega sufijo hostName al nombre de colas AMQ
                gc_messages   : "1000";       # Cantidad de Mensajes antes de ejecutar GC
                gc_times      : "disabled";   # Horas en que se ejecutar una tanda de GC, ej:
                    |12:40|18:27|
                stats:{
                        windowSeconds      : "120"; # En segundos
                        updatesBeforeScrub : "50";
                        reportInterval     : "120"; # En segundos
                }; #stats
                interpreter   :{
                        ccxmlThreads       : "1";
                        vxmlThreads        : "1";
                        javascriptsRuntime : "43";
                        dispatcherNumber   : "1";
                        scriptsDirectory   : ".";
                        visualizeFile      : "disabled";
                        routingRules       : "/opt/sixlabs/etc/control/routing-rules.conf";
                        constantsFilePath  : "/opt/sixlabs/etc/control/cnstnts.js";
                }; #interpreter
                gateway:{
                        gateway_mode       : "false";
                        ipStackSip         : "192.168.1.204";
                        portStackSip       : "5060";
                }; #gateway
                monitor:{
                        name               : "sigasControl";
                        pass               : "sigascontrol001";
                        id                 : "1.1";
                        serverIp           : "127.0.0.1";
                        serverPort         : "10123";
                        clientIp           : "127.0.0.1";
                        clientPort         : "10036";
                        regDelay           : "60";
                        active             : "false";
                }; #monitor
                console:{
                        server             : "127.0.0.1";
                        port               : "2523";
                        active             : "false";
                }; #console
                logs:{
                        path               : "/opt/sixlabs/var/log/";
                }; #logs
                debug:{
                        level              : "ALL";
                }; #debug
                media:{
                        name               : "control";
                        asr_mrcp           : "disabled";
                        tts_mrcp           : "disabled";
                }; #media
                traps:{
                        server             : "192.168.1.204";
                        community          : "sixbell.cl";
                        active             : "true";
                }; #traps
                activemq:(
```

```
71                      {
72                              uri                     : "tcp://192.168.1.204:61613?wireFormat=stomp&transport.
                                    useAsyncSend=true";
73                              group-id                : "CONTROL_ASN11";
74                              org-id                  : "CONTROL_ASN11_1";
75                              producer-list           : (
76                                      { connector-sip           : "SGNL_SIP_CONTROL10";},
77
78                                      { connector-mrcp          : "CONN_MRCP";},
79                                      { media-resource          : "MediaResource15";},
80
81                                      { connector-file          : "CONN_FILE16";},
82
83                                      { connector-socket        : "CONN_SOCKET";},
84                                      { connector-diameter : "SGNL_DIAMETER_CONTROL";}
85                              );
86                              msg-type                : "asn1";
87                      },
88                      { # BAS - DBENABLER - SMILODON LOCAL
89                              uri                     : "tcp://192.168.1.204:61613?wireFormat=stomp&transport.
                                    useAsyncSend=true";
90                              group-id                : "CONTROL_TEXT";
91                              org-id                  : "CONTROL_TEXT_1";
92                              producer-list : (
93                                      { connector-db            : "queue/OrchServiceQueue";},
94                                      { bas                     : "queue/OrchServiceQueue";},
95                                      { smilodon-local          : "queue/OrchServiceQueue";}
96                              );
97                              msg-type                 : "text";
98                      },
99                      { # SMILODON CENTRAL
100                             uri                     : "tcp://192.168.1.204:61613?wireFormat=stomp&transport.
                                    useAsyncSend=true";
101                             group-id                : "CONTROL_SMILODON";
102                             org-id                  : "CONTROL_SMILODON_1";
103                             producer-list : (
104                                     { smilodon-central        : "queue/OrchServiceQueue";}
105                             );
106                             msg-type                : "text";
107                     }
108                     );#activemq
109             console_topic:{
110                     uri              : "tcp://192.168.1.204:61613?wireFormat=stomp&transport.useAsyncSend=true";
111                     consumer-list : (
112                             { console : "TOPIC-CONTROL";},
113                             { console : "TOPIC-CONTROL_CCXML_1";},
114                             { console : "TOPIC-BLAHF";},
115                             { console : "TOPIC-Los_Leones_1200";}
116                     );
117             };
118             }; #control
119     };
```

B . Archivo Routing Rules

