

## Arquitectura y seguridad

En el desarrollo del SIGOB nos hemos enfrentado a diversos problemas que nos han llevado a investigar y desarrollar nuestras propias tecnologías. En este documento presentamos cada uno de los desarrollos que hemos llevado a cabo según las áreas de cada problema.

### Integración

#### de las aplicaciones

Luego de la migración del SIGOB a la plataforma Win32 y de su puesta en operación en múltiples países, surgió la necesidad de que los módulos que componen el sistema pudieran interactuar de forma integrada entre sí.

#### ■ Descripción del Problema

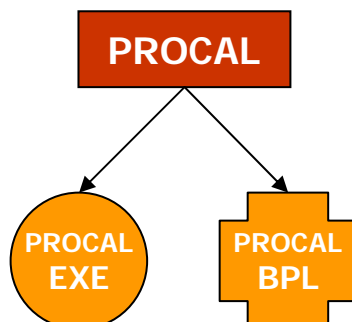
La integración debía ser tal que fuese posible, por ejemplo, consultar y/o modificar desde la Agenda Personal de un funcionario los detalles de una Meta de un Programa Calendario de acuerdo a la relación entre una de sus tareas programadas, sin necesidad de salir de la Agenda para ingresar al instrumento de Programas Calendarios.

Este requerimiento hacía necesaria alguna forma de poder comunicar los procesos ejecutables de cada módulo y poder compartir estructuras de información entre ellos. Todo ello nos llevó a investigar los mecanismos disponibles en la plataforma Win32 para comunicación entre procesos, y desarrollar un mecanismo propio de integración de nuestras aplicaciones.

#### ■ Diseño de la Solución

Nuestro primer paso fue **dividir nuestras aplicaciones** en pares de módulos ejecutables (EXE) y librerías de enlace dinámico de un formato particular usado para empaquetar clases de objetos (BPL), de forma tal que la funcionalidad de cada módulo que se necesite en todo el sistema fue dispuesta en la BPL, y la funcionalidad específica del módulo fue compilada en el EXE.

La Figura 1 muestra como ejemplo la división del módulo de Programas Calendarios (PROCAL) en estas dos partes.



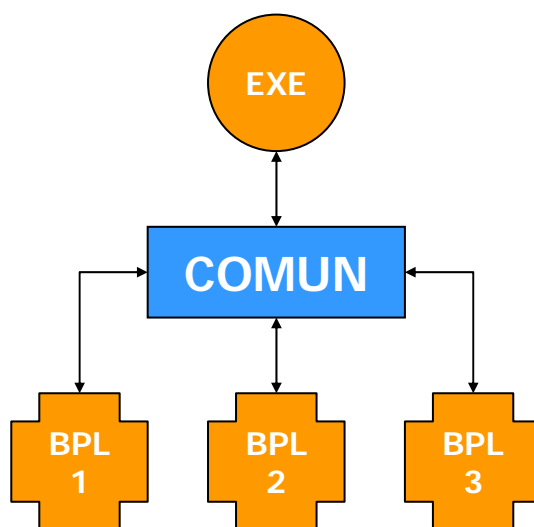
**Figura 1:**  
División del módulo de Programas  
Calendarios en Ejecutables y Librerías.

Al modularizar las aplicaciones que componen el SIGOB, decidimos usar la tecnología de Paquetes (BPL) que Delphi ofrece para compartimentar las aplicaciones y su acceso a los componentes usados en común por ellas. Ello porque el modelo de ejecutable stand-alone que Delphi ofrece resulta inconveniente en términos de recursos cuando se trata de aplicaciones que comparten componentes de software, ya que los mismos se compilan de forma completa en cada aplicación por separado, duplicando varias veces el mismo código compilado. En cambio los paquetes Delphi nos permiten distribuir los componentes compartidos en librerías BPL especiales, reduciendo el tamaño de los archivos ejecutables y mejorando el rendimiento de las aplicaciones al cargar sólo una copia del software compartido en memoria.

