Capítulo 4

Implementación

Capítulo 4 Implementación

Como se mencionó en los capítulos anteriores, las herramientas desarrolladas para la transmisión de audio, imagen, video y texto fueron desarrollados en el lenguaje de programación java. Dentro del lenguaje java existen otros paquetes que se utilizaron para el desarrollo de las herramientas, los cuales se mencionaran a lo largo de este capítulo mas detalladamente.

En este capítulo también se describen todos los pasos que se tuvieron que seguir para el desarrollo e incorporación del servidor "Conexion" desarrollado para el CU Communicator el cual interactuará con las herramientas desarrolladas. El servidor "Conexion" está desarrollado con una sintaxis especial del lenguaje C, la cual es desarrollada bajo la Arquitectura Galaxy.

Las herramientas desarrolladas para la transmisión de video, imagen y audio se implementaron para que funcionen mediante un applet. Para las herramientas de desarrolladas de transferencia de audio y video, se captura y reproduce los datos multimedia. En cambio para la herramienta de imagen solo se visualiza. La transferencia del texto se realizó mediante un servlet, el cual envía y recibe texto. Más adelante se explicará detalladamente la implementación de cada una de las herramientas desarrolladas.

Una vez que se desarrollaron las herramientas, se implemento el servidor "Conexion" el cual mantiene una comunicación con cada una de las herramientas mediante uno de los API de java llamado JNI (Java Native Interface). Este API nos permite combinar el lenguaje Java con lenguajes nativos como lo es el lenguaje C. El servidor "Conexion" en el momento que recibe una petición de transferencia de imagen, texto, video o audio, envía

un mensaje para que se inicie la transferencia. El Hub es que recibe los mensajes e inicializa la transferencia. Por razones de tiempo no se pudo modificar las reglas del Hub para que interactuará con los otros servidores, pero se deja especificada la manera de cómo se debe hacer.

4.1 Transferencia de video

La captura de video se realizó mediante un API de Java llamada Java Media Framework (JMF). JMF permite la captura y reproducción de video en diversos formatos. JMF Satisface el cruce entre plataformas, protocolos y contenido neutral, captura y reproducción de los medios audiovisuales, además de soportar una amplia gama de formatos de archivo de video y audio [Gordon & Talley 99].

El JMF se utilizó para implementar un reproductor (player) multimedia que recibe y reproduce datos multimedia desde fuentes almacenadas localmente o que se están produciendo en tiempo real. Para el desarrollo del reproductor se tomo en cuenta el modelo de seis estados de Java Media basados en dos estados fundamentales Stopped y Started.

Este modelo es el que acompaña al diagrama de estados (ver Fig. 4.1), con el estado Stopped en verde y el estado Started en blanco. Los estados pueden estar sin realizar (Unrealized), Realizándose (Realizing), Realizados (Realized), Prefetching, Prefetched, y Started. Las transiciones entre Realizing a Realized y Prefetching a Prefetched son automáticas (Realizing y Prefetching) son estados dinámicos de longitud indeterminada. Otras transiciones son traídas por medio de llamadas a métodos.

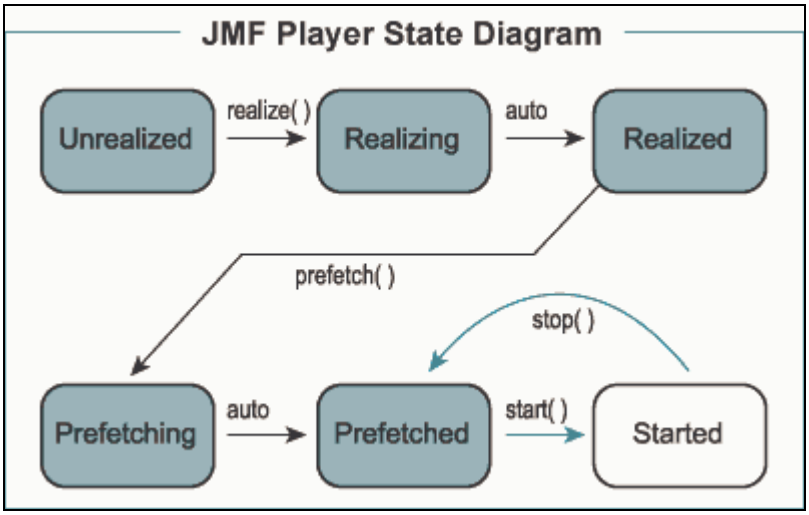


Fig. 4.1 Diagrama de estados del funcionamiento del JMF [Gordon & Talley 99]

Además, el reproductor se creará directamente de un URL que nos proporciona una manera fácil de insertar datos multimedia dentro de un Applet y aplicaciones basadas en Java.

El modelo que utiliza Java Media Framework es muy parecido al que utilizan los dispositivos de video. Primeramente se tiene una fuente la cual se encargará de capturar los datos mediante una cámara, los datos serán almacenados como streams. Una vez que son almacenados se leen y procesan mediante algún dispositivo, para que finalmente se envíen estos datos a un dispositivo de salida.