```
interpreter_type{
        CCXML {
                header.org.type|not|equalsnumeric|8
        }
        VXML {
        }
}

CCXML_filename {
        /opt/sixlabs/var/lib/control/scripts/ccxml_apps/tmpas.ccxml {
        }
}

VXML_filename {
        main.vxml {

        }
}
```

C . Archivo de configuración de MR NMS

```
[Configuracion]
        BusQueueUri             = tcp://192.168.1.204:61613?wireFormat=stomp&transport.useAsyncSend=true
        BusTopicUri             = tcp://192.168.1.204:61613?wireFormat=stomp&transport.useAsyncSend=true
        TopicList{0}            = TOPIC_SMIlodon_MR
        TopicList{1}            = TOPIC_MR_1
        TopicList{2}            = TOPIC_BLAHF
        TopicList{3}            = TOPIC_GROUP1
        BusTimeMonitor          = 5000
        CacheTimeOut            = 60000
        DownloadPath            = /opt/sixlabs/var/lib/mr/media/cache/
        MRIdentifier            = MediaResource15
        PoolNotAllowedNumber    = -1
        PoolNotAllowedOffset    = -1
        Protocol                = RAW

[Constabla]
        Boards                  = /opt/sixlabs/etc/mr/Boards.txt
        Channels                = /opt/sixlabs/etc/mr/Channels.txt

[Gestion]
        EnableSNMPTrap          = true
        SNMPServers             = 192.168.1.204
        TrapConfig              = /opt/sixlabs/etc/mr/TrapConfig.cfg

[IP]
        BehindNAT               = false
        DtmfDecMode             = 1
        DTMFDuration            = 200
        DtmfEncMode             = 1
        DtmfPayLoad             = 0x00000065
        DtmfPlayOut             = -1
        EchoDspFusionMode       = 0xa
        EnableDtmfToIvr         = true
        EnableInBandDtmf        = false
        EnableRTCP              = true
        FramesPerPacket         = 2
        g723rate5_3             = false
        InputGain               = 0x00000400
        JITTER_ADAPT_ENABLED    = 0
        JITTER_DEPTH            = -1
        NATAddress              = ""
        NOTCH_FILTERS           = 0
        OutputGain              = 0x00000400
        PrintStatus             = false
        VAD                     = true
        Codec{0}                = PCMU
        Codec{1}                = PCMA
        DynPlG726_16            = 102
        DynPlG726_24            = 103
        DynPlG726_32            = 104
        DynPlG726_40            = 105
        DynPlAMR                = 106
        DynPliLBC20             = 107
        DynPliLBC30             = 108

[PSTN]
        CtaDefaultServer        = ""
        CTBusMaxStreams         = 32
        CTBusTimeslotsPerStream = 128
        CTBusUnidirectional     = 1
        echo_adapttime          = 1000
        echo_filterlength       = 0
        echo_mode               = 0xa
        OamEnabled              = false
        ParameterFile           = /opt/sixlabs/etc/mr/parameters.txt
        RecEnconderWavDefault   = 23
        RPCPort                 = 9010
        RulesFile{0}            = /opt/sixlabs/var/lib/mr/media/spanish.tbl
        RulesFile{1}            = /opt/sixlabs/var/lib/mr/media/english.tbl
        RulesFile{2}            = /opt/sixlabs/var/lib/mr/media/portuges.tbl
        SwitchThread            = false
        VoxFile{0}              = /opt/sixlabs/var/lib/mr/media/spanish.vox
        VoxFile{1}              = /opt/sixlabs/var/lib/mr/media/english.vox
        VoxFile{2}              = /opt/sixlabs/var/lib/mr/media/portuges.vox

