Sistema de gestión para un servidor de video bajo demanda*

Carlos Varela, Víctor M. Gulías, Alberto Valderruten, Carlos Abalde
LFCIA, Departamento de Computación, Universidade da Coruña
Campus de Elviña, 15071 A Coruña, España
email: {cvarela,gulias,valderruten,carlos}@dc.fi.udc.es

RESUMEN

En este artículo se describe un sistema de gestión para un servidor de vídeo bajo demanda usando SNMP. Permite por una parte la monitorización del servidor VoDKA, un sistema que proporciona servicios de vídeo a usuarios que pueden solicitar extractos de información en cualquier momento, y por otra parte soporta la gestión de cambios en el servidor, todo ello usando el estándar de gestión SNMP. Dicha gestión puede así realizarse a través de herramientas externas con independencia de la que se utilice. Para su implementación se explotan las posibilidades de Erlang/OTP, lenguaje funcional para desarrollar aplicaciones de tiempo real blando, tolerantes a fallos y distribuidas, usando el paradigma de paso de mensajes.

Palabras clave: Monitorización, Instrumentación, Programación Funcional, Erlang/OTP, Vídeo bajo Demanda, Evaluación del Rendimiento.

ABSTRACT

In this paper we describe a management system for a video-on-demand server based on SNMP. It allows the monitoring of the VoDKA server, a system which offer video services to users that can demand for data streams at any time, and also provides the management of the changes on the server, using the standard SNMP. Thus, the management can be done using any standard tool. The concurrent functional language Erlang/OTP has been chosen for the development because of its suitable features for the implementation of soft real-time fault-tolerant distributed processing applications, using message-passing.

Keywords: Monitoring, Instrumentation, Functional Programming, Erlang/OTP, Video on Demand, Performance Evaluation.

*Trabajo parcialmente financiado por CICyT, Proyecto TIC 2002-02859, *VRDADER: Verificación, Rendimiento y Disponibilidad de Aplicaciones Distribuidas en Entornos Reales* y por *Xunta de Galicia,* Proyecto PGIDT02TIC00101CT, *Diseño, construcción y validación de un sistema jerárquico de almacenamiento de alta capacidad, de bajo coste de adquisición y funcionamiento: CheapTB.*

1. Introducción

Un servidor de vídeo bajo demanda (servidor VoD, por sus iniciales en inglés) proporciona servicios de vídeo al usuario final, que puede solicitar la visualización de una secuencia de vídeo (películas bajo demanda, tele-enseñanza, compras desde el hogar, servicios de noticias interactivas, etc.) en cualquier momento, sin una previsión o planificación pre-establecida. La creciente demanda de este tipo de servicios condiciona el desarrollo de servidores VoD con un alto nivel de escalabilidad, tanto en capacidades de almacenamiento como en ancho de banda. Para ello ha propuesto en [1, 2] un sistema de almacenamiento jerárquico basado en un cluster de componentes de consumo con Linux. En su implementación se ha usado el lenguaje funcional concurrente Erlang [3], desarrollado por Ericsson, debido a las facilidades que ofrece para escribir aplicaciones de tiempo real complejas, tolerantes a fallos y distribuidas usando el paradigma de paso de mensajes.

Como base no sólo para la gestión del sistema distribuido durante su fase operacional, sino también para las etapas de validación y verificación del proceso de evaluación del rendimiento que se integra en la metodología de desarrollo, resulta fundamental disponer de los mecanismos de monitorización apropiados. Dichos mecanismos deben permitir la observación del sistema a diferentes niveles de detalle y se han de diseñar de forma que perturben en la menor medida posible el comportamiento funcional del sistema. Se ha recurrido a un conocido patrón de diseño, el *Observador* (*Observer* [4]), en el cual se desacopla el sujeto observado de aquellos agentes que observan sus evoluciones. Para afectar lo menos posible a la carga de cada uno de los sujetos, se introduce un agente mediador (*Mediator* [4]) local encargado de registrar los observadores, recibir las notificaciones de los sujetos que cohabitan en el nodo y distribuir los eventos a los observadores relevantes [5].

Pero para gestionar un servidor de vídeo esto no es suficiente. En este artículo se exploran las posibilidades de Erlang/OTP para el desarrollo de una infraestructura de gestión que pueda ser integrada en el servidor de vídeo bajo demanda VoDKA. El sistema construido permite la monitorización del estado de los recursos, la modificación del estado y la generación de datos estadísticos.

2. Gestión de redes

Las tecnologías de gestión de red se centran en la monitorización, interpretación y control de los comportamientos de la red. Los estándares de gestión de red buscan integrar estas funciones dentro de la heterogeneidad de dispositivos y protocolos que existen en los sistemas de red actuales.

Se pueden analizar los fundamentos de la gestión de red centrándose en las dos principales operaciones involucradas: monitorización y control. Aunque son conceptos de la gestión de redes, en este trabajo se van a aplicar a un servidor de *streaming* existente.

Monitorización

La monitorización se realiza observando y analizando el estado y el comportamiento de los sistemas. La información de *monitorización* puede clasificarse en *(i) estática*, cuando caracteriza la configuración actual y sus elementos, que varían con poca frecuencia, *(ii) dinámica*, cuando está relacionada con eventos que suceden en el sistema como un cambio de estado, y *(iii) estadística*, que incluye información que puede ser inferida a partir de la dinámica como la media de paquetes transmitidos por unidad de tiempo en un sistema.

