UNIVERSIDAD CATOLICA ANDRES BELLO
DIRECCIÓN DE POSTGRADO
GERENCIA DE SISTEMAS DE INFORMACIÓN

DISEÑO Y CONSTRUCION DE UNA BASE DE DATOS DE RELACIONES ENTRE COMPONENTES DE SOFTWARE Y APLICATIVOS DESARROLLADOS POR LA GERENCIA DE BANCA VIRTUAL BANESCO

Proyecto de Investigación presentado por:
José Enrique Curiel Alvis

Profesor Guía:
Licenciado Jhonnatan A. Mazzaro G.

Caracas, Octubre 2007

RESUMEN.

El trabajo especial de grado de especialista que se presenta a continuación muestra los aspectos más relevantes envueltos en el diseño, desarrollo e implantación del repositorio de relaciones de los diferentes elementos que son desarrollados o bien son responsabilidad de la Gerencia de Banca Virtual de Banesco.

Principalmente el objetivo que se persigue al implantar un repositorio de relaciones o de configuración, es contar con una fuente de información que muestre la manera como los distintos componentes de software, servicios, archivos, equipos manipulados por personal de la Gerencia de Banca Virtual de Banesco se relacionan unos con otros, así como la naturaleza y características de estas relaciones. Como se explica más adelante, el repositorio también es conocido como base de datos de configuración (CMDB por sus siglas en inglés).

La metodología empleada para contar con la CMDB se resume en la identificación de los diferentes tipos de elementos con los cuales trabaja la Gerencia de Banca Virtual de Banesco, para luego definir las características de cada uno y los distintos tipos de relaciones que pueden existir entre ellos. Posteriormente, se diseña el modelo de información representativo de todo lo anteriormente expuesto y se completa el proceso con su desarrollo e implantación del repositorio.

Con todo esto, la Gerencia de Banca Virtual de Banesco contará con información que le permite a su personal evaluar con exactitud el impacto que puede tener la modificación de un elemento en el resto, incluyendo el conocer como esto puede afectar el desempeño de los diferentes servicios bajo su tutela. Esto cobra mayor importancia al considerar que como parte de las labores diarias de la gerencia, se ejecutan constantemente operaciones de mantenimiento correctivo y perfectivo sobre la totalidad de los componentes de software, servicios, archivos, equipos bajo su responsabilidad. Así mismo se llevan a cabo nuevos desarrollos que conllevan un aumento en la cantidad de elementos que deben ser mantenidos.

INDICE DE CONTENIDO.

INDICE DE FIGURAS.

INTRODUCCIÓN.

Tal como todas las actividades del quehacer humano, el desarrollo de aplicaciones de software ha evolucionado desde sus inicios, permitiendo mejorar los tiempos de culminación de los proyectos, alcanzar un uso más eficiente de los recursos de hardware donde residen los aplicativos a la par con la aparición de nuevos paradigmas de programación.

Precisamente, con la aparición de nuevas formas de desarrollar software, surgen nuevos retos que superar. Un ejemplo es la concepción de la programación orientada a objetos, y como cambia la forma en que tradicionalmente se realizan actualizaciones o modificaciones en los servicios que son desarrollados bajo este esquema.

Con la programación orientada a objetos, cobra mayor importancia la tarea de identificar las dependencias que guardan aplicativos, que en teoría, no tienen relación alguna. La razón estriba en que la funcionalidad propia de algún servicio o elemento de software puede ser empleada por dos o más servicios diversos. Con esto, al requerir llevar a cabo una modificación en el elemento de software, se corre el riesgo de desconocer el impacto que pueda tener la actualización, si no se tiene la certeza de cual o cuáles aplicativos hacen uso del elemento cambiado.

El explicar los pasos que se siguieron para el diseño, desarrollo e implantación de una herramienta (llamado repositorio de configuración o base de datos de configuración) que permite minimizar el riesgo mencionado en el párrafo anterior constituye el objetivo del trabajo que se presenta a continuación, que consta de cinco apartados:

En el primero se presenta un esbozo del problema al cual se busca dar solución, incluyendo los antecedentes o aspectos que dieron origen a la problemática, la justificación del esfuerzo realizado en el desarrollo de la herramienta y el alcance y limitaciones del proyecto.

El segundo apartado se orienta a exponer las bases teóricas necesarias para comprender la terminología empleada a lo largo del desarrollo. Se tocan aspectos como mejores prácticas sugeridas en el campo de la tecnología, beneficios obtenidos con la uso del repositorio de configuración, concepto de modelo entre otros puntos de interés.

En el tercero se expone el desarrollo propiamente dicho de base de datos de configuración. En este apartado se presenta el objetivo principal que se quiere alcanzar,

el método empleada para su logro, criterios que se siguieron para el diseño del repositorio así como su representación gráfica y textual.

El cuarto apartado presenta la evaluación del proyecto, donde se exponen elementos que permiten calificar el trabajo como exitoso, orientados a demostrar el logro del objetivo propuesto.

Finalmente en el quinto se presenta las conclusiones a las cuales se llego con el desarrollo del proyecto así como las recomendaciones que buscan asegurar el correcto uso de la herramienta desarrollada.

EL PROBLEMA.

Antecedentes.

La Gerencia de Banca Virtual de Banesco es la unidad encargada del desarrollo, implantación y mantenimiento de aplicativos cliente/servidor y en entorno internet que dan soporte a los diferentes productos y servicios ofrecidos por Banesco. Ejemplo de tales aplicaciones son:

- Sitio Institucional: Constituye la representación de la institución en la red. En él se muestra información de los diferentes productos y servicios ofrecidos por la institución tanto a empresas como personas, promociones, indicadores financieros, novedades, ubicación de agencias, novedades, etc.
- Servicio Banca Online: Sistema de banca virtual que permite a los usuarios realizar operaciones financieras en la red, con los diferentes productos que tiene con la institución.
- Servicio Fideicomisos Online: Permite la autogestión del fideicomiso tanto desde el rol del patrono como del beneficiario.
- Servicio Pago Electrónico: Sistema en la red mediante el cual se ejecutan de forma automática los pagos de nóminas de empresas, pagos a proveedores y cobros de domiciliaciones de tarjetas de crédito.
- Servicio Intercambio Electrónico de Datos: Permite realizar de forma automática el pago de la nómina de empresas, con tan sólo instalar un software y programar los pagos en el computador.

De esta manera se realizan tareas propias del área, desarrollo de nuevas aplicaciones para dar soporte al área de negocio de la institución (atención de nuevos requerimientos, análisis de impacto, estimación de costos y esfuerzos de desarrollo, construcción y puesta en producción), así como el mantenimiento preventivo, perfectivo y correctivo de los servicios en la red disponibles al público.

Obedeciendo a una reestructuración de la gerencia, Banca Virtual quedo dividida en dos gerencias, a saber:

- Gerencia de Nuevos Proyectos: Encargada de la elaboración de proyectos de gran magnitud destinados a cumplir con los nuevos requerimientos expresados por las distintas áreas de negocio.
- Gerencia de Soporte y Mantenimiento: Encargada de la elaboración de proyectos de mediana y baja magnitud orientados a la incorporación de

nuevas funcionalidades a las aplicaciones existentes, perfeccionamiento/optimización, mantenimiento correctivo. También es responsable de la integridad y continuidad de los servicios ofrecidos por las distintas aplicaciones en producción.

Planteamiento del problema.

La gran cantidad de aplicaciones desarrolladas por la Gerencia de Banca Virtual se caracteriza por el elevado nivel de interdependencia que existe entre ellas, esto es, que una funcionalidad específica ofrecida por alguna de ellas puede ser utilizada por una o varias aplicaciones/servicios diferentes.

La explicación se encuentra en que la totalidad de las aplicaciones desarrolladas por la Gerencia de Banca Virtual se basa en la programación orientada a objetos. Esto es, diferentes componentes de software que realizan diferentes actividades son elaborados con la intención que puedan ser utilizados por distintas aplicaciones o servicios. Así se logra (a) reusabilidad de código, (b) desarrollo de aplicaciones altamente modificables y extensibles a partir de componentes reutilizables y, (c) disminución del tiempo de desarrollo de aplicaciones.

Debido a la dinámica del negocio, el personal perteneciente a la gerencia de soporte y mantenimiento requiere realizar continuas modificaciones en los componentes, bien sea para introducir mejoras o extensiones de funcionalidad, realizar actualizaciones en los servicios o bien corregir defectos. Esto lleva a situaciones en las que (a) un cambio que se realiza en un componente para un servicio en particular, impacta negativamente la continuidad de otros; (b) ha sido necesario reversar y retrasar la liberación de nuevos servicios en función a como estos afectan la continuidad de otros.

A pesar que el número de modificaciones, actualizaciones y liberaciones de nuevos servicios no siempre ha afectado la disponibilidad del resto, se puede disminuir aún más el impacto, surgiendo la iniciativa de refinar el marco de procesos y herramientas que apoyan esas actividades.

En base a lo anteriormente descrito, se identifica la necesidad de contar con un repositorio de información donde consultar información pertinente a los componentes de software y servicios (por ejemplo, ubicación física del código fuente, responsable técnico, problemas identificados, versión) y la relación existente entre todos los componentes de software que forman parte de los aplicativos desarrollados. Gracias a

ello (a) se dispone con datos suficiente para vislumbrar como los cambios desarrollados en algunos de los componentes afectan la continuidad del resto de los servicios, (b) se asegura la calidad en el software que se desarrolla para apoyar al negocio, (c) se implantan de manera eficiente los continuos cambios solicitados por los usuarios, (d) se alcanza una mayor adaptabilidad de los servicios/aplicativos al entorno cambiante.