La herramienta desarrollada con JMF nos permitió implementar interfaces que permiten un adecuado manejo de los datos. A continuación se describirán algunas de las principales interfaces utilizadas para desarrollar la aplicación que captura y transmite el video:

- **Interfaz Datasource** [Sun Microsystems 2001]**:** nos permite manejar el contenido que se encuentra dentro de un stream de datos, debido a que nos provee la ubicación y el software que lo produjo. Incluyendo a los dispositivos encargados de la captura de multimedia como es una cámara de video.
- **Interfaz Controller** [Sun Microsystems 2001]**:** esta interfaz define el estado básico de un stream y provee mecanismos de control para cualquier objeto que tenga como funcionalidad controlar, presentar o capturar tipos de dato multimedia. Existen 2 tipos de controladores, el primero es el Player encargado de procesar la información. El segundo es el Processor el cual está encargado de ejecutar la información.
- **Interfaz Player** [Sun Microsystems 2001]**:** es un objeto que implementa la interfaz Player para procesar un flujo de datos que cambian con el tiempo, leyendo datos de un Datasource y ejecutándolo en un momento preciso.

Interfaz Processor [Sun Microsystems 2001]**:** Es un tipo de Player, ya que puede ejecutar una entrada de datos multimedia. Esta interfaz provee la libertad, a quien desarrolla la aplicación, de aplicar el proceso de datos que le funcione de mejor manera. De esta manera la aplicación puede desarrollar efectos que serán presentados en tiempo real.

Mediante JMF se pueden implementar el soporte para casi todos los formatos de video (ver tabla 4.1).

- D indica que el formato puede ser codificado y presentado.

- E indica que el stream de datos multimedia puede ser codificado dentro del formato.

Tabla 4.1 Formatos de video soportados

Tipo de Dato Multimedia	**JMF 2.1.1**	**Versión Cross Platform**	**JMF 2.1.1**
Video: H.261	-	D	D
Video: Cinepak	D	D, E	D
QuickTime (.mov)	lectura / escritura	lectura/escritura	lectura / escritura
MPEG Layer III Audio (. mp3)	sólo lectura	lectura / escritura	lectura / escritura
MPEG layer 1, 2 audio	D	D, E	D, E
MPEG Layer II Audio (. mp2)	sólo lectura	lectura / escritura	Lectura / escritura
Video-only stream	-	D	D
Multiplexed System stream	-	D	D
MPEG-1 Video (. MPG)	-	sólo lectura	sólo lectura
Type 1 & 2 MIDI	-	D	D
MIDI (.mid)	sólo lectura	sólo lectura	sólo lectura
Video: JPEG (411, 422, 111)	D	D, E	D, E
Video: RGB	D, E	D, E	D, E
Video: YUV	D, E	D, E	D, E
Video: VCM**	-	-	D, E
Flash (.swf, .spl)	sólo lectura	sólo lectura	sólo lectura
Macro media Flash 2	D	D	D
GSM (. gsm)	lectura / escritura	lectura / escritura	lectura / escritura
GSM mono audio	D, E	D, E	D, E
Hot media (. mvr)	sólo lectura	sólo lectura	sólo lectura
IBM Hot Media	D	D	D

El formato seleccionado para la captura del video fue JPEG. La decisión de utilizar dicho formato fue tomada mediante la comparación de la calidad (ver Fig. 4.2), requerimientos del CPU (ver Fig. 4.3) y los requerimientos de ancho de banda (ver Fig. 4.4) entre los principales tipos de formatos aceptados por JMF.