En un sistema de monitorización se distinguen los elementos que se pueden ver en la figura 1. Los cuatro

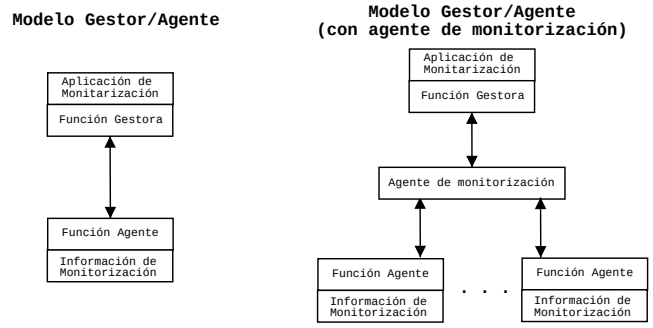

Figura 1: Esquema de un sistema de monitorización

componentes principales del sistema son: La *aplicación de monitorización*, que incluye las funciones de monitorización que son visibles para el usuario. La *función gestora*, que realiza la función básica de recuperar información de monitorización para la aplicación. La *función agente*, encargada de recolectar y almacenar información de uno o más elementos y comunicar la información al monitor. En cuanto a la *información de monitorización*, es la información que representa los recursos y sus actividades.

Opcionalmente, como se puede ver en la segunda parte de la figura, puede haber un módulo con información estadística, un *agente monitorizador*, que genera agregaciones y análisis estadísticos de la información. Si no está con el gestor, toma el papel de agente para comunicarse con él.

La información que es útil para la monitorización es recogida y almacenada por los agentes y se hace disponible para los sistemas de gestión, usando para esto último dos técnicas posibles: *polling* y *event-reporting*. El *polling* es una interacción basada en mensajes de petición/respuesta entre el gestor y los agentes. El gestor solicita parte o toda la información que posee un agente. El agente espera peticiones y devuelve información almacenada en su base de información de gestión. Un sistema de gestión puede usar *polling* para aprender acerca de la configuración de un sistema, para obtener información periódica o para investigar algo en detalle después de ser alertado de un problema. También es utilizado para generar respuestas a preguntas específicas de un usuario. Con el *event-reporting* la iniciativa, y con ello la complejidad, es del agente que, periódicamente o cuando se haya cumplido alguna condición, envía información al gestor. Aquí el gestor sólo tiene que configurar la actividad del agente, el periodo de recepción de informes o la condición para el envío de informes, y ponerse en espera.

Control

El *control* está vinculado con la modificación de los componentes de configuración que pueden provocar acciones predefinidas que son realizadas por esos componentes. El control se estructura en dos áreas: *control de la configuración* y *control de la seguridad*. El control de la seguridad se encarga de ofrecer seguridad en los ordenadores de una red para los recursos gestionados. El control de la configuración se centra en la definición de la información de configuración, el cambio de valores de atributos, la definición y modificación de relaciones entre los elementos del sistema, la inicialización y terminación de operaciones en el sistema.

3. SNMP

El término SNMP, *Simple Network Management Protocol*, se usa para referirse a una colección de especificaciones de gestión de red que incluyen al protocolo en sí, la definición de estructuras de datos y conceptos asociados [6]. El modelo de gestión de red usado para gestionar redes TCP/IP incluye los elementos *gestor*, *agente*, *base de datos de gestión (MIB)* y *protocolo de gestión*, relacionados entre sí según puede verse en la figura 2.

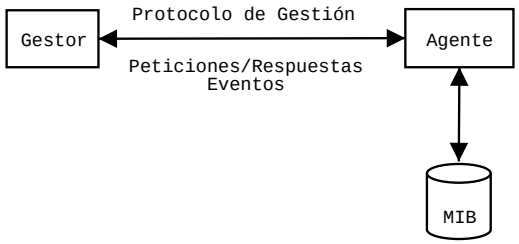

Figura 2: Conceptos básicos de los estándares de gestión

El *gestor* sirve como interfaz para la persona encargada de la gestión. Como mínimo debe incluir: (i) un conjunto de aplicaciones para el análisis de datos, recuperación de fallos ..., (ii) una interfaz para monitorizar y controlar la red, (iii) la capacidad de traducir los requerimientos actuales en la monitorización y control de los elementos remotos en la red y (iv) una base de datos con información de todas las entidades en la red. Sólo los dos últimos aspectos están sujetos a estandarización SNMP.

El *agente* es el otro elemento activo en el sistema de gestión de red. Suele ejecutarse en el dispositivo a gestionar o en una estación con acceso a los recursos gestionados. Responde a peticiones del gestor y puede asíncronamente enviarle información acerca de algún evento importante.

Los recursos de una red pueden ser gestionados representando esos recursos como objetos. La colección de objetos se llama *base de datos de gestión (MIB)*. Todos los objetos gestionados en el entorno SNMP están dispuestos en una estructura de árbol. Los objetos hoja son los objetos que se gestionan, donde cada uno representa algun recurso, actividad o información relativa a lo que tiene que ser gestionado. Además asociado con cada tipo de objeto en una MIB hay un identificador del tipo ASN.1 *Object Identifier* (en adelante, Oid) que es una secuencia de enteros. El identificador sirve para nombrar los objetos y para identificar la estructura de los tipos de objetos. Empezando desde la raiz del árbol, cada valor que compone el identicador de un objeto identifica una rama en el árbol. El gestor realiza la monitorización recuperando los valores de los objetos de la MIB. Asímismo puede realizar acciones o cambiar la configuración modificando valores de variables específicas. Están definidas en este momento dos MIBs estándar, conocidas como MIB-I y MIB-II [7].

Los agentes y el gestor se comunican por medio del *protocolo de gestión de red*. El protocolo utilizado para la gestión de TCP/IP es el *Protocolo Simple de Gestión de Red*, SNMP[8, 9], que incluye las capacidades siguientes: (i) *get*, para recuperar el valor de objetos en el agente; (ii) *set*, para establecer valores de objetos en el agente; y (iii) *trap*, para notificar al gestor eventos significativos.