Los usuarios del repositorio constituyen el personal de perteneciente a la Gerencia de Nuevos Proyectos y a la Gerencia de Soporte y Mantenimiento.

Justificación.

El contar con un repositorio de relaciones donde se encuentre información útil de los diferentes aplicativos, servicios y componentes desarrollados por la Gerencia de Banca Virtual, es necesario en virtud que:

- Permite realizar una mejor estimación del esfuerzo requerido al momento de actualizar algún componente, al tener identificado otros componentes o servicios que deben ser adecuados para que el cambio no afecte su desempeño.
- Ofrece la posibilidad de no solo conocer, dado un componente en particular, con cual o cuales otros componentes o servicios se relaciona, si no que rol o papel tienen cada uno de ellos en esa relación.
- Permite identificar los servicios cuyo funcionamiento se debe monitorear una vez se han introducido cambios en algún componente que guarda relación con ellos.
- Disminuye la afectación de los servicios responsabilidad de la Gerencia de Banca Virtual como producto de modificaciones mal planificadas y estimadas.
- Permite conocer la relación que existe entre los servicios desarrollados por la Gerencia de Banca Virtual y los elementos que son responsabilidad de otras áreas de la institución. Con esto se puede evaluar con exactitud como pueden verse afectados los servicios en virtud de cambios o actualizaciones que no son promovidas por la Gerencia de Banca Virtual.

Alcance y limitaciones.

El esfuerzo que se invierte en la implantación de una herramienta de apoyo en las labores de administración de la configuración, que es el objetivo principal del trabajo que se presenta a continuación, abarca el diseño, desarrollo e implementación de una CMDB que sea útil en las labores diarias de la gerencia.

Para alcanzar este objetivo, se definen los tipos de elementos que forman parte del repositorio, las características de cada uno de ellos y la naturaleza de las relaciones que guardan entre si. Seguidamente se realiza el diseño del modelo de datos y se avanza a las etapas de desarrollo e implementación del repositorio.

El presente trabajo no prevé el desarrollo de ningún tipo de interfaz para facilitar el uso de la CMDB, por el contrario, la premisa bajo la cual se trabaja es de concebir el diseño del modelo de datos basado en un estándar que permita una fácil exportación a cualquiera de las herramientas existentes en el mercado, orientadas a presentar a los usuarios una interfaz amigable que facilite su experiencia en el manejo de los elementos presentes en el repositorio.

La limitación que se tiene se relaciona con el punto anterior, esto es, la herramienta seleccionada para el uso del repositorio debe ser compatible con la infraestructura tecnológica presente en el banco, en especial, debe ser capaz de ejecutarse eficazmente en el sistema operativo presente en todos los equipos de computación utilizados por el personal de la gerencia. No debe requerir para su utilización la compra de ningún tipo de licencia por parte de Banesco, esto es, su uso no debe acarrear ningún tipo de problema legal a la institución.

MARCO TEÓRICO.

Biblioteca de Infraestructura de Tecnologías de Información.

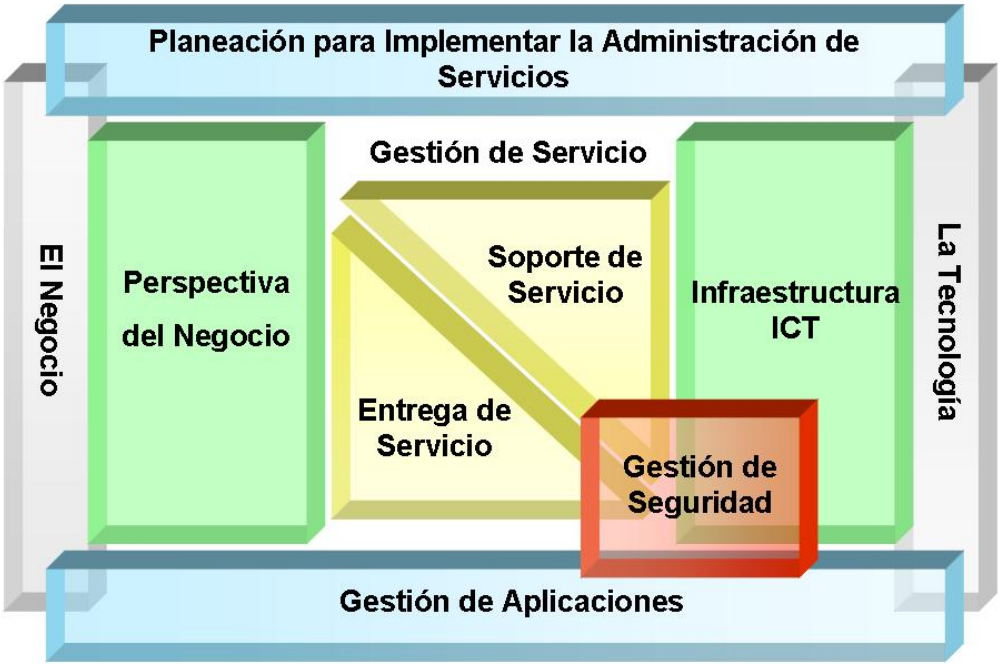
En la actualidad existe un gran número de metodologías de trabajo orientadas a maximizar el empleo de los recursos de tecnología y proporcionar las bases para la prestación de servicios de cara al cliente con un alto nivel de calidad.

De todas ellas sobresale una que se basa en la recopilación de mejores prácticas puestas en ejecución por un grupo importante de organizaciones y cuyo resultado ha sido satisfactoria para ellas, nos referimos a la Biblioteca de Infraestructura de Tecnologías de Información (ITIL según sus siglas en inglés), desarrollado hacia finales de los años 80 por la Agencia Central de Computación y Telecomunicaciones británica (CCTA por sus siglas en inglés) ahora denominada Ministerio de Comercio (OGC por sus siglas en inglés) del Reino Unido. Cossío y Palomino (2005) definen ITIL como "el marco de referencia más aceptado y utilizado en el mundo. Proporciona un conjunto de las mejores prácticas extraídas de organismos del sector público y privado que están a la vanguardia tecnológica a nivel internacional". Según la OGC, ITIL constituye "un conjunto de guías en la administración y provisión de servicio operacionales de tecnologías de información".

La Biblioteca de Infraestructura de Tecnologías de Información ha evolucionado desde su primera concepción, y antes de ser llevado a la práctica por vez primera, su modelo de procesos fue, y sigue siendo, probado y mejorado, garantizando englobar siempre las mejores prácticas para la administración de los servicios de Tecnología de Información (TI).

Estas guías se agrupan en una serie de siete libros donde se describe la manera en que los procesos relacionados con la provisión de servicios pueden ser organizados y coordinados de mejor manera, aparte de contener las mejores prácticas relacionadas con la prestación de servicios de TI. Los libros correspondientes a entrega de servicios y soporte de servicios agrupan los procesos más implantados dentro de la gerencia de servicios de TI. El resto de los libros se enfocan en procesos de gestión de seguridad, gestión de aplicaciones, perspectiva del negocio, planeación para implementar la administración de servicios e infraestructura de Tecnología de Información y Comunicaciones. En la Figura 1 se observa el esquema de ITIL.

Figura 1: Estructura de ITIL (tomada de Information technology infrastructure library, 2005).



- Entrega de Servicios: Constituye la administración de los recursos de TI, incluyendo un conjunto de mejores prácticas orientadas a asegurar que los servicios de TI son provistos tal cual fue acordado entre el cliente y el proveedor. Este proceso se divide en las siguientes disciplinas.
    - Gestión de Nivel de Servicio: Encargado de asegurar que el servicio es prestado en el momento y en la forma en que fue acordado con el cliente. Este proceso también se relaciona con la evaluación del impacto en la calidad del servicio proveniente de un cambio.
    - Gestión de la Capacidad: Asegura que la infraestructura de TI es empleada de la manera más eficiente posible, alineando los recursos de TI con la demanda de servicio.
    - Gestión de Continuidad de Servicios de TI: Encargado de asegurar que los componentes de TI que sustentan servicios críticos puedan ser recuperados dentro de un espacio de tiempo (por lo general

acordado con el cliente), luego de una interrupción grave en su funcionamiento.

- Gestión de la Disponibilidad: Identifica los niveles de disponibilidad de servicios de TI para su uso en las revisiones que, de los niveles de servicios, se realizan con el cliente.
- Gestión Financiera de TI: Asegura que los elementos de tecnología (infraestructura) sean adquiridos a un precio efectivo. Así mismo, calcula el costo asociado a proveer el servicio de TI. Este costo será luego cubierto por el cliente.