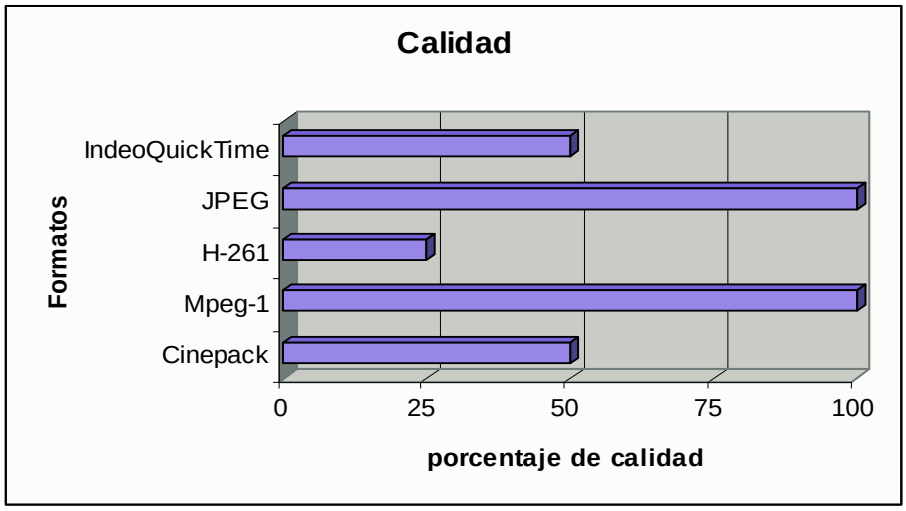


Fig. 4.2 Tabla de comparación de calidad entre formatos

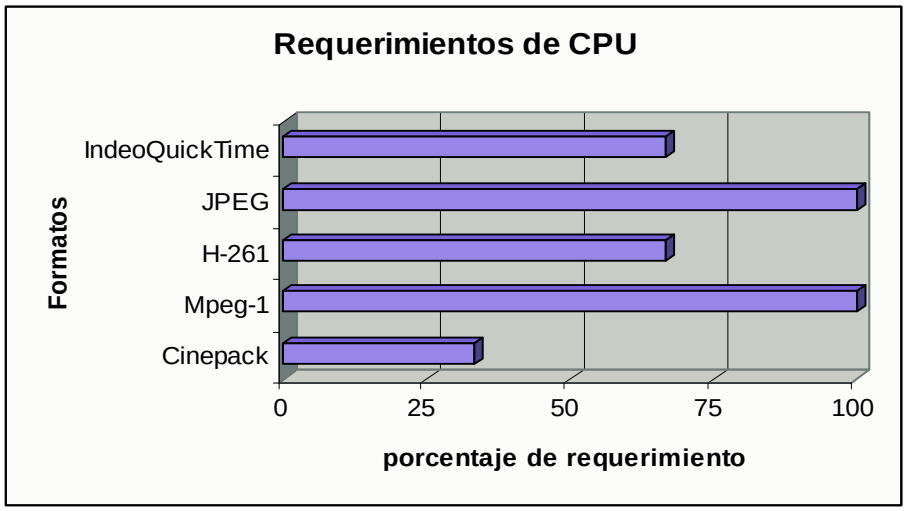


Fig. 4.3 Tabla de comparación de los requerimientos de CPU de cada formato

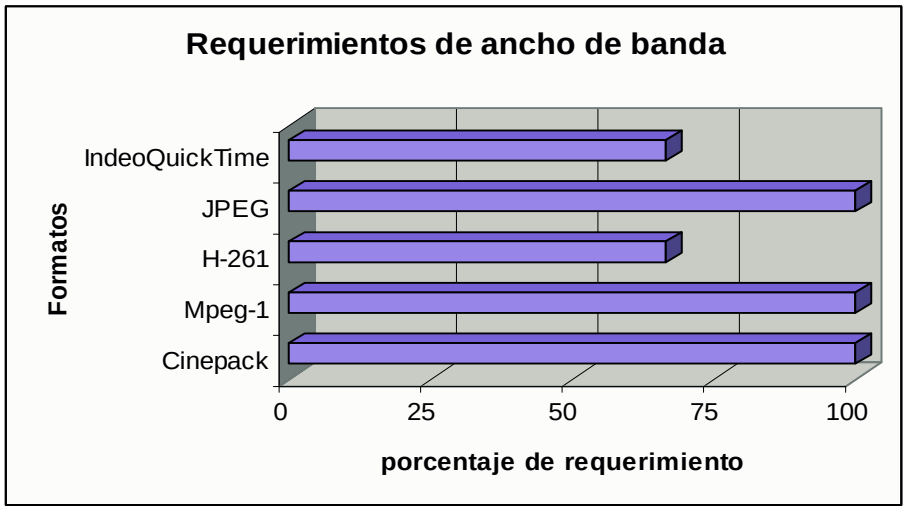


Fig. 4.4 Tabla de comparación de requerimiento de ancho de banda de cada formato

En base a estas graficas se escogió el formato JPEG, fue el que mejor se adaptó a nuestras necesidades. Debido a que se pretende utilizar esta herramienta una vez que este agregada al CU Communicator para el desarrollo de aplicaciones educativas, la calidad de captura y visualización son factores muy importantes dentro de las aplicaciones educativas. La herramienta desarrollada se puede ver en la Fig. 4.5.

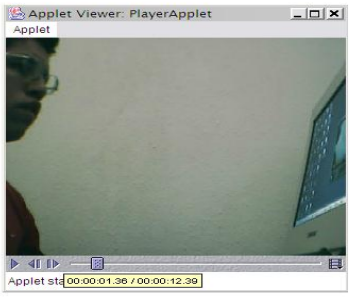


Fig. 4.5 La visualización del video transferido utilizando Applet viewer.

El tiempo que maneja Java Media Framework está establecido en nanosegundos. La interfaz clock es la encargada de dar el control y sincronización de tiempo, el cual nos servirá para controlar un stream de datos multimedia.

Una vez que se capturó el video este deberá ser transmitido, para lo cual se utilizó el protocolo Real Time Transmission Protocol (RTP). Este es un protocolo el cual siempre está recibiendo datos a pesar de que exista alguna perdida de información [Linden deCarmo 99]. Debido a que la transmisión de datos debe ser en tiempo real, se podía utilizar un protocolo el cual no fuera tan confiable viéndolo desde el punto de vista de integridad de los datos. Por este motivo se escogió el protocolo RTP, ya que es un protocolo el cual nos permite enviar y recibir datos a través de Internet.

Mediante RTP podemos identificar el tipo de stream que se emitió, así como el orden en que los datos deberán ser presentados. RTP no envía los datos ordenados, su única función es el de enviarlos. El encargado de ordenarlos es el que los recibe. La transmisión de datos mediante RTP se realiza mediante sesiones. La sesión contiene un IP y dos puertos. Cada cliente que se comunique mediante RTP recibirá el nombre de participante y puede realizar dos tipos de funciones: de emisor, receptor o ambas.

4.2 Transferencia de Audio

Para la captura de audio se utilizó el API de Java Sound, este ha sido una parte de las bibliotecas estándares de la plataforma de Java 2. Encontrado en el paquete de javax.sound.sampled, el API Java Sound proporciona la ayuda para la reproducción y

captura del audio. Además, la biblioteca ofrece software-basado en la mezcla de audio y de dispositivos de MIDI (Musical Instrument Digital Interface). El paquete de javax.sound.sampled consiste en ocho interfaces, doce clases a nivel superior, doce clases internas, y dos excepciones. Para registrar y para reproducir audio, se necesita solamente ocuparse de un total de siete interfaces del paquete. Primero se examina la grabación. El proceso básico de la grabación es de la siguiente manera:

- Se escoge el formato audio en el cual se desea registrar los datos. Esto incluye especificar el índice del muestreo y el número de los canales (mono vs. estéreo) para el audio. Se especifican estas características usando la clase nombrada de AudioFormat. Hay dos constructores para crear un objeto de AudioFormat:
  - AudioFormat(AudioFormat.Encoding encoding,
    float sampleRate, int sampleSizeInBits,
    int channels, int frameSize, float frameRate,
    boolean bigEndian)

  - AudioFormat(float sampleRate, int sampleSizeInBits,
    int channels, boolean signed, boolean bigEndian)

En el primer constructor se puede fijar los encodings del formato de audio, mientras que el resto de variables toman un valor por default. Las codificaciones (encoding) disponibles son ALAW, PCM_SIGNED, PCM_UNSIGNED y ULAW. El encoding por default es usado por el segundo constructor que es PCM [Josua Bloch 01].