En la revisión SNMPv3 se mejora la seguridad creando el modelo de seguridad basado en el usuario, USM[10], y el modelo de control de acceso basado en vistas, VACM[11].

4. Arquitectura de gestión

La arquitectura propuesta está basada en el modelo gestor/agente con agentes de monitorización. En cada nodo se instala un agente SNMP que recogerá la información estática y dinámica y atenderá las peticiones del gestor. Además, y para evitar que el gestor tenga que hacer peticiones a todos los nodos, se plantea construir una jerarquía de agentes de modo que cada agente contenga toda la información que posean los agentes que se encuentren por debajo de él. El agente también puede generar información estadística al margen de que otros sistemas puedan igualmente generarla. Esta arquitectura está representada en la figura 3.

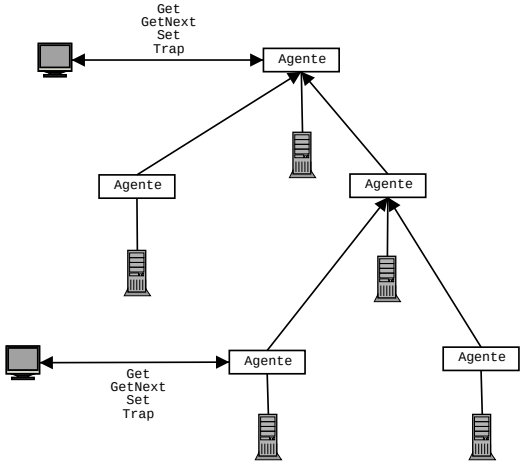

Figura 3: Arquitectura de gestión

En cada nodo el agente se comunica con el sistema VoDKA que allí se encuentre para conseguir la información que deba mantener en su MIB. El gestor es el encargado de pedir los datos a los agentes de forma periódica para actualizar el estado del sistema.

Un sistema VoDKA normalmente está desplegado en varios nodos por lo que será necesario más de un agente, uno en cada nodo. Los agentes mantienen la información en una MIB. Por lo tanto, los elementos que participan en el sistema de gestión son:

- El *agente*, que recoge información de VoDKA y contesta peticiones SNMP;
- La *MIB*, la base de información del agente, en la que guarda los datos que se necesitan para la gestión de VoDKA;
- El servidor *VoDKA*, el elemento que está siendo gestionado;

- El *Cliente SNMP*, encargado de consultar o modificar información en el agente.

Estos elementos se representan en el diagrama de clases de la figura 4. Queda por ver cómo es la estructura de la MIB y para eso hay que analizar la estructura de VoDKA para extraer su modelo de datos.

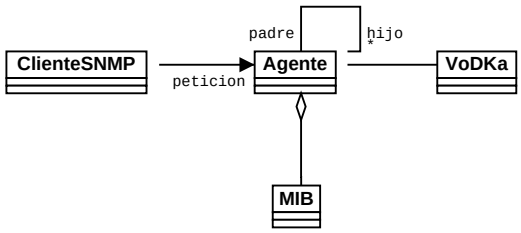

Figura 4: Diagrama de clases conceptual

5. Diseño del agente

El diseño del agente se realizó en varias etapas tras las cuales se fue incrementando la funcionalidad del mismo:

1. A partir del toolkit de Erlang para la creación de agentes SNMP [12] y del mecanismo de paso de mensajes de Erlang, se construyó un agente capaz de entender mensajes SNMP y de gestionar su estado interno de una forma sencilla.
2. Se diseñó un mecanismo que permitiese obtener la información que contiene el sistema VoDKA en un nodo y lo almanacenase en la MIB del agente.
3. Se diseñó la forma de crear la jerarquía de agentes.
4. Se introdujo el mecanismo que permite modificar el estado del servidor.
5. Por último se incluyó la posibilidad de dar de alta nuevas instancias en la MIB.

Primer agente

Para conseguir el primer objetivo lo primero que se necesita es un mecanismo cómodo que permita la inserción de valores en la MIB. Por cómodo se entiende que no sea necesario especificar los Oids de los objetos, ni conocerse el índice de las columnas de las tablas, como se especifica en los mensajes SNMP, sino que sea suficiente con dar su nombre. Algo del estilo:

```
insertar_fila(
   tabla_almacenamientos,
   [{pidTablaAlmacenamientos, Pid},
    {descripcionTablaAlmacenamientos, Nombre},
    {siguienteTablaAlmacenamientos, Siguiente},
    {estadoTablaAlmacenamientos, Estado},
    {tipoTablaAlmacenamientos, Tipo}]
);
```

para insertar una fila en una tabla, o

```
obtener_valores_fila_tabla(tabla_medios, Clave, [tamanoTablaMedios])
```

para obtener el valor de una columna en una tabla conociendo el valor de la clave.

Para ello se construye una capa *software* por encima de los módulos `snmp` y `snmp_index`, que forman parte del SNMP toolkit de Erlang. Esta capa está formada por tres módulos que se pueden ver en la figura 5.

El módulo `tabla_mib` proporciona una implementación de una tabla para la MIB, y es la encargada de realizar las operaciones de modificación, inserción y validación de valores en la tabla. Este módulo es un módulo general y al crear una tabla con él hay que configurar ciertos parámetros que se pasan a la función que crea la tabla. Las funciones que exporta este módulo se pueden dividir en dos grupos: las que no usan Oids ni índices y las que sí los manejan.