Nuestro siguiente problema fue que para **lograr una comunicación entre los procesos** era necesario compartir los códigos de entrada de cada librería en todos los módulos. Es decir, si la Agenda Personal necesitaba comunicarse con Programas Calendarios y con Centro de Gestión, entonces había que incorporar el código de entrada a las librerías de ambos módulos en la Agenda; de la misma forma si Programas Calendarios requiere comunicarse con Centro de Gestión, hay que insertar el código de entrada de las librerías en el software de Programas Calendarios. Todo esto nos introducía nuevos problemas, ya que si una de las librerías cambiaba entonces había que replicar esos cambios en cada módulo que hiciera uso de ella.

Al ser SIGOB un sistema complejo y de muchos módulos, necesitábamos un nivel de abstracción superior que nos evitara el posible caos de mantener los cambios en los instrumentos. Fue así como diseñamos un **módulo coordinador** de la comunicación entre los procesos, de forma tal que todo requerimiento de procesos y sus respuestas fueran manejados exclusivamente por dicho módulo.

La figura 2 muestra el esquema de funcionamiento del módulo coordinador COMUN.



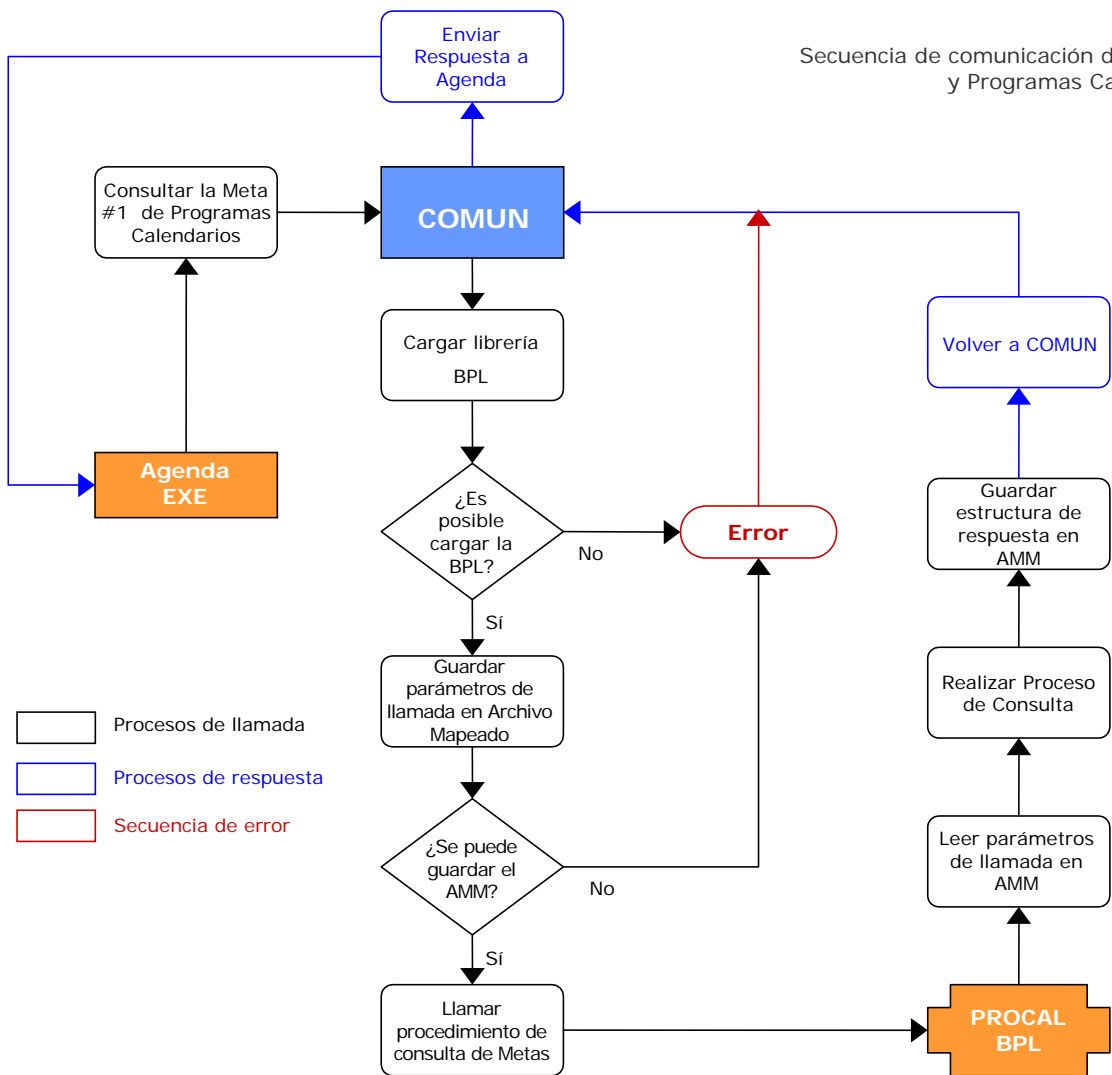
**Figura 2:**  
Esquema de funcionamiento del coordinador COMUN