- Después de seleccionar el formato audio, se necesita obtener un DataLine. Esta interfaz representa una alimentación de audio de la cual se puede capturar. Se utiliza un subinterfaz de DataLine para hacer la captura. La subinterfaz se llama TargetDataLine. Para obtener el TargetDataLine, es mediante enviar una petición al AudioSystem. Pero cuando se realiza de esta forma, se necesita especificar la información sobre la línea. La especificación debe realizarse en forma de un objeto de DataLine.Info. Se deberá crear un objeto de DataLine.Info que sea específico al tipo de DataLine y al formato audio.

- Ahora ya se tiene la fuente de entrada. Sé podría ver a TargetDataLine como un flujo de entradas. Sin embargo, se requiere de ciertas operaciones antes de que se pueda leer. La operación que se debe realizar es la apertura de la línea usando el método del open(), y después inicializando la línea usando el método start().Después de estas operaciones la línea ya esta lista [Linden deCarmo 99].

Una vez que se ha capturado el audio, el siguiente paso fue el examinar como se reproducirá. Hay dos diferencias dominantes en reproducir audio con respecto a la captura del audio. Primero, cuando se reproduce audio, los octetos vienen de un AudioInputStream en vez de un TargetDataLine. En segundo lugar, se escribe a un SourceDataLine en vez en de un ByteArrayOutputStream. Además de esto, el proceso es igual. Para conseguir el AudioInputStream, se necesita convertir el ByteArrayOutputStream en la fuente del AudioInputStream. El constructor de AudioInputStream requiere los octetos de la corriente de la salida, la codificación audio del formato usado, y el número de los bastidores de la

muestra. Para conseguir el DataLine es similar a la manera a como se obtiene para la grabación audio, pero para reproducir audio, se necesita tener un SourceDataLine en vez de un TargetDataLine. La herramienta utilizada para la captura y reproducción de audio se puede ver en la Fig. 4.6 .

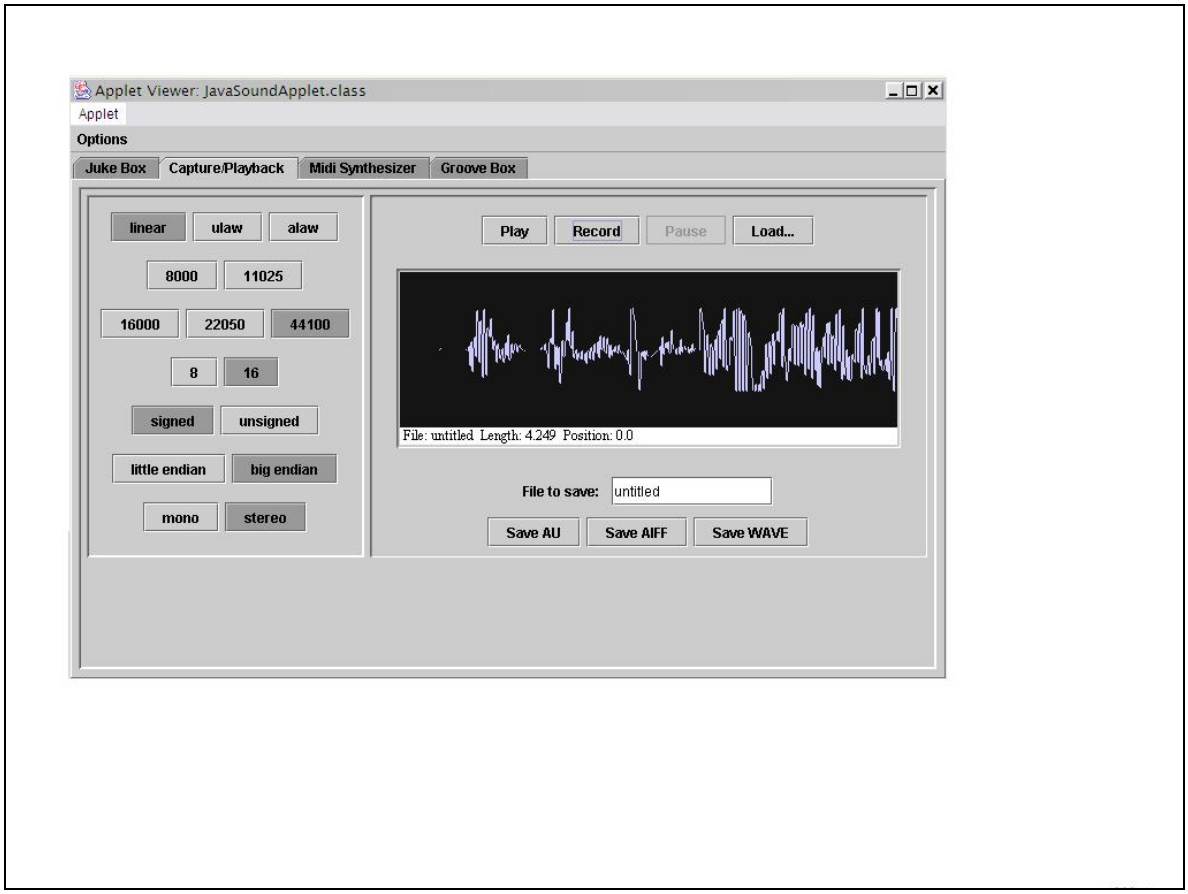


Fig. 4.6 Herramienta utilizada para la captura y reproducción del audio.