[TRACE]
         LogLevel               = detail
         LogPath                = /opt/sixlabs/var/log/
         GestionLogPath         = /opt/sixlabs/var/log/
```

D . Archivo de configuración Señalizador ISUP

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--
        ########## engine-connector-isup_hss.xml ##########
-->
<engine>
        <connector>
                <instances number="480" offset="1"/>
                <routers-channels>
                        <channel type="bus" module="control" uri="tcp://192.168.2.165:61613?wireFormat=
                            stomp&transport.useAsyncSend=true" org="SGNL_ISUP25" dest="CONTROL_ASN26"/>
                        <channel type="bus" module="stack" uri="tcp://192.168.2.165:61613?wireFormat=stomp&
                            transport.useAsyncSend=true" org="SGNL_ISUP_STACK25" dest="STACK_ISUP20"/>
                </routers-channels>
                <instances-resources>
                        <group name="cis-1" group="cist-1" dpc="123" destination="LaCisterna" cic-offset="1
                            " cic-number="30" mr-queue="MR_1" boardId="0" boardName="CG_6565" streamId="0"
                             timeslot-offset="0" busType="0"/>
                        <group name="cis-2" group="cist-2" dpc="123" destination="LaCisterna" cic-offset="
                            32" cic-number="30" mr-queue="MR_1" boardId="0" boardName="CG_6565" streamId="
                            4" timeslot-offset="0" busType="0"/>
                        <group name="prov-1" group="prov-1" dpc="121" destination="providencia" cic-offset=
                            "1" cic-number="30" mr-queue="MR_1" boardId="0" boardName="CG_6565" streamId="
                            8" timeslot-offset="0" busType="0"/>
                        <group name="pdv-1" group="pdv-1" dpc="122" destination="PdeValdivia" cic-offset="1
                            " cic-number="30" mr-queue="MR_1" boardId="0" boardName="CG_6565" streamId="12
                            " timeslot-offset="0" busType="0"/>
                </instances-resources>
                <opc value="333"/>
                <monitor registrationInterval="60">
                        <remote ip="127.0.0.1" port="10456"/>
                        <local ip="127.0.0.1" port="20456"/>
                        <module name="sigas at austras" pass="sigaspass" id="0.2"/>
                </monitor>
                <topics enable="on" uri="tcp://192.168.2.168:61613?wireFormat=stomp&transport.useAsyncSend=
                    true">
                        <topic name="TOPIC_CONN_ISUP"/>
                        <topic name="TOPIC_"/>
                        <topic name="TOPIC_"/>
                        <topic name="TOPIC_"/>
                </topics>
        </connector>
</engine>
```

E . Archivo de configuración Señalizador NCC NMS

```
1
2   <?xml version="1.0" encoding="ISO-8859-1" ?>
3   <!--
4           Configuration file created by Sixlabs SCE 3.0
5           (c) Copyrights 2009 Consorcio Tecnologico Sixlabs Ltda.
6           ########## engine-ncc_sixlabs.xml ##########
7   -->
8   <engine>
9           <connector>
10                  <instances number="1" offset="1"/>
11                  <routers-channels>
12                          <channel type="bus" ip="127.0.0.1" name="SGNL_NCC41" destination="CONTROL_ASN26"/>
13                          <channel type="bus" ip="127.0.0.1" name="SGNL_NCC41" destination="CONTROL_ASN27"/>
14                          <channel type="board" number="1" name="Board_Channel"/>
15                  </routers-channels>
16                  <instances-resources>
17                          <group-name="pbx-six1" group="group1" trunk="0" destiny="pbx-six1p" timeslot-offset
                                ="0" timeslot-number="30" boardId="0" boardName="null" streamId="0" busType="0
                                "/>
18                  </instances-resources>
19          </connector>
20  </engine>
```