Para realizar la comunicación de todos los procesos, creamos dos estructuras de información globales: Una para los requerimientos de proceso (Tllamada) y otra para la respuesta (TRespuesta) a esos requerimientos. Implementamos también un espacio de memoria común, a través de un Archivo de Memoria Mapeado, para poder compartir datos entre las aplicaciones.

Como muestra la figura 2, el módulo COMUN coordina toda la comunicación entre los diferentes módulos del sistema. Para lograr el nivel superior de abstracción que buscábamos, centralizamos todo el código de comunicación en una función común que es conocida por todos los módulos. Dicha función realiza todo el proceso enviando los requerimientos de los módulos clientes a los módulos servidores, retornando la respuesta de los módulos servidores a los módulos clientes, y manejando todas las condiciones de error (como por ejemplo, la ausencia de uno de los módulos servidores). Una ventaja inmediatamente notable es que ninguno de los módulos conoce los detalles de implementación de los demás (como nombre de Archivo, ubicación y formato de llamada). El único módulo que todas las aplicaciones conocen es el coordinador COMUN.

Por otra parte, la comunicación entre procesos no está limitada a los pares EXE-BPL: a través del coordinador es posible comunicar módulos BPL entre sí, permitiendo por ejemplo que el módulo BPL de Programas Calendarios pida algo al módulo BPL de Centro de Gestión. Ello gracias a la estructura de nuestro espacio de memoria compartido que es gestionado por el coordinador. Nuevamente es notable cómo las BPL no saben de la existencia de las demás.

A modo de ejemplo, en la figura 3 se puede apreciar la secuencia del proceso de comunicación entre el módulo Agenda Personal y el módulo Programas Calendarios.



## Seguridad y Conexión

### a Bases de Datos

Dado que SIGOB es un sistema que maneja información crítica para las instituciones en las que opera, requiere de un mecanismo de seguridad que garantice que esa información esté disponible sólo para los usuarios registrados en el sistema.

En su diseño SIGOB cuenta con esquemas de seguridad de sistema, que incluyen el mecanismo usual de credenciales de usuario (nombre de usuario y contraseña) para el acceso, y un extenso conjunto de roles y atributos sobre la información para cada usuario en cada módulo. Sin embargo, existe un detalle en el funcionamiento del motor de bases de datos que nos decidió a desarrollar un mecanismo de seguridad adicional.

#### ■ Descripción del Problema

Usamos el esquema nativo de seguridad de SQL Server para comunicar nuestras aplicaciones con la base de datos. El mismo se basa en un par de credenciales de usuario que se transmiten al momento de realizar una conexión a la base, donde el motor realiza el proceso de autenticación correspondiente. El problema de este esquema es que las credenciales son transmitidas como texto plano por la red, donde existe el riesgo de ser interceptadas por terceros.