- Soporte de Servicios: Encargado de velar por que el usuario tenga acceso a los servicios apropiados que le puedan ayudar a mejorar las funciones de su negocio/organización. Este proceso se divide en las siguientes disciplinas:
  - Gestión de la Configuración: Salischiker (2005) "Proceso para identificar y definir los elementos de configuración en un sistema registrando e informando el estado de ellos, y verificando la entereza y veracidad de los elementos de configuración."
  - Gestión de Incidentes: De acuerdo a lo expuesto en Wikipedia el objetivo de la Gestión de Incidentes es "la de restaurar la operación normal del servicio tan pronto como sea posible y minimizar el impacto sobre las operaciones del negocio, asegurando que los mejores niveles de calidad y disponibilidad posible del servicio son mantenidos".
  - Gestión de Problemas: Wikipedia "El objetivo de la Gestión de Problemas es minimizar el impacto adverso de incidentes y problemas en el negocio, causados por errores dentro de la infraestructura de TI, y prevenir la recurrencia de incidentes relacionados con estos errores".
  - Gestión de Cambios: Directory of ITIL, ITSM & security services & software "Es la práctica de asegurar que todos los cambios en los elementos de configuración [software o hardware perteneciente a la infraestructura de TI] son llevados a cabo de una manera planeada y autorizada". Se debe asegurar que todo cambio obedece a una razón de negocio, identificar los servicios o elementos de configuración afectados con el cambio y contar con un plan de reverso en el caso

que el cambio produzca una ruptura del servicio o bien degenere en un estado no deseado de algún elemento de configuración.

  - Gestión de Versiones: El objetivo de este proceso es según Combalia (2005) “Mantener una visión integral de un cambio sobre un servicio de TI y asegurar que todos los aspectos de Versión [agrupación de cambios en un servicio/componente de TI], tanto técnico como no técnicos son tomados en conjunto”.
  - Servicio de Mesa de Ayuda (Service desk según su denominación en inglés): Su objetivo es según Combalia (2005) “Proveer un único punto de contacto entre los usuarios y la organización de Gestión de TI para aconsejar, guiar y restaurar rápidamente los servicios TI”. también se encarga de mantener informados a los usuarios acerca de los eventos, acciones (por ejemplo de mantenimiento) y cambios en los servicios que podrían afectarlos.

- Gestión de Seguridad: Se encarga de proteger y mantener la integridad, disponibilidad y confiabilidad de los elementos de TI. Supervisa que los aspectos de seguridad de los servicios son provistos en conformidad a los niveles acordados entre cliente y proveedores.
- Gestión de la Infraestructura de Tecnología de Información y Comunicaciones (ICT por sus siglas en inglés): Encargada de los procesos y herramientas necesarias para mantener estable y funcional la infraestructura de comunicaciones.
- Gestión de Aplicaciones: Cubre el ciclo de vida del desarrollo nuevos proyectos de software, poniendo especial énfasis en la recolección y definición de requerimientos que cumplan los objetivos del negocio.
- Perspectiva del Negocio: Cossío y Palomino (2005) establecen que el objetivo de la Perspectiva de Negocio es “proporcionar a la alta dirección el diseño, la arquitectura y los componentes fundamentales para definir la Infraestructura de Tecnologías de Información y Comunicaciones (…) indispensable para impulsar los procesos estratégicos del negocio”.
- Planeación para Implementar la Administración de Servicios: Se enfoca en los elementos clave que deben considerarse en la implantación de ITIL

dentro de una organización, asegurando que los servicios proporcionados se ajusten a los requerimientos del negocio.

Beneficios de ITIL.

Dentro de una organización, son muchas y muy variadas las áreas cuyas labores diarias experimentan mejoras en su eficiencia, gracias al seguimiento de las mejores prácticas recomendadas por ITIL, orientadas a prevenir la ocurrencia de situaciones que afectan tanto la imagen de las empresas de servicios de cara a sus clientes como el normal desenvolvimiento de las actividades de producción. Algunas de estas situaciones no deseadas son por ejemplo, (a) interrupciones del servicio ocasionadas por modificaciones no certificadas en componentes de la infraestructura, (b) vencimiento de licencias de software, (c) lentitud en la recuperación de servicios luego de eventos inesperados, (d) índices de disponibilidad de servicio bajos.

De acuerdo con Combalia (2005) y Cossío y Palomino (2005) los beneficios obtenidos con el seguimiento de las recomendaciones de ITIL son:

1. Soporte para los procesos de negocios y las tareas de toma de decisiones en TI.
2. Definición de funciones, roles y responsabilidades en el sector de los servicios.
3. Reducción de gastos en procesos de desarrollo, procedimientos e instrucciones de trabajo.
4. Los servicios de TI cumplen los requerimientos del negocio en particular.
5. Mejora en la satisfacción de los empleados y reducción de fluctuaciones de niveles de personal.
6. Mejor comunicación e información entre personal TI y sus clientes.
7. Obtener información acerca de los cambios que proporcionarán un mayor beneficio para la organización.
8. Obtener una visión clara de la capacidad de las TI y sus ventajas para la organización.
9. Aumentar la satisfacción de los clientes.

Gestión de la Configuración.

Aparte de identificar y definir los elementos que forman parte de la infraestructura de TI de una organización, la Gestión de la Configuración resulta útil en la construcción de un modelo lógico donde sea posible entender como se relacionan

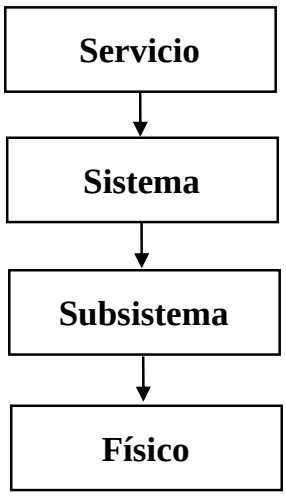
entre si estos elementos, y como estos se relacionan con los servicios ofrecidos por una organización. De acuerdo a Foro HelpDesk (2006) "la aplicación práctica de esto es la creación de los registros lógicos de los EC [elemento de configuración] que describen los servicios de TI o una descomposición de ellos en varios sistemas y subsistemas que alimentan hasta el servicio de más alto nivel".

Algunos de los beneficios de este modelo de acuerdo con Foro HelpDesk (2006) son:

- Se facilita la comprensión de cómo todos los EC dentro del alcance del proceso se relacionan con algún servicio al negocio.
- Se identifica cuales figuras de la disponibilidad se relacionan con EC individuales, con grupos de EC y con objetivos generales de la Disponibilidad del Servicio.
- Se conocen cuáles EC apoyan a múltiples Servicios de TI.
- Se establece la prioridad de los EC en relación a la criticidad del negocio.

Tal como se muestra en la figura 2, la descomposición del modelo divide los EC en cuatro niveles, que describen los agrupamientos de estos elementos que facilitan un servicio general del negocio. Los primeros tres niveles son lógicos, mientras el último representan los elementos físicos que existen en la infraestructura. Para cada uno de los sistemas y servicios del negocio existe una replica de la estructura lógica, siendo esta de naturaleza estática a menos que un nuevo servicio sea introducido al entorno.

Figura 2: Modelo lógico de gestión de configuración (tomado de Foro HelpDesk, 2006).



Esta estructura de servicios permite a los diferentes coordinadores técnicos relacionar los diferentes EC, cambiar las relaciones entre ellos o eliminarlas cuando estos EC no formen parte de la infraestructura.

Para manejar los EC, la Gestión de la Configuración debe ser soportada por un repositorio o base de datos de la gestión de configuración capaz de tener toda la información de los EC, incluyendo atributos específicos y relaciones entre ellos.

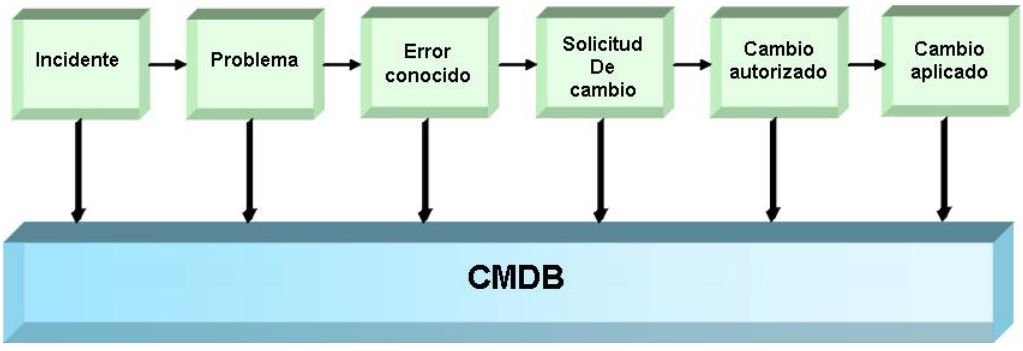
La Gestión de Configuración es una parte importante dentro de la administración de los servicios de TI. Sirve como eje central para compartir y ofrecer información. Aunque la información ofrecida por la Gestión de Configuración es utilizada por todos los procesos ITIL, es especialmente útil para la Gestión de Problemas, Gestión de Cambios, Gestión de Versiones y la Gestión de Incidentes.

Base de Datos de Configuración.

La base de datos de configuración (CMDB según sus siglas en inglés) se define como el repositorio que de acuerdo con Enterprise Management Associates (trans. 2006) “debe contener las relaciones entre todos los componentes incluyendo incidentes, problemas, errores conocidos, cambios y liberaciones. También contiene información sobre empleados, ubicación de proveedores y unidades de negocio”. En esencia la CMDB es en un mapa compuesto de toda la infraestructura y recursos operacionales, mostrando como estos se relacionan con servicios de negocio.