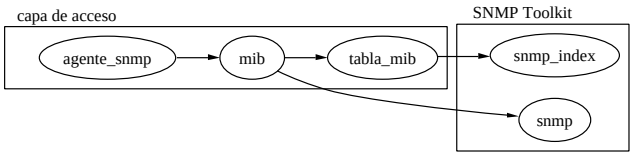

Figura 5: Relación entre la capa de acceso y el SNMP Toolkit

- **Funciones que no manejan Oids ni índices:**
  - `insertar_fila/2`: recibe como parámetros una lista de pares {`NombreColumna, Valor`} que representa una fila y una TablaMib. Inserta la fila en la tabla. Hay que tener cuidado de insertar **todas** las columnas (no importa el orden).
  - `eliminar_fila/2`: recibe como parámetros una tupla con los valores de la clave de una fila que será eliminada y una TablaMib.
  - `actualizar_fila/3`: recibe como parámetros una tupla con los valores de la clave, una lista como la que recibe la función `insertar_fila/2` y una TablaMib. Lo que hace es actualizar los valores de las columnas indicadas en la fila de la tabla que tenga esa clave.
  - `actualizar_fila_acumulando/3`: igual que `actualizar_fila/3` sólo que en lugar de sustituir los valores por los dados, éstos se suman a los ya existentes.
  - `obtener_fila_tabla/2`: recibe una clave y una TablaMib. Devuelve una lista con los valores de las columnas según el orden en que se definieron.
  - `obtener_valores_fila_tabla/3`: Igual que la anterior pero recibe un parámetro más para indicar los nombres de las columnas que se desean.
- **Funciones que manejan Oids y/o índices:**
  - `create/5`: recibe como parámetros un *atom()* que representa el nombre de la tabla, una clave del tipo *key_types()* de snmp_index, una lista con los índices de las columnas que forman el índice de la tabla, un *integer()* con el índice de la columna que es de tipo RowStatus y una lista de elementos de tipo *atom()* que representan los nombres de las columnas en el orden que el que están definidas en la MIB. Devuelve una TablaMib cuya representación interna no es necesario conocer.
  - `inserta_valores/3`: Recibe como parámetros un key() de snmp_index que representa el índice de la fila, una lista de tuplas {`Indice, Valor`} que representan los valores para cada columna y una TablaMib. Inserta las columnas en la tabla aplicando la función de inserción indicada al crearla. Devuelve {`noError, ValoresNoInsertados`} o {`genErr, Indice`} donde `ValoresNoInsertados` son los elementos de la lista de columnas que no se insertaron e `Indice` es el índice de la primera columna donde hubo un error.
  - `obtener_fila/2`: dado un Oid y una MibTable obtiene una lista con los valores de la fila correspondiente a ese Oid. Devuelve {`ok, {Fila, ColumnasIndice, RowStatusCol, NumeroColumnas}`} o `undefined` donde *Fila* es una lista con los valores de la fila, *ColumnasIndice* en una lista con los índices de las columnas que forman el índice, *RowtatusCol* es el índice de la columna con el RowStatus y *NumeroColumnas* es el número de columnas devueltas.
  - `obtener_siguiente_fila/2`: igual que la anterior pero obtiene la siguiente fila en orden lexicográfico.
  - `columna_row_status/1`: Obtiene el índice de la columna con el RowStatus.

El segundo grupo de funciones está pensado para ser usado por las funciones de instrumentación de la MIB.

Las funciones de instrumentación necesarias para el agente genérico proporcionado por el *toolkit* se definieron en el módulo `mib`. Este módulo, además, proporciona operaciones para manejar los objetos de las tablas sin necesidad de conocer su Oid. El módulo hace uso de tablas definidas con el módulo `tabla_mib` manteniendo una colección de ellas en su estructura interna. Para indicar la estructura de las tablas que se almacenan en la MIB se define un fichero que debe llamarse *tablas.txt* y está formado por términos ERLANG de la forma siguiente, utilizando variables que tienen el mismo significado y forma que las de la función `tabla_mib:create/9`:

`{NombreTabla, Clave, ColumnasIndice, ColumnaRowStatus, NombresColumnas}`

El módulo `mib` exporta las funciones siguientes, que buscan todas ellas una tabla en su estructura interna y delegan la operación correspondiente en ella:

- `insertar_fila/2`: recibe el nombre de una tabla y una lista de pares `{NombreColumna, Valor}` que representa una fila.
- `eliminar_fila/2`: recibe el nombre de una tabla y una tupla con los valores de la clave de una fila que será eliminada.
- `actualizar_fila/3`: recibe el nombre de una tabla, una tupla con los valores de la clave y una lista como la que recibe la función `insertar_fila/2`.
- `actualizar_fila_acumulando/3`: igual que `actualizar_fila/3`.
- `obtener_fila_tabla/2`: recibe el nombre de una tabla y su clave.
- `obtener_valores_fila_tabla/3`: igual que la anterior.

Además, el módulo exporta las funciones de instrumentación para el agente genérico del *toolkit* del ERLANG. Estas funciones son:

- `tabla(get, RowIndex, Cols, NombreTabla)`
- `tabla(get_next, RowIndex, Cols, NombreTabla)`

El módulo `agente_snmp` proporciona una fachada. A través de esta fachada se inicia el agente genérico de ERLANG y la MIB. Además permite un arranque remoto del agente. Este módulo exporta las funciones siguientes:

- `up/2`: recibe como primer parámetro un nodo y como segundo una lista de opciones, e inicia el agente en el nodo indicado.
- `insertar_fila/2`, `eliminar_fila/2`, `actualizar_fila/3`, `actualizar_fila_acumulando/3`, `obtener_fila_tabla/2`, `obtener_valores_fila_tabla/3`: delegan todas ellas en el módulo `mib`.