Por otra parte, las credenciales para la conexión a la base de datos no pueden ser las mismas que las usadas por los usuarios para autenticarse en el SIGOB, ya que ello abre la posibilidad de que usuarios con avanzados conocimientos informáticos puedan usar sus credenciales de acceso para intervenir directamente en la base de datos sin usar el sistema, con el enorme riesgo que eso conlleva.

Adicionalmente existía el requerimiento de poder conectarse desde una estación cliente a más de una base de datos con el sistema instalado. De esa forma sería posible compartimentar la información de diferentes áreas en varias bases de datos con el sistema instalado en ellas, y dar acceso a los usuarios a la información distribuida sin necesidad de cambiar de estación de trabajo.

#### ■ Diseño de la Solución

Diseñamos un esquema de doble autenticación que nos permite mantener un estricto control sobre los accesos al sistema. Al igual que la comunicación entre los módulos, decidimos centralizar el mecanismo de seguridad en el módulo coordinador COMUN.

El esquema de doble autenticación consiste en dos pares de credenciales de usuario:

1. Credenciales de acceso al sistema: El nombre de usuario y contraseña que cada usuario utiliza para autenticarse en el sistema. Las mismas son conocidas por cada usuario y por el administrador.
2. Credenciales de conexión a la base de datos: El nombre de usuario y la contraseña del Login correspondiente de la base de datos. Estas credenciales deben ser conocidas sólo por el administrador de la base de datos.

Si bien los usuarios conocen sus credenciales de acceso al sistema, las mismas sólo son útiles dentro de él. Es decir, un usuario avanzado no podría utilizar esas credenciales para acceder a la base de datos fuera del contexto de ejecución del sistema, ya que SIGOB utiliza las credenciales privadas para conectarse a la base de datos.

Todas las credenciales son almacenadas de forma encriptada en la base de datos, usando el algoritmo Rijndael para el cifrado de la información. La información de las credenciales de usuario siempre se maneja cifrada, nunca se manipula la misma de forma textual (sin encriptar), por lo cual la única forma de acceder a las credenciales de un usuario es poseer el algoritmo y las llaves privadas de cifrado que se encuentran seguras, almacenadas de forma compleja en el código compilado de los módulos del sistema.