La base de datos de configuración no solamente es una herramienta útil para las disciplinas presentes en las categorías de Soporte de Servicios, sino para todo el resto de las disciplinas de ITIL, por ser una fuente de información confiable para. La figura 3 representa la relación entre Servicio de Soporte y la CMDB.

Figura 3: Relación de CMDB y Servicio de Soporte (tomado de Enterprise Management Associates, 2005).



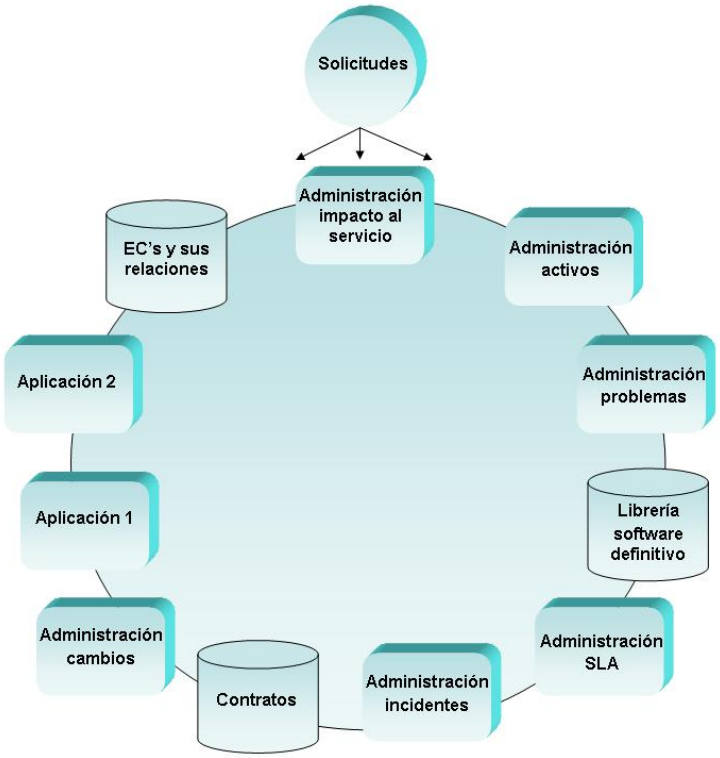
A pesar que el término base de datos de configuración implica una entidad con arquitectura tecnológica, ITIL no recomienda ninguna implementación en particular, de manera de mantener una postura neutral de la forma como las mejores prácticas son ejecutadas.

Evolución de la CMDB.

La concepción de la CMDB ha evolucionado desde principio de los años 90 hasta nuestros días desde repositorios aislados de información, repositorios integrados de información, una base de datos centralizada y única hasta llegar a lo que conocemos como CMDB federada. Cada uno de ellos se explica a continuación:

- Repositorios aislados de información: Al principio, la CMDB se formaba por un conjunto de aplicaciones que almacenaban su propia información y, comúnmente, otras bases de datos almacenando información de configuración. En la figura 4 se muestra la representación del concepto de repositorio aislado.

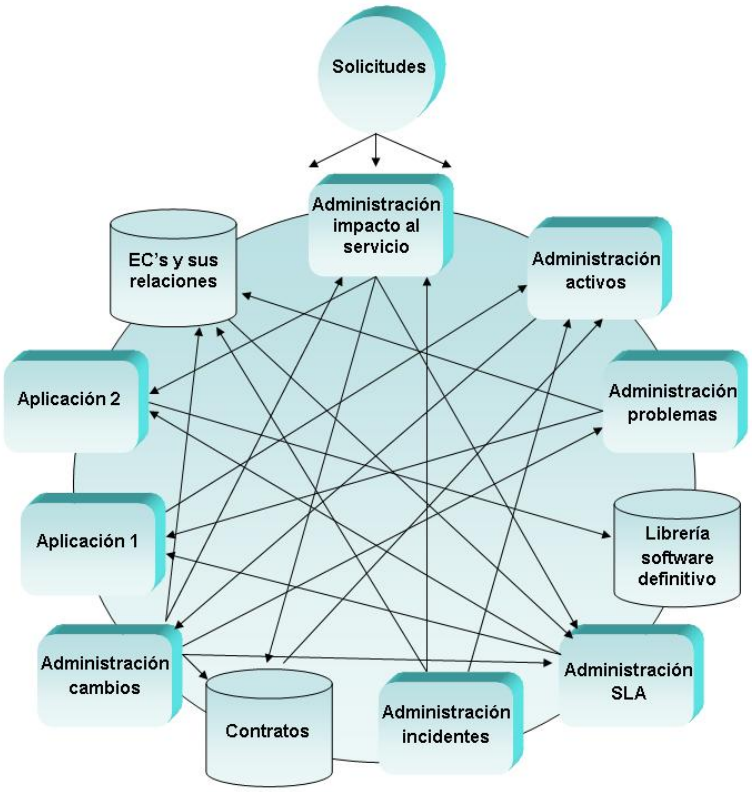
  Figura 4: Repositorios de información aislados (tomado de BMC Software, 2005).



Una limitación del enfoque es que al momento de requerir información, se tenía que realizar un gran esfuerzo en conocer donde se encuentra y accederla. Adicionalmente, este enfoque no permitía al usuario guardar información de los relaciones entre los EC.

- Repositorios integrados de información: Seguidamente, las organizaciones crearon las CMDB integrando las distintas fuentes de datos y aplicaciones, conectando cada consumidor de información con cada proveedor del cual requiriera datos. Este enfoque es mostrado en la figura 5.

Figura 5: Repositorios de información integrados (tomado de BMC Software, 2005).

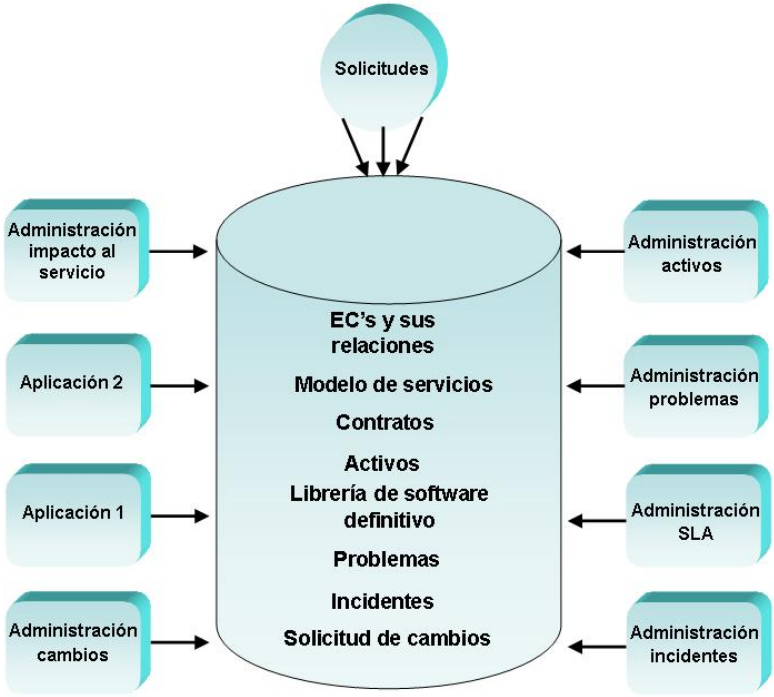


Gracias a la integración de fuentes de datos y aplicaciones, se permitió a los distintos procesos de la administración de la configuración compartir información, aumentando la utilidad de la CMDB. Sin embargo, eran necesarios muchos recursos para crear y mantener las distintas integraciones. Al igual que el enfoque anterior, alguien no familiarizado con la estructura podría no saber donde buscar para obtener información.

- Base de datos centralizada: Bajo este modelo, cualquier aplicación integrada con la CMDB (proveedores de información como consumidores por igual) pueden acceder a la información, lo cual lleva el compartir un paso más allá que la simple integración de repositorios. Se ofrece así un punto único de entrada para de datos, haciendo de la CMDB la fuente de registros a la cual los usuario pueden enviar todos las solicitudes.

  Si embargo, este modelo presenta desventajas. Requiere gran capacidad en un solo lugar, creando un cuello de botella en vista que todas las solicitudes y modificaciones de información pasan por la misma ruta. También requiere una migración masiva para colocar toda la información en una sola base de datos, creando un modelo de datos complicado que debe cambiar si cualquiera de las aplicaciones integradas con la CMDB cambia. La figura 6 representa este modelo de CMDB.

Figura 6: Base de datos centralizada (tomado de BMC Software, 2005).



- Modelo de datos federado: De acuerdo con Mueller (trans. 2006) este enfoque "se caracteriza por una base de datos centralizada enlazada con otros

repositorios de información con un modelo común de datos que trae la información desde un lugar a otro, sin la necesidad de re-escribir código".

De acuerdo a la misma fuente, este modelo "coloca los datos primarios y ampliamente compartidos de los EC en un repositorio común de información y federa información de atributos no críticos desde otras bases de datos". Algunos de los beneficios de este enfoque son los siguientes:

- La CMDB no se convierte en un cuello de botella: La carga generada por las solicitudes de información de los EC puede ser repartida entre diferentes repositorios de información.
- La información transaccional puede ser almacenada en bases de datos en mejor capacidad de manejar altos volúmenes de solicitudes, en lugar de una CMDB: La información es provisto de manera más eficiente. En lugar de colocar toda la información en una sola CMDB, los solicitantes de datos los obtienen de repositorios individuales optimizados para entregar el tipo de información requerida.
- No se requiere migrar datos relacionados o modificar la CMDB para contener esa información: Un nuevo tipo de información se almacena en una de los repositorios vinculados a la CMDB, evitando cambiar el modelo de esta última para dar cabida a un nuevo tipo de dato.