Consiguiendo información de VoDKA

Hay dos alternativas claras para obtener información del sistema VoDKA. Cada vez que le llega una consulta al agente, éste busca la información en el sistema por medio de la interfaz de los servidores genéricos (RGSs) que pueden hacer introspección para recuperar propiedades definidas dentro de ellos. O bien, se registra un observador en el sistema que escuche los eventos y almacene dicha información en el agente. La primera alternativa supone que cada vez que se realice una petición se explore todo el sistema (para sacar una "foto" del estado) y obtener la respuesta a partir de los datos recogidos. La segunda, en cambio, supone que se estén enviando datos al agente incluso en momentos en que esos datos no interesan porque no van a ser examinados.

Dado que el modelo de interacción con el agente es del tipo **petición/respuesta**, se puede suponer que habrá muchas consultas al agente por lo que la primera alternativa degradaría mucho el rendimiento del sistema, al ser una operación muy costosa consultar el estado de todos los RGSs, particularmente en un sistema distribuido. Se optó por registrar un observador y, para hacer esto, se desarrolló otra capa que se puede ver en la figura 6.

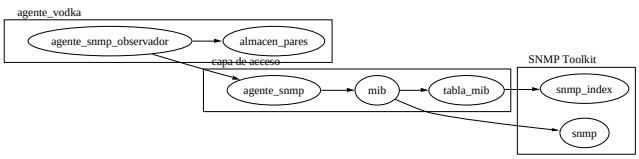

Figura 6: Módulos del agente

El módulo `agente_snmp_observador` proporciona una fachada que tiene el mismo comportamiento que `agente_snmp` y además se registra en el monitor de VoDKA para escuchar los eventos del sistema. Para obtener el mismo comportamiento que `agente_snmp` delega todas sus operaciones en ese módulo.

El módulo `almacen_pares` almacena pares de valores del tipo {`Valor1, Valor2`} y permite búsquedas sobre ellos. Su utilidad en el agente es la de almacenar los pares {*Pid,* {*Nodo, Nombre*}} en el caso de los *traders* y para los *pipes* que atienden una conexión {*PidPipe,* {*NodoConexion, PidConexion*}}. Este almacenamiento se necesita debido a que los eventos que envía el sistema al agente mandan el **pid** del proceso y las claves de los *traders* son el nodo en el que se encuentran y el nombre del *trader*. En el caso de los *pipes* interesa una forma rápida de conocer si un *pipe* atiende a una conexión para actualizar el progreso de la misma.

Para ver este comportamiento veremos dos ejemplos. En el primer ejemplo vemos qué sucede al darse de alta un almacenamiento en el sistema (figura 7).

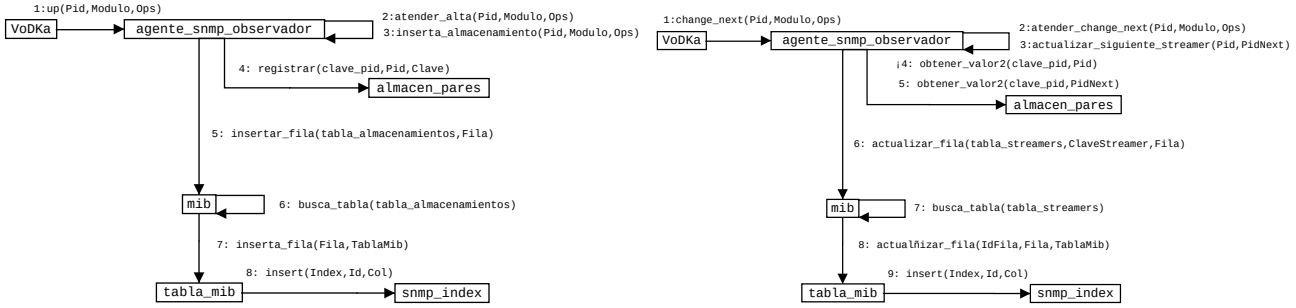

Figura 7: Alta de un almacenamiento y cambio del trader siguiente de un streamer

El sistema informa de un alta y envía al agente un evento de tipo **up** que contiene información como el pid, nombre del módulo y otras cosas que dependen del tipo de *trader* que se dé de alta. El agente recibe el evento y, tras analizarlo, concluye que es un almacenamiento y pasa a insertarlo en la MIB, pero antes de insertarlo registra el pid en un almacén de pares llamado *clave_pid* donde se guardará el pid junto con la clave del *trader* en la tabla de almacenamientos. Luego invoca la operación `insertar_fila` sobre la MIB indicando la fila y la tabla sobre la que quiere insertar, que en este caso es la tabla `tabla_almacenamientos`. Una vez localizada le indica a la tabla la fila que tiene que insertar, la cual realiza la operación a través del módulo `snmp_index`.

En un segundo ejemplo examinamos qué sucede cuando se modifica el *trader* siguiente de un *streamer* (figura 7). Aquí el sistema informa al agente de un cambio de *trader* siguiente en un *trader*. Después de analizar el evento llega a la conclusión de que el *trader* es un *streamer* y procede a actualizar la información que posee de éste, pero para lograr este fin necesita conseguir las claves de los dos *traders* y se las pide al almacén de pares. Un vez que conoce las claves le pide a la MIB que actualice en la tabla de *streamers* la fila que tiene la clave del *streamer*. La MIB busca la tabla y actualiza el contenido de la fila.

Jerarquía de agentes

Lo normal es que un sistema VoDKA se encuentre desplegado en varios nodos. Para minimizar el número de agentes a consultar para conocer el estado del sistema es conveniente poder definir jerarquías de agentes que permitan que la información de los agentes más bajos en la jerarquía se encuentre también en sus agentes inmediatamente superiores.