F . Archivo de configuración Connector File

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!--
        (c) Copyrights 2009 Consorcio Tecnologico Sixlabs Ltda.
        ########## engine-smtp_sixlabs.xml ##########
-->
<engine>
        <connector>
                <instances number="100" offset="0"/>
                <routers-channels>
                        <channel type="bus" ip="192.168.20.20" name="CONN_FILE28" destination="
                            CONTROL_ASN34"/>
                </routers-channels>
        </connector>
</engine>
```

G . Archivo de configuración Connector SMPP

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<engine>
        <connector>
                <instances number="100" offset="0"/>

                <smpp-params systemType="type_p" interfaceVersion="52" addrTon="2" addrNpi="1" addrRange="
                    666"/>

                <server-connection>
                        <channel type="server" ip="127.0.0.1" port="2775" name="serverA"/>
                        <auth systemId="sysId" password="passwd" />
                </server-connection>

                <routers-channels>
                        <channel type="bus" uri="tcp://127.0.0.1:61613?wireFormat=stomp&transport.
                            useAsyncSend=true" name="CONN_SMPP" destination="CONTROL_ASN1_1"/>
                </routers-channels>

                <engine-timers>
                        <timer name="wait4response" timeout="10000"/>
                </engine-timers>

                <topics enable="on" uri="tcp://127.0.0.1:61613?wireFormat=stomp&transport.useAsyncSend=true
                    ">
                        <topic name="TOPIC_CONN_SMPP_1"/>
                        <topic name="TOPIC_CONN_SMPP_N"/>
                </topics>
        </connector>
</engine>
```

H . Archivo de configuración Connector SMTP

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<engine>
        <connector>
                <instances number="100" offset="0"/>

                <mail-server address="111.111.111.111"/>

                <routers-channels>
                        <channel type="bus" ip="127.0.0.1:61613?wireFormat=stomp&transport.useAsyncSend=
                            true" name="CONN_SMTP39" destination="CONTROL_ASN34"/>
                </routers-channels>

                <topics enable="on" uri="tcp://127.0.0.1:61613?wireFormat=stomp&transport.useAsyncSend=
                    true">
                        <topics name="TOPICS_CONN_SMTP_1"/>
                        <topics name="TOPICS_CONN_SMTP_N"/>
                </topics>
        </connector>
</engine>
```

I . Archivo de configuración Connector Socket

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<engine>
        <connector>
                <instances number="100" offset="0"/>
                <connection-params maxSize="11000"/>

                <routers-channels>
                        <channel type="bus" uri="tcp://127.0.0.1:61613?wireFormat=stomp&transport.
                            useAsyncSend=true" name="CONN_SOCKET" destination="CONTROL_ASN_1"/>
                </routers-channels>

                <engine-timers>
                        <timer name="wait4response" timeout="1000"/>
                </engine-timers>

                <topics enable="on" uri="tcp://127.0.0.1:61613?wireFormat=stomp&transport.useAsyncSend=true
                    ">
                        <topic name="TOPIC_CONN_SOCKET_1"/>
                        <topic name="TOPIC_CONN_SOCKET_N"/>
                </topics>
        </connector>
</engine>
```