Características de una CMDB.

Con el objetivo de asegurar que la CMDB funcione como elemento de soporte a la toma de decisiones dentro de la empresa, existen una serie de características que requiere tener para poder administrar efectivamente los EC que contiene. Estas son:

- Federación de datos: El concepto de federación se refiere a un repositorio de datos central que directamente contiene la información básica de los EC, al mismo tiempo que establece vínculos con información presente en fuentes externas.

  Como beneficio de la federación de datos tenemos:

  - Se ahorra el esfuerzo de importar, seguir y reconciliar los datos en la CMDB.
  - Los datos federados pueden estar en múltiple locaciones.

- Se protege la inversión realizada en otros repositorios de datos.

- Modelo de datos flexible: La variedad de EC de configuración que normalmente forman la infraestructura tecnológica de las empresas es muy grande, desde computadores personales hasta equipos de red y servidores. Sin un modelo de datos que refleje estos tipos de EC y los tipos de relaciones que existen entre ellos, la CMDB almacenará atributos que no son pertinentes a sus EC, dejando por fuera los atributos importantes y haciendo más difícil de búsqueda de grupos de EC. Este modelo de datos debe ser tanto extensible como orientado a objetos.
  - Modelo de datos orientado a objetos: Un modelo de este tipo posee une jerarquía de clases en la cual una clase hereda atributos de su superclase situada por encima en la jerarquía, y entonces agrega sus propios atributos con el fin de crear un tipo de objeto más específico. Las subclases pueden tener sus propias subclases, permitiendo extender la jerarquía hasta el nivel de detalle que se requiera.
  - Modelo de datos extensible: La infraestructura y la tecnología que la soporte por lo general se encuentran en constante cambio. Es así que tanto los EC como las relaciones entre ellos son objetos de constante cambio, por lo que se requiere un modelo de datos que sea extensible. Se debe tener la capacidad de agregar nuevas clases o atributos, modificarlos o bien eliminarlos.
- Particionamiento de configuraciones: De acuerdo con BMC Software (trans. 2006) particionamiento “es la habilidad para dividir datos de configuración en piezas de clases llamadas datasets, cada una representando un conjunto de datos en un cierto momento determinado”. De esta manera es posible que el mismo EC existe en más de un dataset.

  De acuerdo a la misma fuente, el particionamiento de configuraciones resulta de mucha beneficio ya que permite representar:
  - Una configuración obsoleta.
  - Una configuración que ha sido probada y que se considera estable.
  - Una configuración futura.
  - Diferentes versiones de una configuración actual.
  - Subconjuntos de una configuración completa.

- Reconciliación de configuraciones: El grupo BMC Software (trans. 2006) explica este apartado de la manera siguiente:

  Cuando se tiene más de un dataset conteniendo las mismas instancias, reconciliación es el proceso de identificar las instancias concordantes en todos los datasets y entonces ya sea comparar las diferentes versiones de cada instancia y reportar las diferencias o bien converger los datasets en un nuevo dataset. Esto permite observar los cambios en el tiempo, o determinar una configuración deseada cuando se tienen múltiples fuentes.

  - Identificación de instancias: La explicación ofrecida por la misma fuente es la siguiente "Antes de comparar diferentes versiones de algo, se debe determinar si estas representan la misma entidad. La identificación cumple con esto, aplicando reglas que se especifican a las instancias de la misma clase en dos o mas datasets diferentes".
  - Comparación de datasets: Al respecto, BMC Software (trans. 2006) ofrece la siguiente explicación: "Una actividad de comparación opera sobre instancias en dos datasets y genera un reporte indicando las instancias que aparecen solo en un dataset y detalla diferencias entre instancias que aparecen en ambos."

    Un uso de la comparación es alertar sobre un cambio en una configuración que se espera permanezca estática. Alternativamente, si se tiene un cambio en progreso, se puede emplear la comparación para conocer si el cambio alcanza el nuevo estado esperado.
  - Convergencia de datasets: De acuerdo a la misma fuente, "la convergencia toma dos o más datasets y los fusiona en un nuevo dataset de acuerdo a reglas de precedencia especificadas". Es de utilidad en casos donde diferentes aplicaciones administrativas proveen información que se solapa sobre un mismo EC.

    Las reglas de precedencia establecen valores de peso para unas clases y atributos en particular, esto es, si en cualquier

dataset, una clase o atributo en particular posee el peso más alto, entonces esta clase o atributo tendrá su valor colocado en el dataset resultado de la convergencia.

- Acceso abierto a datos: La información contenida en la CMDB es inútil de no estar disponible a usuarios y aplicaciones. Es importante recordar el permitir a ambos grupos leer y escribir en la CMDB. Para tal fin es necesario contar con las siguientes características:
  - Acceso programático: La CMDB debe proveer una interfaz de aplicación o algún otro método para programas, de manera que puedan consultar o modificar los datos de las instancias y las clases de su modelo de datos.
  - Carga masiva de datos: La CMDB debe proveer una forma para importar múltiples instancias en una sola operación, de manera que las aplicaciones puedan rápidamente insertar información en ella.
  - Independencia de plataforma y base de datos: La CMDB debe ser compatible con distintos sistemas operativos y bases de datos, y así obtener mayor flexibilidad con el ambiente.

Objetivos de una CMDB.

Aparte de servir como fuente confiable de información para las categorías de Soporte de Servicios en ITIL, los objetivos de la CMDB se puede resumir como sigue:

- Suministrar una visión simple de la infraestructura de TI y sus activos, así como los procesos de negocio que dependen de ellos.
- Permitir conocer las relaciones que existen entre los diferentes EC que forman parte de la infraestructura de TI.
- Permitir a los administradores de TI conocer como un cambio en cualquiera de los EC puede afectar el funcionamiento de algún otro o de los procesos de negocio relacionados a ellos.
- Reducir costos relacionados con la administración de activos, en virtud que se conoce con exactitud la disponibilidad y el uso que se da a cada uno, evitando así llegar al sobre-aprovisionamiento en alguno de estos.

Beneficios obtenidos de una CMDB.

Los beneficios ofrecidos por la implantación de una base de datos de configuración están muy enlazados por con los obtenidos con la aplicación de la administración de la configuración, en virtud que una CMDB se erige como la columna vertebral sobre la cual funciona esta última.

De acuerdo con BMC Software (trans. 2006), algunos beneficios son:

- Mejoramiento de la calidad de servicio y soporte al servicio más confiable: No es posible mejorar la calidad del servicio sin tener conocimiento de los componentes que conforman este último. Precisamente la CMDB provee la `configuración para cada uno de los servicios y establece el fundamento sobre el cual es posible mejorar la calidad en cada uno de ellos.
- Mejoramiento en los procesos de continuidad de servicio: Supóngase el caso donde una oficina con alrededor de 200 estaciones de trabajo se incendia, ocasionando pérdida total de equipos, ¿cómo saber la tecnología que estaba instalada en esa oficina y la manera es que estaba configurada?, sin esa información, ¿cómo puede la oficina ser reconstruida?. La CMDB provee tanto los detalles tecnológicos como los detalles de configuración para todas las localidades, ofreciendo un excelente punto de partida para la reconstrucción.
- Visión clara de la capacidad de la TI: Para lograr esto, se debe conocer el estado completo de la infraestructura de TI. Una de las funciones de la CMDB es mantener registro del estado de todos los elementos de infraestructura, los cuales sufren constantes cambios.
- Mejoramiento de la información relacionada a los servicios: Dentro de la CMDB, cada elemento de configuración posee uno o más atributos que describen con cual(es) servicio(s) se relaciona.
- Mayor flexibilidad para el negocio por medio de un entendimiento mejorado del soporte de TI: Al tener control de los recursos, mientras más información se posea, se gana mayor flexibilidad en el negocio. Al conocer todos los elementos que se deben considerar en un cambio potencial, entonces se puede asegurar que esos elementos cumplen los requerimientos de un cambio una vez este sea implantado. Esta es la razón por la cual las relaciones en la CMDB son tan importantes. Una vez se conozca el

componente que será cambiado, la CMDB muestra todos los componentes relacionados. De esa manera se conoce si es necesario modificar esos componentes relacionados para asegurar su funcionamiento con el componente originalmente modificado.

- Incremento de la motivación y satisfacción de los empleados por medio de un mejor entendimiento de la capacidad y mejor administración de expectativas: Al contrario de la creencia popular, la mayoría de las personas prefieren laborar en un ambiente organizado y estructurado. Confían en que prácticamente ningún cambio falla o crea nuevos incidentes, de manera que el servicio de mesa de ayuda puede concentrarse en las necesidades de los clientes, en lugar de distraerse en la administración de incidentes. La CMDB provee las bases para contar con información confiable y oportuna que contribuye enormemente a la motivación de los empleados y alcanzar mayores niveles de satisfacción.
- Incremento en la satisfacción del cliente, ya que los proveedores de servicios conocen y entregan lo que se espera de ellos: La CMDB provee el conocimiento, mientras el resto de ITIL provee la entrega. Sin embargo, entrega de servicio sin conocimiento esta destinada al fracaso.