Para crear la jerarquía se mantienen referencias internas al padre que se pasan en el parámetro de opciones al crear el agente en la función `up/2`. En la lista de opciones hay que incluir la tupla {`parent, Pid`} donde *Pid* es el pid del agente padre. Una vez creado el agente éste informará al padre cada vez que altere el contenido de su MIB para que realice el mismo cambio en la suya. Podemos ver un ejemplo de esto en la figura 8. Aquí el agente que se encuentra en el nodo 1 conoce la información de todo el sistema por lo que un agente conectado a este nodo podría monitorizar todo el sistema. Un agente conectado al nodo 2 no sabría nada acerca del nodo 1 ni del nodo 3.

De este modo se consigue minimizar el número de consultas necesarias en un sistema grande; además como la notificación de los eventos es local en el nodo la información que se pasa a los padres (que normalmente se encontrarán en nodos distintos) ya es una información filtrada y en la mayor parte de los casos más pequeña que la información que envía el evento. Esto proporciona una alternativa mejor que registar varios observadores

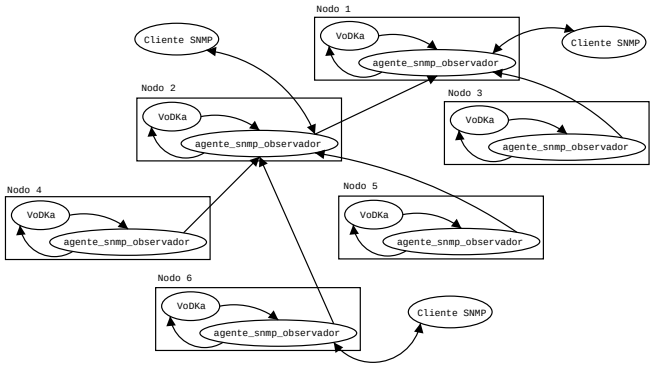

Figura 8: Configuración en varios nodos

en distintos nodos para un solo agente, al reducir el tráfico de gestión en la red.

Esta forma de mantener la información en los nodos introduce una restricción en las operaciones de inserción de nuevas instancias que viene dado por la forma en la que insertan datos usando SNMP. En primer lugar, con el sistema explicado hasta ahora no hay nada que impida que desde un nodo se dé de alta un *trader* en otro nodo. Por otro lado una de las formas comunes de dar de alta una fila de una tabla en la MIB de un agente es invocar operaciones `get` o `getnext` hasta encontrar una clave no utilizada. Si se hace esto en el nodo A y se da de alta una fila en otro nodo B puede ser que el nodo A no se encuentre por debajo en la jerarquía y sin embargo la clave de la fila que se intenta dar de alta ya exista, en cuyo caso se debería enviar un error informando de que no se puede usar.

Modificar el estado del servidor

La modificación de valores sucede en dos fases. Primero se validan los valores para ver si son correctos: por ejemplo, si se modifica el grupo de cachés al que pertenece una caché deberá mirarse si ese grupo caché existe. Después se hace la modificación en el sistema.

Las modificaciones no se hacen directamente sobre la MIB; en lugar de eso se modifica el sistema realizando las operaciones necesarias y el mecanismo de eventos visto antes informará al agente del cambio y en ese momento se reflejará dentro del agente.

Para permitir modificaciones hace falta crear 3 nuevas funciones de instrumentación:

- `tabla(is_set_ok, RowIndex, Cols, NombreTabla)`: comprueba que los parámetros están bien, en caso de no estarlo se llama a **undo**, sinó a **set**. Aquí también se pueden reservar recursos para hacer la modificación, ...
- `tabla(set, RowIndex, Cols, NombreTabla)`: realiza la modificación de los parámetros.
- `tabla(undo, RowIndex, Cols, NombreTabla)`: libera recursos solicitados por **is_set_ok** y deshace los cambios que haya realizado.

El problema que nos encontramos es que en cada tabla de la MIB hay que realizar unas validaciones y unas acciones distintas. La solución está basada en el patrón *Estrategia* (*Strategy* [4]). Lo que nos gustaría es que cada vez que se invoque **is_set_ok** o **set** se hicieran las comprobaciones o las modificaciones que corresponden a la tabla indicada por *NombreTabla*. Para esto primero se definen dos nuevas funciones en el módulo `tabla_mib`:

- `modifica_valores/3`: Recibe como parámetros el Oid de la fila, una lista de tuplas {`Indice, Valor`} y una TablaMib. Modifica la fila indicada por el Oid cambiando los valores indicados en la lista de tuplas, en la tabla *MibTable*. Devuelve {`noError,0`} en caso de no haber errores o {`genErr, Indice`} donde `Indice` indica la columna que provocó el fallo.
- `valida_valores/3`: recibe los mismos parámetros que la anterior pero esta vez valida las columnas. Devuelve {`noError,0`} o {`wrongValue, Indice`} donde `Indice` indica la primera columna que no pasó la validación.

El mecanismo que permite que cada tabla valide las columnas que quiera y de la forma que quiera es el siguiente: Se definen unas funciones de validación que deben cumplir los siguientes requisitos:

- Tener al menos un parámetro que será el Oid de la fila. Este parámetro será siempre el primero.
- Devolver `noError` en caso de que sea correcto y {`wrongValue, NumParam`} en caso de que un parámetro no pase la validación siendo el primer parámetro (el Oid) el 0.

La función `valida_valores/3` llamará a estas funciones para validar las columnas en cada validación de los valores de una fila. En caso de que una sola validación dé error devolverá {`wrongValue, ColumnaFallo`}.