Una vez que se analizo la forma en como esta aplicación captura el audio se genero una librería mediante jni (Java Native Interface). Esta librería recibe un arreglo de bytes el cual es transferido al servidor “conexion” . El servidor “conexion ” aunque ya funciona

bajo la arquitectura galaxy , no procesa los datos recibidos debido a que no se generaron las reglas necesarias para que pudiese interactuar con los demas servidores, en este caso con el servidor de audio.

4.3 Transferencia de Imagen

Para la transferencia de imagen, el CU Communicator no tiene integrado todavía una infraestructura mediante la cual se puedan visualizar imágenes recibidas por Web. Es por ello que se tuvo que pensar a futuro, y sobre todo que la aplicación iba a ser utilizada para desarrollar aplicaciones del tipo educativas. Analizando estos puntos se optó por realizar una aplicación, la cual se actualiza en el momento que llega la petición de una nueva imagen, solo recibirá la dirección en donde se encuentra la imagen.

La aplicación esta desarrollada para estar en constante actualización, pensando que en un futuro el CU Communicator tendrá un servidor en donde se produzcan las imágenes. La aplicación se optó por utilizar threads o hilos de ejecución que son segmentos de código de un programa que se ejecutan secuencialmente de modo independiente de otras partes del programa

Los *threads* nos permitirán que el flujo de datos sea dividido en dos o más partes, cada una ocupándose de realizar alguna tarea. En nuestra aplicación el thread se encargará por un lado de visualizar la imagen, y por otro accederá a recursos del sistema para leer el archivo de la imagen. Éste se actualiza cada 3 segundos.

4.4 Transferencia de Texto

Para desarrollar la herramienta que se encargará de transferir el texto se tomó en cuenta la infraestructura del CU Communicator. Es por ello que se desarrolló una aplicación en la cual se puede capturar texto, y también puede recibir y desplegarlo, estando en constante actualización (ver Fig. 4.7). Esta aplicación envía el texto al nuevo servidor "Conexion" del CU Communicator, el cual simula la respuesta con el dato recibido.

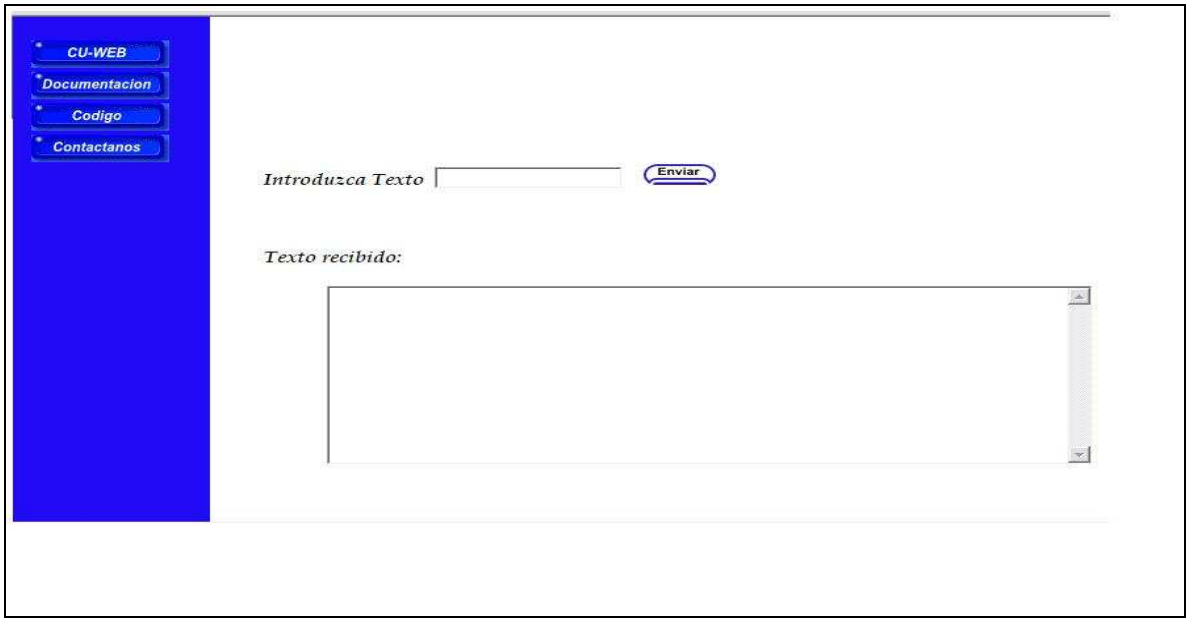


Fig. 4.7 Herramienta de transferencia de texto en donde se obtiene y despliega el texto recibido.

4.5 Desarrollo del servidor de conexión para CU Communicator

Hay 4 pasos básicos que se deben seguir para construir un servidor para el CU Communicator:

1. Establecer las librerías.
2. Escribir las funciones de transferencia.
3. Escribir la función de iniciación del servidor.