Modelo de Información Común.

Uno de los asuntos importantes que surge al hablar de EC presentes en una CMDB es la manera como estos son representados. Para ello, el estándar más frecuentemente referenciado es el modelo de información común (CIM por sus siglas en inglés) propuesto por la Fuerza de Tarea de Administración Distribuida (DMTF por sus siglas en inglés), organización sin fines de lucro que agrupa un número importante de miembros de la industria dedicados a promover la administración de empresas y sistemas así como la interoperabilidad.

CIM es un modelo de información, una vista conceptual de un ambiente administrado, que intenta unificar y extender la instrumentación existente en diferentes estándares de administración, aplicando para ello la estructura básica y técnicas de conceptualización del paradigma orientado a objetos. CIM no requiere una instrumentación o formato de repositorio en particular. Es solamente un modelo de

información, unificando los datos, empleando un formato orientado a objeto, haciéndolo disponible desde cualquier número de fuentes.

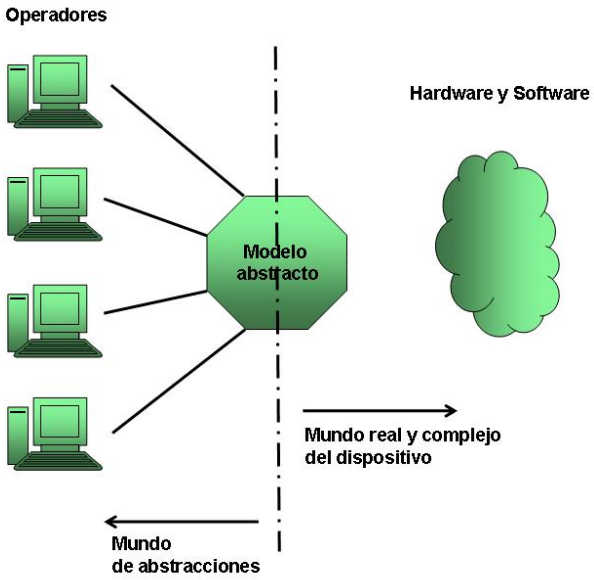
Muy unido a CIM se encuentra la Administración de Empresas Basada en Web (WEBEM por sus siglas en inglés), que constituye un conjunto de estándares de tecnología de internet y administración desarrollados para unificar la administración de ambientes de computación distribuidos. WEBEM provee la habilidad para intercambiar información de CIM de manera eficiente e interoperativa, incluyendo para esto protocolos, lenguajes de consulta y mapeo.

El concepto de modelo.

La creación de un lenguaje apropiado para expresar ideas ha sido desde siempre un elemento retador para científicos e ingenieros. Por ello se requiere seleccionar un lenguaje en el cual expresar los conceptos para administrar dispositivos y servicios, que sea lo suficientemente rico para incluir las complejas y abstractas relaciones que existen entre elementos, y simple para que diferente software pueda manejarlo eficientemente.

Para explicar el concepto de modelo nos remitimos a la figura 7. La información que permite a un dispositivo operar correctamente es almacenada en el lado derecho de la figura. Esta información puede estar almacenada en registros de un circuito integrado, en la memoria de un computador, almacenada en un solo lugar o estar distribuida.

Figura 7: El rol del modelo (tomado de Hobbs, C, 2004).



En el lado izquierdo de la figura se muestra una serie de operadores. Ellos tienen diferentes modelos mentales o ideas del dispositivo que ellos están administrando, y puede ser apropiado para el sistema de administración reflejar el modelo mental de los operadores incluso si es equivocado o incompleto. Es importante que los mensajes que les llegan a los operadores concuerden con la idea que poseen del dispositivo que administran.

Para ejemplificar esto, Hobbs (trans. 2006) señala lo siguiente:

> Si el operador pertenece al negocio de planificar el pintado de casillas telefónicas, entonces su modelo mental del dispositivo es de una cabina metálica de un cierto color, pintada en una fecha determinada. Si, por otro lado, la responsabilidad del operador es configurar un dispositivo de telecomunicaciones dentro de la cabina, entonces jamás pintara la cabina, dejando el color que posee.

El modelo se sitúa precisamente entre los diferentes modelos mentales de los operadores y la representación real de la información. Este intenta ser lo suficientemente abstracto y consistente para soportar los diferentes puntos de vista y al mismo tiempo ser lo más cercano posible a los dispositivos reales de manera de ser eficiente.

En líneas generales un modelo debe:

- Ser expresado en un lenguaje formal de manera que pueda ser manipulado por un computador. Debe también tener una representación gráfica que permita a los humanos comprenderlo más fácilmente.
- Deber ser capaz de amoldarse a las diferentes necesidades de todos los operadores, suministrando a todos información útil.
- Debe ser capaz de expresar abstracciones de alto nivel (servicios) sin caer en el detalle de los elementos administrados.
- Debe soportar el modelado de colecciones de entidades y de asociaciones entre esas entidades cuando estas son relevantes a una aplicación administrada.

Conceptos de modelado.

La mayoría de los profesionales han tomado términos del lenguaje común y les han asignado diferentes significado según su campo de especialidad. Por ejemplo clase para un programador no tendrá el mismo significado que para un profesional de la educación. De la misma manera el modelado ha construido su jerga en base a palabras del lenguaje cotidiano y les ha asignado un significado que no siempre coincide con el uso común que se les da.

A continuación se exponen los conceptos empleados por CIM:

- Clase: De acuerdo con Hobbs (trans. 2006), “el concepto de clase es fundamental para cualquier cosa. Una clase es una amalgama de características que definen un grupo particular de cosas”.
- Propiedades: Una propiedad es un valor empleado para denotar una característica de una clase. Una propiedad se limita a la clase donde se define y debe ser única dentro de ella.
- Calificador: Tal como lo define Hobbs (trans. 2006), “los calificadores ofrecen información adicional sobre clases, asociaciones, propiedades, etc. Ellos pueden especificar [por ejemplo] la longitud máxima de una propiedad de tipo cadena de caracteres, el valor máximo de una propiedad numérica, si una propiedad puede ser modificada o solo leída”.
- Herencia: Es la relación existente entre dos clases (subclase y superclase) donde todos los miembros de la subclase se requiere sean miembros de la superclase. Cualquier miembro de la subclase debe soportar cualquier método o propiedad de la superclase.
- Instancia: Es una unidad de datos. Una instancia es un conjunto de propiedades con valores que puede ser identificado de manera única.
- Método: Según Hobbs (trans. 2006) “un método es otra característica de una clase. Representa una función que a una instancia de la clase se le puede solicitar que ejecute”. Un método se limita a la clase donde se define y debe ser único dentro de ella.
- Asociación: Es una clase que expresa la relación entre dos o más clases.

Beneficios de CIM.

Como se menciono en un apartado anterior, CIM aplica la estructura básica y técnicas del paradigma orientado a objetos, que es precisamente de donde se desprende su valor, teniendo las siguientes capacidades de las que carecen otros modelos planos:

- Abstracción y clasificación: Para reducir el dominio de un problema, se definen conceptos fundamentales de nivel superior (es decir, los objetos del dominio de la administración). Estos objetos son agrupados en tipos (clases) identificando características comunes (propiedades), relaciones (asociaciones) y comportamiento (métodos).
- Herencia de objetos: Al heredar de objetos de nivel superior, una subclase obtiene toda la información (propiedades, métodos y asociaciones) de sus objetos de nivel superior, pudiendo además agregar información extra. Las subclases son creadas para colocar el nivel de detalle y complejidad adecuado en el nivel adecuado dentro del modelo.
- Habilidad para mostrar dependencias y asociaciones entre componentes: Las relaciones entre componentes constituyen conceptos verdaderamente poderosos. Antes de CIM, los estándares de administración capturaban las relaciones en estructuras de datos muy complejas. El paradigma orientado a objetos ofrece una aproximación más elegante, donde las relaciones entre los objetos son modeladas directamente. Adicionalmente, la manera que estas relaciones son nombradas y definidas, describen la semántica de las asociaciones entre objetos. Mayor semántica e información puede ser provista en las propiedades de una asociación.
- Capacidad para definir métodos estándares y heredables: La habilidad para definir el comportamiento de los objetos (métodos) de forma estandarizada puede ser visto como otra forma de abstracción. Así mismo empaquetar este comportamiento con los datos de los objetos es encapsulamiento. Se cuenta así con estándares altamente flexibles capaces de invocar un método “Reset” en un dispositivo bloqueado o bien el método “Reboot” en un computador en la misma situación, independientemente del hardware presente, el sistema operativo o la marca del dispositivo.

El Formato de Objeto Administrado.

Un modelo es expresado en CIM de manera textual empleando un lenguaje llamado Formato de Objeto Administrado (mof por sus siglas en inglés).

De acuerdo a Hobbs (trans. 2006) “mof es un lenguaje textual empleado, como un UML [lenguaje de modelado unificado], para describir modelos CIM. Generalmente contiene mayor detalle que el equivalente diagrama UML, pero es menos fácil para un humano leerlo”.