Para modificar los parámetros se definen funciones de modificación que deben cumplir:

- Tener al menos un parámetro que será el Oid de la fila. Este parámetro será siempre el primero.
- Devolver `noError` en caso de que sea correcto y `genErr` en caso de haber un error cualquiera.

Las funciones de modificación serán llamadas para modificar las columnas de una fila por la función `modifica_valores/3`. En caso de que una sola modificación dé error devolverá {`genErr, ColumnaFallo`}.

Sólo queda asignar estas funciones a cada tabla y dentro de cada tabla saber para qué columnas se aplican. La forma de hacerlo es indicar en la creación de la tabla las dos listas de funciones indicando antes de cada función los índices de las columnas que se pasarán como parámetros. Para eso se añaden dos parámetros a la función `create/5` dando lugar a la función `create/7`, estos parámetros son: una lista de funciones de modificación formada por tuplas con la forma {`Indices, Modulo, Funcion`} donde se indican los índices de las columnas sobre las que se aplica la función, el módulo en el que se encuentra la función y el nombre de la función; donde las funciones deben devolver `noError` en caso de no haber error o `genErr` en caso contrario, y otra lista de funciones de validación, igual que la anterior salvo porque las funciones deben devolver `noError` en caso de no haber error o `genErr` en caso contrario.
Estas nuevas funciones se introdujeron en un módulo llamado `estrategias` quedando los módulos como se ilustra en la figura 9.

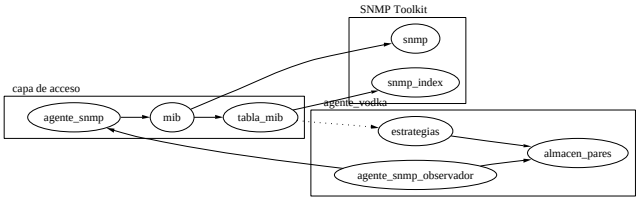

Figura 9: Módulos del agente

Vamos a ver un ejemplo para mostrar el funcionamiento de este mecanismo (figura 10). El cliente quiere

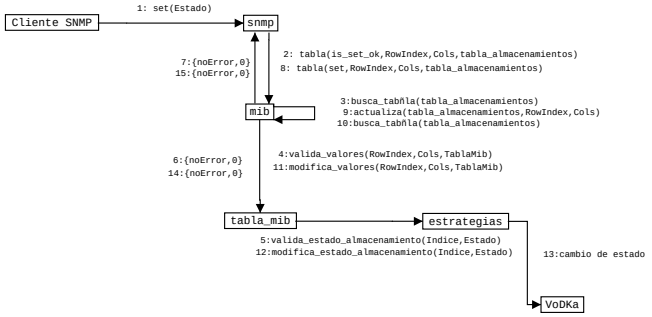

Figura 10: Modificación de un valor

modificar el estado de un almacenamiento e invoca una operación `set` desde el cliente SNMP sobre la tabla de almacenamientos indicando el Oid de la fila. El agente SNMP genérico llama a nuestra función `is_set_ok`, en la MIB se busca la tabla y se llama a `valida_valores/3` que invoca la operación de validación asignada a esa columna, que en este caso es `valida_estado_almacenamiento/2`. El resultado es positivo y se envía un mensaje

con {noError,0} a la MIB que a su vez se lo envía al agente SNMP genérico. Al no haber error se invoca a nuestra operación set que provoca una llamada a modifica_valores/3. modifica_estado_almacenamiento/2 es la función que modifica el estado, esta función provoca una llamada al sistema VoDKA para alterar el valor. Si la llamada es correcta se informa y finalmente el agente genérico muestra al cliente los valores insertados. No se modifica realmente el contenido de la MIB, lo que se hace es provocar un cambio en el sistema.

Alta de nuevas instancias

Finalmente queremos poder dar de alta nuevas instancias en la MIB. Se sabe que se da de alta una nueva instancia porque la columna con el *RowStatus* tiene el valor active (6). La solución es análoga a la anterior solo que en este caso se define una única función de inserción y se introduce la posibilidad de indicar los valores por defecto que se usan en caso de no dar valores para algunas columnas obligadas.

Las funciones de inserción deben tener al menos un parámetro que será el Oid de la fila. Este parámetro será siempre el primero. Y devolver noError en caso de que sea correcto y genErr en caso de haber un error cualquiera. Estas funciones serán llamadas para insertar las columnas de una fila por la función inserta_valores/3. En caso de que la inserción dé error devolverá {genErr, 0}, si no hay error devuelve {noError, ValoresNoInsertados} donde ValoresNoInsertados son los elementos de Cols que no fueron insertados, en este caso la MIB tiene la responsabilidad de llamar a {modifica_valores/3} con esta nueva lista para completar la operación.

Se añaden dos parámetros a la función create/7 dando lugar a la función create/9, estos parámetros son: una lista de funciones de inserción con la misma forma que las otras listas de funciones, si se hace una operación **set** con más columnas que las necesarias para la inserción (i.e. al dar de alta un *trader* no se puede especificar el siguiente trader) después de la inserción se invocan las operaciónes de modificación sobre los parámetros restantes; las funciones deben devolver noError en caso de no haber error o genErr en caso contrario, y una lista de valores por defecto formada por pares {Indice,Valor} que indican el valor por defecto en una inserción para la columna con índice Indice en caso de que no se dé ningún valor para ella.
Veremos qué sucede al dar de alta una caché desde un cliente SNMP (figura 11).