4. Escribir e incorporar las declaraciones del servidor.

4.5.1 Establecer las librerías

Este paso es simple. Todas las librerías del Communicator pueden ser cargadas desde $GC_HOME/include/galaxy/galaxy_all.h. Usualmente se incluirá la librería dentro del archivo en el cual se encuentre el servidor de la siguiente manera:

#include “galaxy/gallaxy_all.h”

4.5.2Escribir las funciones de transferencia

Todas las funciones tienen que tener una misma signatura. Esta deberá ser de la siguiente forma:

Gal_Frame Conexion(Gal_Frame frame, void *server_data);

La variable frame es una lista de argumentos es el mensaje de entrada del frame. El server_data es una estructura que encapsula el ambiente de la invocación de la función de transferencia.

Hay 3 regiones básicas en cada función de transferencia:

- Checar el tipo y la descomposición de los mensajes
- Extraer el proceso
- Construcción de la respuesta

Las funciones de transferencia tienen una complejidad adicional del tipo y descomposición de los mensajes, y la construcción de la respuesta, porque los mensajes que recibe están en forma de una llave de conjunto de valores, esta es una secuencia ordenada

de un tipo de elementos. También el código de la función de transferencia necesita hacer algo del trabajo que el compilador debe hacer en otras circunstancias.

4.5.2.1 Descomposición del mensaje

Nosotros necesitamos recibir de entrada un: input_string con la cual generamos un frame. El frame se genera en base al input_string que recibe.

4.5.2.2Construcción de la respuesta

Necesitamos crear un frame para el mensaje de respuesta. Habrá que convertir las tres operaciones en un frame. El nombre del frame no tiene importancia. Dentro de la respuesta se incluye toda la información. La infraestructura de la respuesta deberá tener cuidado de liberar tanto la entrada como la respuesta después de que la función de transferencia sea procesada.

4.5.3 Escribir Función de iniciación del servidor

El siguiente paso es el de escribir la función de iniciación del servidor. Este paso es opcional. La infraestructura del CU Communicator no tomará en cuenta este paso si no está especificado dentro del código no está especificado. La función es llamada de la siguiente manera:

*GalSS*_init_server:

Void GalSS_init_server (GalIO_serverStruct *server, int argc, char **argv);

La función es llamada desde que el servidor se inicia. Esta función se puede usar para cargar la gramática o otros modelos, o también inicializar el estado del servidor.

4.5.4 Escribir e incorporar las declaraciones del nuevo servidor

El siguiente paso es crear e incorporar a librería que es llamada por el nuevo servidor. Este archivo contiene un serie de órdenes, son las siguientes:

- Un nombre para el servidor.
- Un puerto por default para que el servidor reciba y responda peticiones.
- Una función de transferencia disponible.

```
GAL_SERVER_NAME(conexion)
GAL_SERVER_PORT(14960)
GAL_SERVER_OP(enable_input)
GAL_SERVER_OP(reinitialize)
GAL_SERVER_SW_PACKAGE(MITREUtilities)
```

Estas declaraciones anuncian que el servidor es conocido con el nombre de conexión mediante el método GAL_SERVER_NAME. Recibe peticiones en el puerto 14960 mediante el método GAL_SERVER_PORT. También se declaran las funciones de transferencia que en este caso son enable_input y reinitialize mediante el método GAL_SERVER_OP.

Después se tiene que hacer referencia a este archivo desde el servidor conexión usando la siguiente sintaxis dentro del archivo del servidor conexión:

```
#define SERVER_FUNCTIONS_INCLUDE "conexion_server.h"
#define USE_SERVER_DATA
#include "galaxy/server_functions.h"
```

El primer #define declara le nombre del archivo servidor. El segundo #define forza al servidor a usar correctamente las funciones de transferencia. Por ultimo el #include causa que las declaraciones del servidor sean cargadas y los macros sean expandidos en el medio apropiado.

4.6 Agregar nuevo servidor al CU Communicator

Para agregar un nuevo servidor al CU Communicator se deben seguir los siguientes pasos:

1. Creación del Makefile
2. Generar archivo para comunicación con el Hub
3. Modificar Archivo UI_SLS_FESTIVAL.CSH

4.6.1 Creación del Makefile

Para poder agregar el nuevo servidor lo primero es construir un Makefile para compilar el servidor. La infraestructura del Galaxy Communicator provee un sofisticado Makefile utilizado para compilar los servidores del Communicator. La siguiente es una versión simplificada del Makefile del servidor conexión:

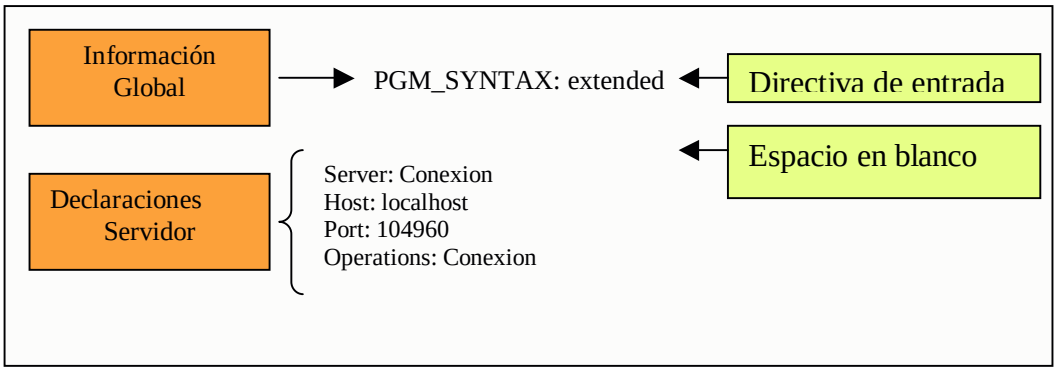
```
#Especifica el nombre del Makefile
MAKEFILE = conexion.make

# especifica la localizacion de la distribucion del comunicator
ROOT_DIR = /home/CU/packages/GalaxyCommunicator-3.2beta2
MITRE_ROOTDIR = /home/CU/packages/GalaxyCommunicator-3.2beta2/contrib/MITRE
TEMPLATES = $(ROOT_DIR)/templates

#Los archivos de archos.make contienen información acerca del sistema operativo y la plataforma

include $(TEMPLATES)/archos.make
#El nombre del servidor ejecutable que se desea compiler

SERVER = Conexión

#Poner el ejecutable en EXECDIR (si no se especifica, el ejecutable se direccionara al directorio bin principal del Galaxy Communicator)
```

```
EXECDIR = bin/

#lista de todos los archivo .c

SOURCES = Conexion.c
#incluir las reglas del Galaxy Communicator

include $(TEMPLATES)/rules.make

#incluir automáticamente el código generado  por la dependencia del archivo.


ifneq ($(findstring clean,$(MAKECMDGOALS)),clean)
include $(MAKEFILE).depend
endif
```

4.6.2 Generar archivo para comunicación con el Hub

Hay dos tipos de información que debe contener el archivo (véase Fig. 4.8):

- Información Global
- Declaraciones del servidor



En la Fig. 4.8. Dependencias envueltas en la declaración del servidor

.

4.6.2.1 Información global y Declaraciones del servidor

PGM_SYNTAX

Mediante la infraestructura del Galaxy Communicator se extiende la sintaxis del Hub y con esto se logra hacer más consistente todo el funcionamiento. Sin embargo, con estas modificaciones no es compatible para poder hacer cambios después. Por ello se escoge un directivo de entrada explícito. Siempre empezara el programa con esta entrada.

VALORES ATOMICOS DEL STRING

El valor del PGM_SINTAX es un conjunto de otras directivas las cuales son String. Por lo regular las directivas tienen la misma forma mediante la cual se crean los frames.

SERVICE_TYPE:

Cada SERVICE_TYPE tiene un nombre, El cual es el valor de del SERVICE_TYPE: Conexion.El bloque que empieza con un SERVICE_TYPE: Directiva también contiene unas OPERATIONS: Directiva y un CLIENT_PORT: Directiva, entre las principales. El valor de la directiva es un string.

CLIENT_PORT:

Es un SERVICE_TYPE que esta en espera de recibir una petición para realizar las conexiones necesarias.

OPERATIONS

Cada service_type declara una serie de operaciones, las cuales pueden ser operaciones de transferencia que son desarrolladas como se explicó anteriormente. El Hub escogerá un servicio que es proporcionado mediante las reglas declaradas dentro del archivo del Hub que corresponden a un tipo de operación.

SERVICE_PROVIDER

Cada bloque de un service_provider corresponde a un servicio proporcionado al contactar al Hub. El valor del service_provider es una directiva, la cual es el nombre de un service_type implementado. Desde cada service_type se definen las operaciones soportadas. Para ejemplificar se puede ver la sintaxis del servidor Conexion:

SERVICE_TYPE: Conexion

OPERATIONS: Conectar

SERVICE_PROVIDER: Conexion

HOST: localhost

PORT: 16490

HOST

Cada service_provider necesita especificar la localización para que el Hub pueda contactarlo. El valor del HOST es un string que llama a la maquina donde los service_provider recibe peticiones.

PORT

El valor del PORT es un entero que corresponde a un numero de puerto que el service_provider recibe peticiones.

Para declarar la localización completa del servidor se deberá especificar una directiva la cual es un string "host": "port" . Mediante esta directiva el www contacta al service_provider para el service_type Conexion el cual recibe peticiones en el puerto 16490 vea Fig.4.9.

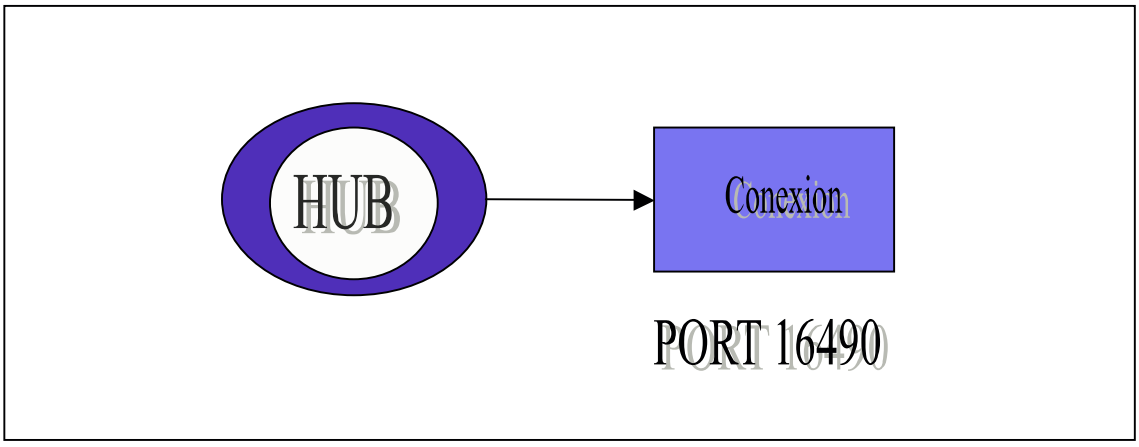


Fig. 4.9 Llamado del servidor Conexión en el puerto 16490

4.6.3 Modificar Archivo UI_SLS_FESTIVAL.CSH

Para poder ver el nuevo servidor dentro de la interfaz del CU Communicator se tiene que agregar dentro del archivo UI_SLS_FESTIVAL.CSH el siguiente comando:

```
-T "Conexion" --start --open --input_line -c "$TOOLS_ROOT/run_server $MITRE_SERVERS/conexion -prog_name main" \
```

Con el comando start sé inicializa el servidor, después que se inicializa con open se despliega una interfaz visible. Por último con el comando input_line se despliega un campo de escritura. La nueva interfaz del CU Communicator se puede ver en la Fig.4.10. .