El formato de objeto administrado establece la sintaxis para escribir las definiciones. Los principales componentes de una especificación mof son descripciones textuales de clases, propiedades, asociaciones, referencias, métodos y declaraciones de instancias. A continuación un resumen de la sintaxis seguida en mof para la declaración de estos elementos.

- Declaración de clase: La sintaxis para especificar una clase en mof es muy similar a su contraparte en lenguaje de programación C++:

  [<Calificadores>]
  Class <nombre de la clase> : <nombre de la superclase>
  {
  <propiedad o método>;
  <propiedad o método>;
  …..
  <propiedad o método>;
  };

- Declaración de propiedades: Cada propiedad tiene el siguiente formato:

  [<calificadores>]
  <tipo> <nombre> { = <valor por defecto> };

- Declaración de métodos: Cada método tiene el siguiente formato:

  [<calificadores>]
  <tipo de retorno> <nombre del método>(<parámetro>, … <parámetro>);

  Donde cada parámetro consiste en una lista de calificadores en corchetes, seguidos por un tipo, seguido por el nombre del parámetro.

En al anexo A se encuentra una notación formal para describir la sintaxis de mof, llamada mof BNF. Esta notación, introducida por John Backus y Meter Naur, se ha

convertido en la manera más común de expresar lenguajes en la ciencia de la computación.

En el anexo B se encuentra la tabla completa con los distintos tipos de datos intrínsecos válidos en mof empleados en la definición de propiedades, valores de retorno y parámetros de los métodos y en las asociaciones.

Representación UML de CIM.

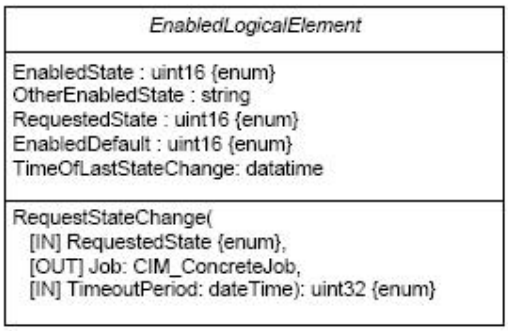
El lenguaje de modelado unificado (UML por sus siglas en inglés) es según Hobbs (trans. 2006) “un lenguaje gráfico estandarizado por el Grupo de Administración de Objeto [OMG por sus siglas en inglés] el cual es empleado para ‘visualizar, especificar, construir y documentar’ la estructura de un sistema”.

Existen distintos símbolos para todos los elementos empleados dentro de CIM. Seguidamente se detalla la manera que estos son representados empleando UML:

- Clase: La definición de una clase es representada en UML por un rectángulo. Este rectángulo se divide en tres partes:
    1. La subdivisión superior contiene el nombre de la clase. El nombre puede ser escrito en itálica para indicar que la clase no tiene instancias, es decir, solo puede ser una superclase de otras clases.
    2. La subdivisión central lista las propiedades que caracterizan a la clase.
    3. La subdivisión inferior lista algunas o todas las acciones (métodos) que la clase puede ejecutar.

    En la figura 8 se muestra la representación de una de las clases empleadas en CIM, donde se describen fácilmente los diferentes elementos que la componen, esto es, el nombre de la clase, sus propiedades y métodos.

Figura 8: Representación de una clase en UML.



- Asociaciones: A pesar que una asociación es una clase in CIM, no es representada empleando un rectángulo. Existen dos tipos de asociaciones, a saber:
  1. Agregación: Representada por una línea verde con un diamante en el lado de la clase "agregada". Una agregación es también conocida como una relación "tiene un" o "es miembro de".

     Una relación de composición, representada por una línea verde con un diamante y círculo relleno en el lado de la clase agregada redefine una agregación a una relación "todo/parte".
  2. No agregación: Representadas por una línea roja (sin flechas) entre las clases relacionadas.

  La cardinalidad de las asociaciones a ambos lados, esto es, el número de clases envueltas en la relación, se indican por valores numéricos o un asterisco (*). Las siguientes cardinalidades son empleadas comúnmente en CIM:

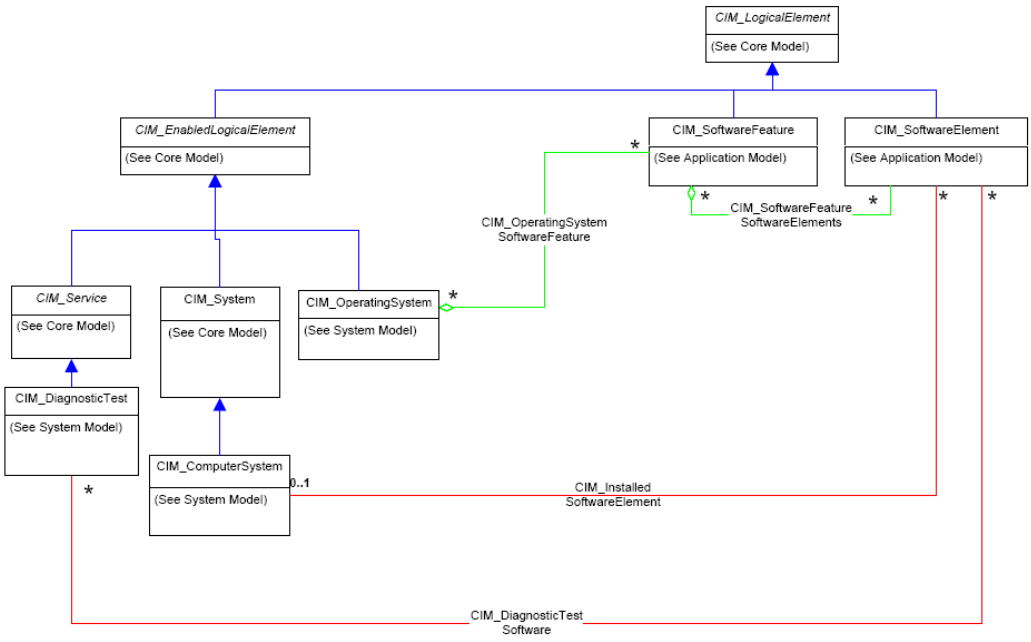
0..1	Indica una referencia opcional de un solo valor
1	Indica una referencia requerida de un solo valor
1..n ó 1..*	Indica ya sea una referencia simple o multi valorada que es requerida.
, 0..n ó 0..	Indica una referencia opcional ya sea simple o multivalorada.

  En la tabla anterior, una referencia "requerida" significa que el objeto y la asociación deben ser instanciados cuando la otra clase referenciada es instanciada.

  El nombre de las asociaciones usualmente se coloca cerca del centro de la línea.
- Herencia: Representadas con líneas azules con flechas, también son conocidas como relaciones del tipo "es un". Las relaciones de herencia no son etiquetadas o nombradas.

La figura 9 es la representación gráfica de las asociaciones en sus diferentes tipos y de la herencia entre clases.

Figura 9: Representación de asociaciones y herencia entre clases UML.



MARCO METODOLÓGICO

Objetivo General.

Construir el repositorio donde consultar información sobre los diferentes aplicativos/servicios desarrollados por la gerencia, sus componentes y relaciones entre ellos.

Tipo de Investigación.

El tipo de investigación corresponde a Estudio Aplicado, ya que como lo indica Landeau (2005), este tipo de estudio "corresponde a la asimilación y aplicación de la investigación a problemas definidos en situaciones y aspectos específicos".

Generalmente, el Estudio Aplicado, también conocido como activo o dinámico, es empleado con frecuencia en el contexto industrial o empresarial, orientado a la generación de materiales, sistemas, instrumentos, métodos y modelos útiles en la solución de una problemática identificada.

Método.

El siguiente proyecto consiste en la construcción de un repositorio de información sobre los diferentes aplicativos/servicios desarrollados por la Gerencia de Banca Virtual, sus componentes y relaciones entre ellos. Gracias a esto se puede identificar de mejor manera la forma como los cambios en algunos de estos componentes de software puede afectar la continuidad de los servicios que lo emplean.

Definición de Variables.

Previo a la descripción de las actividades necesarias para alcanzar el objetivo, se ofrecen una serie de definiciones importantes de ITIL, CIM y desarrollo de software, que serán de utilidad para la realización del proyecto. Seguidamente se presentan las definiciones elaboradas por Salischiker (2005) en relación a ITIL.

Elemento de configuración (EC): "Componente de una infraestructura que está (o tiene que estar) bajo el control de la Gestión de Configuración. Los EC pueden variar en complejidad, tamaño y tipo de un sistema completo (incluyendo todo el hardware, software y documentación) hasta un módulo simple o un componente menor de hardware".

Herramienta para la gestión de configuración: “Un producto de software que provee soporte automatizado para control de cambios [modificaciones], configuraciones o versiones [de elementos de configuración]”.

Impacto: “Medida de criticidad sobre el negocio de un incidente. A menudo igual al grado con que un incidente distorsiona el nivel de servicio acordado o esperado”.

Incidente: “Cualquier evento que no es parte de la operación estándar de un servicio que causa, o puede causar, una interrupción de ese servicio o una disminución de la calidad del mismo”.

Problema: “Causa subyacente desconocida de uno o más incidentes”.

Liberación: “Una colección de elementos de configuración nuevos o modificados que están probados y se introducen en conjunto en el ambiente de producción”.