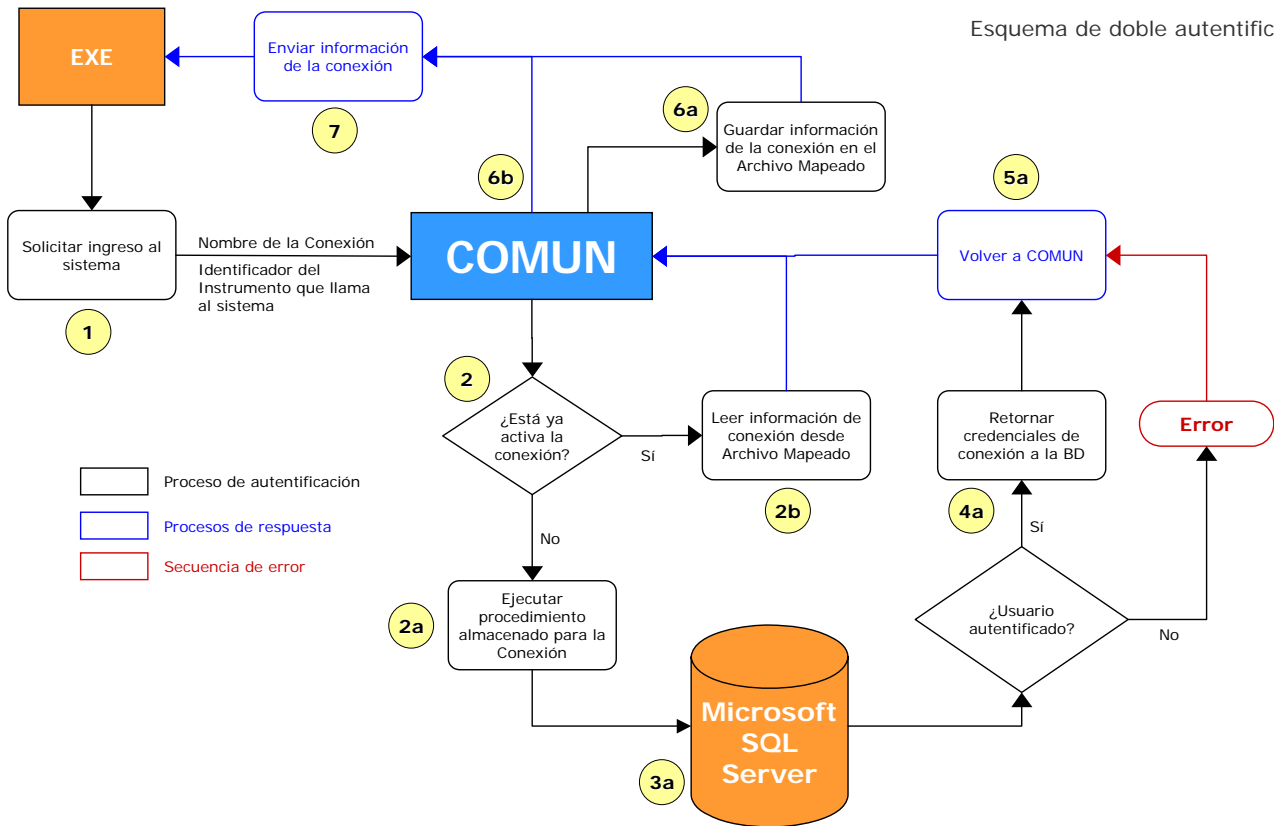
El uso de credenciales de conexión a la base de datos para cada usuario por separado nos ha permitido además definir un mecanismo de roles y permisos de acceso a los objetos de la base de datos. Es así como los usuarios del sistema sólo tienen atributos limitados de lectura y modificación de determinadas tablas en la base de datos, y tienen prohibido acceder a los objetos de administración de la base.

Para soportar la conexión a múltiples bases de datos incorporamos en este esquema un modelo de conexiones por nombre. El mismo consiste en asignar los parámetros de conexión públicos de cada base de datos (dirección del servidor en la red, nombre de la base de datos) a una estructura de información almacenada en el Registro del perfil de cada usuario en Windows, la cual es identificada por un nombre único en la estación cliente.

El esquema de doble autenticación funciona de la siguiente manera:

Figura 3:

Esquema de doble autenticación



1. Al inicio de la ejecución de uno de los módulos clientes, se invoca al módulo coordinador y se le solicita ingreso al sistema. Como parte de la solicitud, se le entrega como información el nombre de la conexión a usar y un identificador del instrumento que llama.
2. El módulo COMUN utiliza la información entregada por el módulo cliente para averiguar si ya hay una conexión activa con ese nombre, lo que significaría que otro módulo cliente ya se autenticó en la misma conexión previamente. Dos posibilidades se abren aquí:
  - a) No existe información en memoria de la conexión con el nombre dado: Iniciamos el proceso de autenticación remoto. Establecemos una conexión temporal a la base de datos usando credenciales de un usuario predefinido, que sólo tiene permiso de ejecutar el procedimiento almacenado de autenticación. Seguimos en el paso 3.