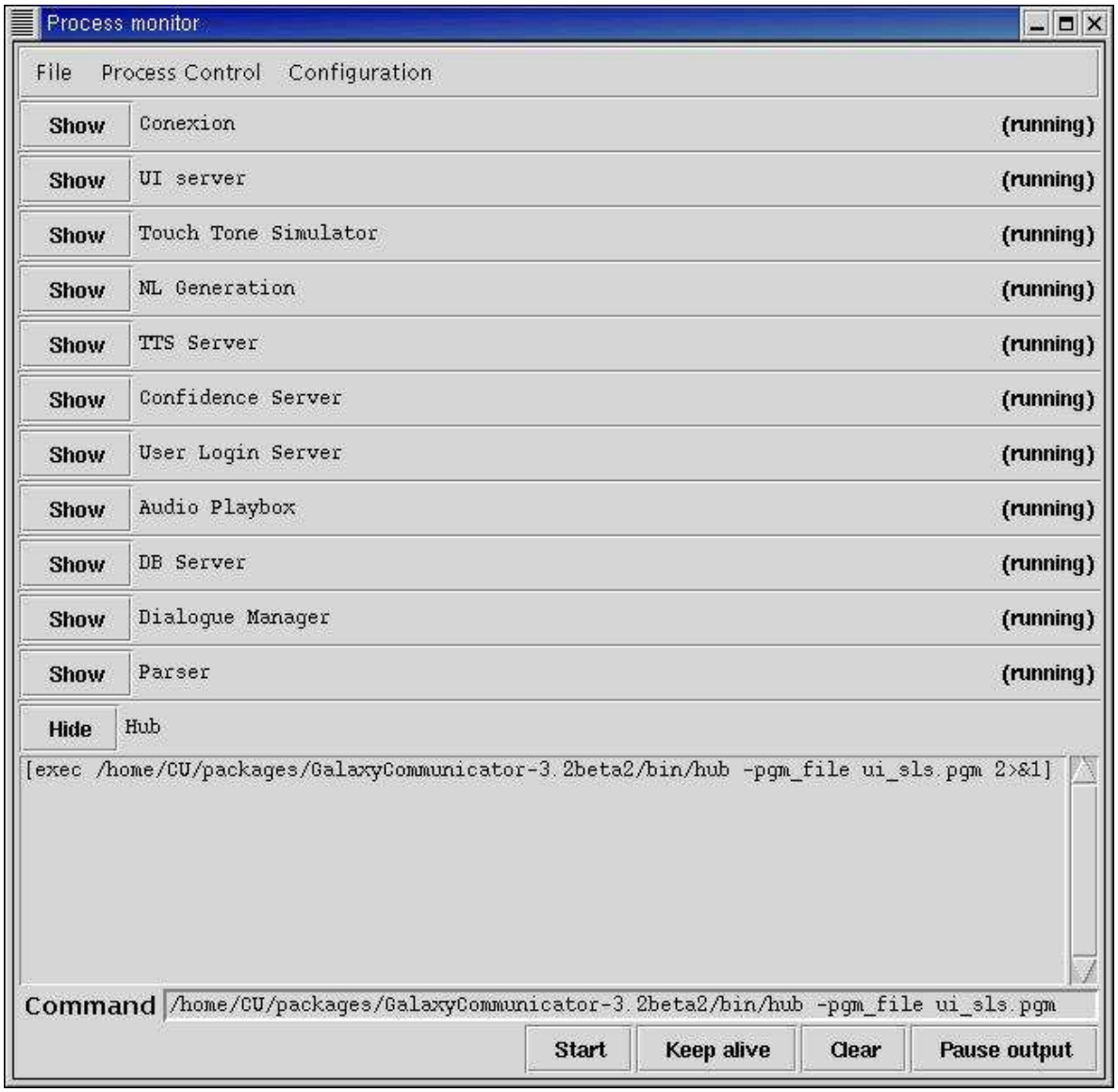


Fig. 4.10 Interfaz con el nuevo servidor conexión agregado

4.7 Comunicación entre el Servidor Conexion y las herramientas desarrolladas

La comunicación entre el servidor conexión y las herramientas desarrolladas en java se realizó mediante JNI (Java Native Interface). JNI es una herramienta que nos sirve para acceder de Java a códigos nativos. Debido a que el servidor conexión aunque utiliza una

sintaxis específica, esta desarrollada en el lenguaje de programación C, y de esta forma podemos hacer que interactué el servidor del CU Communicator con las herramientas desarrolladas en Java.

4.8 Conclusión

En este capítulo describimos las aplicaciones, sintaxis y lenguajes que se utilizaron para desarrollar las herramientas y el servidor “Conexion”. También la forma en como se integró el servidor “Conexion” a la arquitectura Galaxy y la manera en como se comunica el servidor “Conexion” con las herramientas desarrolladas.