J . Archivo de configuración Stack SS7 - MTP2

```
1  ##########################################################
2  ### MTP2 CONFIGURATION FILE EXAMPLE FOR 2.x VERSIONS ###
3  ##########################################################
4  # WARNING !!: YOU MUST MERGE THE STACK SECTION OF SS7_PH_CONF
5  # FILE WITH THIS FILE TO WORK PROPERLY .
6  # WATCH FOR DUPLICATED ENTRIES .
7  ##########################################################
8
9  ####################################################
10 ################ APPL SECTION ######################
11
12 ################ LOG SECTION ######################
13 # Each level could take 0 or 1 value:
14 # 1 => DO LOG MESSAGES OF THIS TYPE
15 # 0 => DO NOT LOG MESSAGES OF THIS TYPE
16 #
17 # LOG_SYSTEM 1
18 # LOG_ERROR 1
19 # LOG_INIT 1
20 # LOG_NETWORK 0
21 # LOG_ROUTE 0
22 # LOG_TRAFFIC 0
23 # LOG_DEBUG 0
24 # LOG_TIMER 0
25 # LOG_SUCCESS 0
26 # LOG_MEMORY 0
27 # LOG_PATH /opt/ sixlabs /var /log
28 #
29 # Override previous logs settings
30 # LOG_ALL 1
31
32 # MTP2_LOG_FLUSH YES
33 # MTP2_LOG_MEMORY_MB 128
34 ############### END OF LOG SECTION #################
35
36 ############### QUEUES SECTION ####################
37 # MTP2_REDUNDANCY NO
38 # RD_QUEUE_KEY 0 x60800
39 # MTP2_QUEUE_KEY 0 x60802
40 # MTP2_QUEUE_MTYPE 0x20
41 # MTP2_ADM_QUEUE 0 x60210
42 # MTP2_ADM_MTYPE 0x20
43 ####################################################
44
45 ############### TCP/IP PORTS SECTION ###############
46 # MTP3_MTP2_PORT_1 60204
47 # MTP3_MTP2_PORT_2 60214
48 # MTP2_ETHEREAL_PORT 60220
49 ######### END OF TCP/IP PORTS SECTION #############
50
51 ############### TRUNKS SECTION ####################
52 # Will this trunk be created ? YES|NO
53 TRUNK_0_ACTIVATE YES
54 TRUNK_1_ACTIVATE NO
55
56 #This trunk is in card : PT|ADAX | DIALOGIC
57 TRUNK_0_CARD_MANUFACTURER ADAX
58 TRUNK_1_CARD_MANUFACTURER PT
59
60 #Set trunk device:
61 #For PT cards : ptixMTP2c [0 -9](\.[0 -3]) ?
62 # [PCI Slot ] & ( TRUNK DEVICE )
63 #For ADAX cards : /dev/ hdcx
64 #For Dialogic cards: dummy value
65 TRUNK_0_DEVICE /dev /hdcx
66 TRUNK_1_DEVICE ptixMTP2c0
67
68 ############### END OF TRUNKS SECTION #############
69
70 ######### END OF APPL SECTION ######################
71 ####################################################
72
73
74 ####################################################
75 ################ MTP3 SECTION ######################
76
77 # Network Indicator value to MTP3 layer
78 # Values : NATIONAL ,
79 # INTERNATIONAL , TRANSPARENT
80 # NETWORK_INDICATOR TRANSPARENT
81 # Network Indicator value to SS7 network
82 # Values : NATIONAL , NATIONAL_PRIORITY ,
83 # INTERNATIONAL , TRANSPARENT
84 # TRUNK_0_NW_ID NATIONAL
85 # TRUNK_1_NW_ID NATIONAL
86
87 # Set if SLT BIT is changed to REGULAR or SPECIAL
88 # TRUNK_0_SLTC_MODE REGULAR
89 # TRUNK_1_SLTC_MODE REGULAR
90
91 ################ END OF MTP3 SECTION #############
```

```
92  ##################################################
93
94
95  ##################################################
96  ################ MTP2 SECTION ####################
97
98  ################ NODE TYPE SECTION ###############
99  # ITU/ANSI ?
100 # MTP2_STANDARD_TYPE ITU