- b) Hay información en memoria de la conexión con el nombre dado: Ya hay una conexión establecida bajo ese nombre en memoria. Esto significa que previamente otro módulo cliente realizó el proceso de autenticación remoto para esa conexión, por lo cual el módulo coordinador dispone de la información de la conexión (credenciales privadas de acceso y variables de entorno). A través de esta verificación evitamos repetir el proceso completo de autenticación remoto, manteniendo un caché local en cada estación con los datos de las sesiones abiertas actualmente. Seguimos en el paso 6b.
3. En el servidor de bases de datos se ejecuta el procedimiento almacenado encargado de la autenticación de las credenciales dadas. Cabe destacar nuevamente que las credenciales de acceso al sistema son encriptadas usando un poderoso algoritmo antes de ser transmitidas por la red, para asegurarnos de que no sean útiles en caso de ser interceptadas por terceros.
4. Si el usuario es autenticado en el procedimiento remoto, el servidor retorna las credenciales privadas de conexión para el usuario autenticado (también fuertemente encriptadas), y un conjunto de variables de entorno para la sesión a ser iniciada. De lo contrario, retorna los códigos de error correspondientes.
5. El módulo coordinador retoma el control luego de la ejecución del proceso remoto de autenticación en la base de datos.
6. Con la información del proceso de autenticación, el módulo coordinador determina qué pasos seguir de acuerdo a los resultados del proceso. Según el camino tomado en el paso 2, se abren acá dos posibilidades:
  - a. Si se realizó el proceso de autenticación remoto, el módulo coordinador guarda la información de conexión recibida en el archivo mapeado de memoria. De esta manera configuramos un caché local con la información de esa conexión, en caso de que algún otro módulo necesite acceder a ella.
  - b. Se lee la información de la conexión existente desde el archivo mapeado de memoria.
7. Finalmente se envía la información de la conexión al módulo cliente y se le devuelve el control de la ejecución.

## Instalación del

### Modelo de Datos

El modelo E/R sobre el cual se sustenta el SIGOB es muy complejo. Con más de 120 tablas diferentes y un número mayor aún de relaciones entre las mismas, resulta un desafío mantener la implementación del modelo en las bases de datos ante los cambios que se realizan entre cada versión.

#### ■ Descripción del Problema

La implementación del modelo físico de datos del SIGOB en una nueva instalación es un proceso bastante sencillo. A pesar de la complejidad del modelo, sólo se construyen las tablas y relaciones del mismo. Sin embargo, la situación es muy diferente cuando se trata de actualizar a una nueva versión un modelo ya instalado y poblado de información.

Es necesario que el proceso de actualización de una base de datos existente produzca el menor impacto posible en la información existente, de forma tal que luego de correr el proceso se pueda operar el sistema sin mayores pasos adicionales.

#### ■ Diseño de la Solución

Decidimos desarrollar un software para la instalación del modelo de datos que soportara los requerimientos que el problema planteaba. Para ello segmentamos el proceso de instalación en 5 etapas:

1. **Desactivación:** Se detecta la existencia del modelo instalado, y se procede a desactivarlo a través de la remoción de sus mecanismos de integridad referencial, índices y llaves primarias.
2. **Respaldo:** Se renombra cada tabla a ser actualizada, de forma tal de mantener un respaldo de su información.
3. **Creación:** Se crean las tablas con la nueva estructura del modelo de datos.

4. **Recuperación:** Se compara la estructura de cada tabla actualizada con la nueva versión, para determinar las diferencias entre ellas. Si no hay diferencias, la nueva versión de la tabla se borra y se repone la versión respaldada de la misma. En caso de haber diferencias, el programa intenta poblar la nueva versión de la tabla con la información de la tabla de respaldo. Si no es posible poblar la nueva tabla (porque las estructuras difieren mucho, por ejemplo), el software avisa al administrador que ha habido problemas con esa tabla, y que su información se encuentra en la tabla respaldada. De esa forma el administrador puede recuperar la información manualmente después de la instalación.
  
5. **Consolidación:** Se crean los índices y llaves primarias de la nueva versión del modelo, se instala el esquema de seguridad primario (roles y atributos sobre los objetos de la base de datos), y finalmente se activa la base de datos para su puesta en operación.

Este diseño ha funcionado exitosamente, y ha sido clave para mantener al día las instalaciones existentes en las instituciones clientes, sin requerir asistencia técnica de ninguna especie.