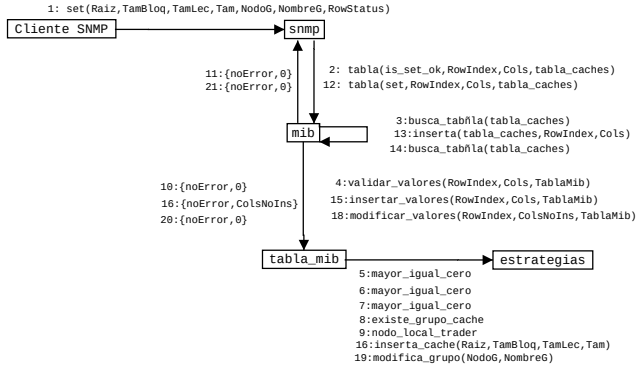

Figura 11: Nueva instancia en la MIB

El cliente quiere dar de alta una caché y especifica la raíz de la caché, el tamaño de bloque, el tamaño de lectura, el tamaño de la caché y el grupo al que pertenece. Como ya vimos, este tipo de operaciones se realiza en dos fases: una de validación de parámetros y otra en la que se hace la modificación o inserción en caso de que la validación sea correcta. El módulo snmp inicia la primera fase invocando la operación tabla(is_set_ok, RowIndex, Cols, tabla_caches) sobre la MIB. La MIB busca la tabla y le pide que valide los parámetros que envió el usuario. La tabla valida los parámetros uno a uno con las funciones de validación, en este caso se comprueba que los tamaños sean mayores que cero, que exista el grupo caché y que se trate de dar de alta en el nodo en el que se encuentra el agente. La operación es exitosa y se informa de ello a la MIB que avisa al agente. Entonces se inicia la segunda fase invocando la operación tabla(set, RowIndex, Cols, tabla_caches). Al estar presente la columna con el *RowStatus* en la operacion **set** y tener el valor active sabe que es una operación de inserción, busca la tabla y le pide que inserte la fila con la clave especificada por el usuario. La tabla invoca la operación inserta_cache que no usa todos los parámetros porque al dar de alta una caché no se puede especificar el grupo al que pertenece, devuelve a la MIB la lista de las columnas que no pudo insertar y ésta

invoca la operación de modificación sobre estas columnas. Finalmente la tabla modifica el grupo caché e informa de que todo fué bien. El agente informa al cliente de las columnas insertadas.

6. Conclusiones

En este trabajo se ha propuesto un diseño para monitorizar y gestionar los cambios del servidor de vídeo con una tecnología basada en la gestión de redes. El uso de un protocolo estándar como SNMP permite centrarse sólo en la adquisición de datos (y generación en caso de datos estadísticos) y en la implementación de las acciones posibles que se permiten realizar, dejando a las aplicaciones de gestión el resto.

El sistema desarrollado es lo suficientemente versátil como para permitir el control del servidor de forma remota y si se utilizan las posiblidades de seguridad de SNMPv3 se puede llegar a realizar una administración completa y segura del servidor usando SNMP.

El uso de patrones de diseño y de ERLANG/OTP para la implementación proporciona una opción robusta para conseguir un sistema distribuido y escalable, ejecutable sobre un *cluster* de ordenadores. Además el uso de este lenguaje permite dotar, de forma sencilla, de tolerancia a fallos a la aplicación desarrollada. El uso de las herramientas del lenguaje reducen de forma drástica los tiempos de construcción de un agente SNMP.

En la actualidad se plantea el uso de este sistema para gestionar los cambios en la arquitectura del servidor en base a resultados previstos por modelos de desempeño que se integren en el proceso de toma de decisiones.

Referencias

[1] M. Barreiro, V. Gulías, J. Sánchez, and J. Jorge, "The tertiary level in a functional cluster-based hierarchical vod system.," in *European Computer Aided Sustems Theory EUROCAST'01*, ISBN 84-699-3971-8, (Las Palmas de Gran Canaria), pp. 174–176, February 2001.

[2] J. Sánchez, V. Gulías, A. Valderruten, and J. Mosquera, "State of the art and design of vod systems," in *International Conference on Information Systems Analisis*, SCI'00-ISAS'00.ISBN 980-07-6694-4, (Orlando, USA), pp. 174–176, July 2000.

[3] J. L. Armstrong, M. C. Williams, C. Wikström, and S. R. Virding, *Concurrent Programming in Erlang*. Prentice Hall, 2nd edition ed., 1996.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*. Addison Wesley, 1995.

[5] A. Valderruten, V. Gulías, J. Sánchez, and J. M. J.L. Freire, "Implementación de un modelo de monitorización para un servidor de vídeo bajo demanda en erlang," in *CLEI*, (Mérida, Venezuela), 2001.

[6] W. Stallings, *SNMP, SNMPv2, SNMPv3 and RMON 1 and RMON 2 Third Edition*. Addison-Wesley, December 1998.

[7] K. McCloghrie and M. Rose, "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II; RFC-1213," *Internet Request for Comments*, no. 1213, 1991.

[8] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP); RFC-1157," *Internet Request for Comments*, May 1990.

[9] J. Case, K. McCloghrie, M. Rose, and W. S., "Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2); RFC-1148," *Internet Request for Comments*, April 1993.

[10] U. Blumenthal and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3); RFC-2274," *Internet Request for Comments*, January 1998.

[11] B. Wijnen, R. Presuhn, and K. McCloghrie, "View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP); RFC-2275," *Internet Request for Comments*, January 1998.

[12] "Erlang OTP Documentation/Simple Network Managent Protocol (SNMP)." http://www.erlang.org, 2002.

[13] M. T. Rose and K. McCloghrie, "RFC 1155: Structure and identification of management information for TCP/IP-based internets," May 1990.

[14] P. Y. Yemini, "A Critical Survey of Network Management Protocol Standars," in *Telecomunications Network Management into the 21st Century* (I. Press, ed.), ch. 2, 1994.