101 ############ END NODE TYPE SECTION ###############
102
103 ############ MTP2 TIMERS SECTION #################
104 # ALL VALUES set in units of 0.01 sec:
105
106 # T1 Alignment ready timer.
107 # ANSI [1300..3000]; recomm: 1300
108 # ITU [4000..5000]
109 # ITU HSL [2500..35000]
110 # DTLK_T1_VALUE 4500
111
112 # T2 Not aligned timer timer.
113 # ANSI [500..2350]; recomm : 1150
114 # ITU/ITU HSL [500..15000]
115 # DTLK_T2_VALUE 1150
116
117 # T3 Aligned timer.
118 # ANSI [500..1400]; recomm: 1150
119 # ITU [100..150]
120 # ITU HSL [100..200]
121 # DTLK_T3_VALUE 100
122
123 # T4n Normal proving period timer.
124 # ANSI [207..253]; recomm: 230
125 # ITU [750..950] ITU
126 # ITU HSL [300..7000] ITU HSL
127 # DTLK_T4N_VALUE 800
128
129 # T4e Emergency proving period timer.
130 # ANSI [54..66]; recomm: 60
131 # ITU/ITU HSL [40..60]
132 # DTLK_T4E_VALUE 60
133
134 # T5 Timer " Sending SIB "
135 # ITU [80..120]; recomm : 100
136 # DTLK_T5_VALUE 100
137
138 # T6 Remote Congestion timer.
139 # ALL [300..600]; recomm : 500
140 # DTLK_T6_VALUE 500
141
142 # T7 Excessive delay of ACK.
143 # ALL [50..200]; recomm : 200
144 # DTLK_T7_VALUE 200
145 ########## END MTP2 TIMERS SECTION ###############
146
147 ########## MTP2 LINKS SECTION ####################
148 #Set MTP2_ID to identify this trunk
149 TRUNK_0_MTP2ID 0
150 TRUNK_1_MTP2ID 1
151
152 # [A] = Trunk number (0 ... N)
153 # [B] = Link number (1 .. 23/31) (T1/E1)
154 #LINK [A]_[B] Timeslot LINK NAME
155 LINK0_1 1 STP_A_1
156 LINK1_1 1 STP_B_1
157
158 # HSL mode : OFF|SN7| SN12
159 #( note OFF => normal (E1 - >31/T1 - >23) timeslot mode )
160 # TRUNK_0_UNCHANNELIZED OFF
161 # TRUNK_1_UNCHANNELIZED OFF
162
163 ########## END MTP2 LINKS SECTION ################
164
165 ############## END MTP2 SECTION ##################
166 ##################################################
167
168
169 ##################################################
170 ############## MTP1 SECTION ######################
171
172 ############## TRUNK 0 SECTION ###############
173 #Set trunk type : E1|T1
174 # TRUNK_0_TYPE E1
175
176 #IF trunk == E1:
177 ## CRC mode : ON|OFF
178 # TRUNK_0_E1_CRC OFF
179
180 ## Remote alarm mode : USE_A| USE_AIS
181 # TRUNK_0_E1_REMOTE_ALARM USE_A
182
183 #IF trunk == T1:
184 ## Framing mode : D4|ESF
185 # TRUNK_0_T1_FRAMING_MODE ESF
186
187 ## Yellow alarm mode : NOT_TRSM | IN_BIT2 | IN_SBIT
```

```
# TRUNK_0_T1_YELLOW_ALARM NOT_TRSM

## Out Of Frame ratio: 2 OVER4 |2 OVER5 |2 OVER6
# TRUNK_0_T1_OOFRAME_RATIO 2 OVER6

## T1 Channel BandWidth : 56|64
# TRUNK_0_T1_CHANNEL_BW 64
############### END TRUNK 0 SECTION ############

############### TRUNK 1 SECTION ################
#Set trunk type : E1|T1
# TRUNK_1_TYPE E1

#IF trunk == E1:
## CRC mode : ON|OFF
# TRUNK_1_E1_CRC OFF

## Remote alarm mode : USE_A| USE_AIS
# TRUNK_1_E1_REMOTE_ALARM USE_A

#IF trunk == T1:
## Framing mode : D4|ESF
# TRUNK_1_T1_FRAMING_MODE ESF

## Yellow alarm mode : NOT_TRSM | IN_BIT2 | IN_SBIT
# TRUNK_1_T1_YELLOW_ALARM NOT_TRSM

## Out Of Frame ratio: 2 OVER4 |2 OVER5 |2 OVER6
# TRUNK_1_T1_OOFRAME_RATIO 2 OVER6

## T1 Channel BandWidth : 56|64
# TRUNK_1_T1_CHANNEL_BW 64
############### END TRUNK 0 SECTION ############
```

K . Archivo de configuración Stack SS7 - MTP3

```
1  ##################################################################
2  #                                                                #
3  #            SIXBELL SS7 STACK CONFIGURATION FILE                #
4  #                                                                #
5  ##################################################################
6  #                                                                #
7  # WARNING!!: YOU MUST MERGE THE MTP2 SECTION OF SS7_PH_CONF      #
8  #             FILE WITH THIS FILE TO WORK PROPERLY.              #
9  #            WATCH FOR DUPLICATED ENTRIES.                       #
10 #                                                                #
11 ##################################################################
12
13 ##################################################################
14 # Sixbell OID
15 SNMP_OID                    4227.2.1.1.1.5
16 ##################################################################
17
18 ##################### STACK log data ##########################
19 LOG_PATH
20 STACK_LOG_FLUSH             YES
21 STACK_LOG_MEMORY_MB         128
22 STACK_LOG_FILESIZE          16
23 ##################################################################
24
25 ############### MTP3 Standard and Node Type ###################
26 # STACK_STANDARD_TYPE (ITU|ETSI|ANSI|SPANISH|BELLCORE)
27 STACK_STANDARD_TYPE         ITU
28 # MTP3_NODE_TYPE (SP|STP)
29 MTP3_NODE_TYPE              SP
30 # NETWORK_INDICATOR (NATIONAL|INTERNATIONAL)
31 NETWORK_INDICATOR           INTERNATIONAL
32 ##################################################################
33
34 ###################    IP ADDRESS USED    ######################
35 # Readed for STACK to contact local MTP2
36 MTP2_IP_ADDRESS_1           192.168.2.168
37 # Readed for STACK to contact remote MTP2
38 MTP2_IP_ADDRESS_2           192.168.2.168
39 #SMT_IP_ADDRESS             127.0.0.1
40 # IP address to by binded.
41 LOCAL_SS7_REDN_ADDRESS      192.168.2.168
42 PEER_SS7_REDN_ADDRESS       192.168.2.168
43 ##################################################################
44
45 #############     RD PROCESS COMMUNICATION      ################
46 # Every process has one queue key and one queue mtype
47 # RD must be configured to use the same values
48 # RD does NOT have a queue mtype, because it answers
49 # the messages using each process mtype.
50 STACK_REDUNDANCY            YES
51 RD_QUEUE_KEY                0x60800
52 STACK_REDN_QUEUE_KEY        0x60804
```

```
STACK_REDN_QUEUE_MTYPE      0x40
#################################################################

#################### ISUP MULTIQUEUE ########################
#
# ActiveMQ support for ISUP protocol
ISUP_AMQ                    YES
ISUP_AMQ_QUEUE              STACK_ISUP19
ISUP_AMQ_IP                 192.168.2.165
ISUP_AMQ_LOCAL_PORT         61613

#Additional connections to AMQ
#Sintax:
#ISUP_AMQ_N    QUEUE          IP
#ISUP_AMQ_1    STACK_ISUP1    127.0.0.1

# Syntax:
# ISUP_LOCAL_SPC_N        SELF_PC
ISUP_LOCAL_SPC_1          333

# Syntax:
# Using UNIX queues:
# Using AMQ (default connection):
# ISUP_DRIVER_N           QUEUE                   NAME
ISUP_DRIVER_1             SGNL_ISUP_STACK25       DRIVER_ISUP_25

# Using AMQ (additional connection):
# ISUP_DRIVER_N           QUEUE                   ISUP_AMQ_X      NAME

# Syntax
#ISUP_MSC_N      DPC   CIC(Y/N)    MIN_CIC    MAX_CIC    DRIVER_NAME
ISUP_MSC_1       123         Y          1         30      DRIVER_ISUP_25
ISUP_MSC_2       123         Y         32         61      DRIVER_ISUP_25
ISUP_MSC_3       121         Y          1         30      DRIVER_ISUP_25
ISUP_MSC_4       122         Y          1         30      DRIVER_ISUP_25
#################################################################

#########################  QUEUES ###########################
TCAP_A_QUEUE_KEY            0x60415
TCAP_B_QUEUE_KEY            0x60425
#################################################################

#########################  PORTS  ###########################
SCCP_SERVER_A_PORT          60514
SCCP_SERVER_B_PORT          60524
MTP3_MTP2_PORT_1            60204
MTP3_MTP2_PORT_2            60214
SMT_PORT                    60304
LOCAL_SS7_REDN_PORT         60404
PEER_SS7_REDN_PORT          60414
HTTPD_PORT                  60420
#################################################################

#####################  MTP2 Conv  ###########################
#USED TO IDENTIFY WICH TRUNKS ARE LOCAL AND REMOTE
# LOCAL_MTP2ID_N            MTP2ID
LOCAL_MTP2ID_1              1
# REMOTE_MTP2ID_N           MTP2ID
REMOTE_MTP2ID_1             2
#################################################################
```