Versión: Según Salischiker (2005) el concepto de versión puede ser “usado por los elementos de configuración software [componente de software] para definir una identificación específica liberada para el desarrollo, revisión o modificación. Prueba o producción”.

Se presentan las definiciones importantes propias de CIM expuestas por Distributed Management Task Force (trans. 2006).

Cardinalidad: “Una relación entre dos clases que permiten que más de un objeto se encuentre relacionado a un objeto simple”.

Clave (Key según su denominación en inglés): “Una o más propiedades de un objeto calificado que permiten identificar de manera única instancias de ese objeto”.

Referencia: “Tipos especiales de propiedades que son referencias o ‘apuntadores’ a otras instancias”.

Espacio de nombre (Namespace según su denominación en inglés): “Un objeto que define el alcance dentro del cual las claves de un objeto deben ser únicas”.

Entre los conceptos propios de hardware y desarrollo de software están:

Sistema Operativo: Es un conjunto de programas orientados a permitir la comunicación entre el usuario y un computador. También se encarga de gestionar los recursos del computador de manera eficiente. El sistema operativo comienza a trabajar desde el mismo momento de encendido del equipo y gestiona el hardware de este desde los niveles más bajos.

Dirección IP (Internet Protocol según su denominación en inglés): Es un número que identifica de manera lógica y jerárquica a un dispositivo (por lo general una computadora) dentro de una red que utilice el protocolo Internet. Habitualmente una persona que se conecta desde su hogar a Internet emplee una dirección IP.

Encriptación: Es el proceso que permite que la información sea cifrada de manera que el resultado sea ilegible a menos que se cuente con los datos necesarios para interpretarla.

Servidor: Es un computador donde se ejecutan programas que realizan tareas en beneficio de otra aplicación o computadores (generalmente denominados clientes). Por ejemplo, una tarea de un servidor es mantenerse a la espera de peticiones de páginas HTML provenientes de los clientes, entregándoles el contenido que estos solicitan.

Tecnología ASP (Active Server Pages según su denominación en inglés): Ambiente de aplicación abierto y gratuito donde se puede combinar código HTML, scripts (guiones) y DLL's para crear aplicaciones para la web.

Guión (Script según su denominación en inglés): Es un programa que puede acompañar a un documento HTML o que puede estar incluido en él. Este programa debe tener una nomenclatura determinada o forma de escribirse, siendo capaces de realizar alguna tarea que ayude en el funcionamiento/presentación del documento HTML.

ASP.NET: Conjunto de tecnologías de desarrollo de aplicaciones web comercializado por Microsoft. Es empleado por los programadores para construir sitios web tanto domésticos como empresariales y servicios web.

Memoria caché: Corresponde a una pequeña porción de memoria muy rápida, cuyo objetivo es disminuir los tiempos de acceso a los datos presentes en ella.

Memoria caché global de ensamblado (GAC por sus siglas en inglés): Es una memoria empleada para almacenar ensamblados para ser utilizados y compartidos por diferentes aplicaciones/servicios presentes en un equipo.

Microsoft .NET: Es un proyecto de Microsoft orientado a crear una nueva plataforma para el desarrollo de software con independencia de plataforma y que permite un rápido desarrollo de aplicaciones

Lenguaje de marcado extensible (XML por sus siglas en inglés): Consiste en un lenguaje de marcado que ofrece un formato para la descripción estructurada de datos, organizando un documento con información en diferentes partes. Es un lenguaje especializado en la gestión de la información en internet.

Lenguaje de marcas hipertextuales (HTML por sus siglas en inglés): En un lenguaje de marcado diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web.

Lenguaje de descripción de servicios web (WSDL por sus siglas en inglés): Es un lenguaje basado en XML empleado para describir las capacidades de un servicio web.

Bibliotecas de enlace dinámico (DLL por sus siglas en inglés): Consisten en un conjunto de código ejecutable común que puede estar compartido entre varias aplicaciones.

Ensamblado: Es un concepto que aparece en .NET y consiste en un fragmento de código ejecutable (por ejemplo una DLL) junto a un fichero que indica que contiene el ensamblado, que versión es, etc. De esta manera es posible instalar versiones diferentes del mismo ensamblado y que cada programa cargue la que necesite empleando la información presente en el fichero.

Archivo Web.config: La definición ofrecida por Duthie (2002) indica que “el Web.config es un archivo XML, legible tanto por las personas como por las máquinas, que almacena los valores de configuración de una aplicación (o una parte de una aplicación)”.

Archivo Machine.config: De acuerdo a Duthie (2002) “este archivo de configuración maestro contiene los parámetros predeterminados de todas las aplicaciones ASP.NET del servidor. Este archivo también contiene los parámetros para la configuración del equipo (como los enlaces remotos y los enlaces de ensamblados), así como otros parámetros”.

Servicios Web (Web Services según su denominación en inglés): Es un segmento de código programado accesible a través de internet. Permite a los programadores desarrollar servicios programables que estén disponibles para otros programadores en internet.

Método.

Para alcanzar el objetivo propuesto, esto es, contar con el repositorio de información, se ejecuta toda una serie de actividades correspondiente a las etapas de inicio, ejecución y seguimiento del proyecto. Esto con la finalidad de asegurar contar en un principio con todos los insumos necesarios que permiten pasar a la etapa de

diseño/desarrollo, y posteriormente, establecer las pautas que permitan evaluar el desempeño/adecuación del repositorio así como su uso.

1. Definición del criterio para seleccionar los elementos de configuración que forman parte del repositorio.
2. Selección de los elementos de configuración que forman parte del repositorio así como la convención para su identificación.
3. Definición de los atributos, características funcionales o físicas de los componentes de software o servicios que son registrados en el repositorio, como por ejemplo, ubicación del código fuente, problemas o incidentes relacionados con ellos derivados o no de algún cambio, responsable técnico, relaciones con otros componentes/servicios, etc.
4. Definición de los tipos de relaciones consideradas entre los EC.
5. Identificación de las diferentes herramientas tecnológicas, datos y procesos existentes empleados para la gestión de configuración.
6. Identificación de las interfases entre el repositorio o CMDB y las diferentes herramientas para la gestión de configuración.
7. Diseño del modelo de datos del repositorio o CMDB.
8. Personalización del repositorio e inicio de carga del mismo.
9. Definición y documentación de los procedimientos para monitorear el progreso de la gestión de configuración y asegurar que la CMDB es utilizada de manera efectiva y eficiente.
10. Definición y documentación de roles, responsabilidades y capacidades del personal envuelto en la gestión de configuración.
11. Seguimiento de todos los cambios que se realizan sobre los EC.

Criterios de Selección de los Elementos de Configuración.

Forman parte del repositorio todos los EC que cumplen al menos uno de los siguientes criterios:

1. Aquellos elementos cuyos valores definen el funcionamiento de algún servicio: Como ejemplo están los archivos de configuración donde se encuentran los parámetros de conexión a una base de datos.
2. Elementos que son expuestos a los clientes de la institución (ahorristas, tarjeta habientes, etc) o que ayudan a exponer un servicio a ellos: Un

ejemplo son los servicios web empleados directamente por algunos clientes de la organización.

3. Componentes de software desarrollados por la Gerencia de Nuevos Proyectos o la Gerencia de Soporte y Mantenimiento.
4. Equipo de computación donde se localice un EC que cumpla al menos unos de los criterios anteriores y que además permite que estos EC sean utilizados por los clientes de la organización.

Selección de los Elementos de Configuración.

Los elementos de configuración que forman parte de la CMDB son los siguientes:

- Librerías de enlace dinámico.
- Archivos de configuración (Por ejemplo, archivos web.config y machine.config).
- Servicios Web.
- Páginas activas de servidor (archivos con extensión .asp).
- Páginas activas de servidor .NET (archivos con extensión .aspx).
- Archivos HTML.
- Archivos de guiones (scripts).
- Servicios/aplicaciones web desarrolladas por la Gerencia de Nuevos Proyectos o la Gerencia de Soporte y Mantenimiento.
- Eventos que afectan el normal funcionamiento de los servicios/aplicaciones.
- Los equipos de computadores donde se localizan algunos de los elementos mencionados anteriormente.
- La agrupación lógica realizada para los equipos de computadores mencionados en el punto anterior.

La identificación de cada uno de los EC se obtiene según el siguiente esquema:

Espacio de nombre_NombreEC

El espacio de nombre de la base de datos de configuración es Banesco.

Por otro lado, la identificación de las asociaciones se realiza en base al siguiente patrón:

Espacio de nombre_NombreEC1NombreEC2

Donde NombreEC1 y NombreEC2 corresponden a los nombres de los dos elementos de configuración relacionados por intermedio de la asociación. Si en la asociación, si solo uno de los EC tiene máximo una instancia permitida, este va de primero en el nombre, de no ser el caso, no importa el orden de colocación.

Representación UML del Modelo de Datos del Repositorio.

A continuación se expone la representación gráfica del modelo de datos del repositorio, desarrollado con el uso de UML. Esta nos permite observar fácilmente la definición realizada de cada una de las clases (atributos, tipos de datos, claves) así como de las asociaciones que las relacionan (nombre y cardinalidad).

No se hace uso de colores para identificar el propósito de las líneas en el diagrama, ya que como fue explicado con anterioridad, aunque su uso está permitido, no forma parte del estándar de UML.

Figura 10: Representación UML clases y asociaciones del repositorio (primer gráfico).

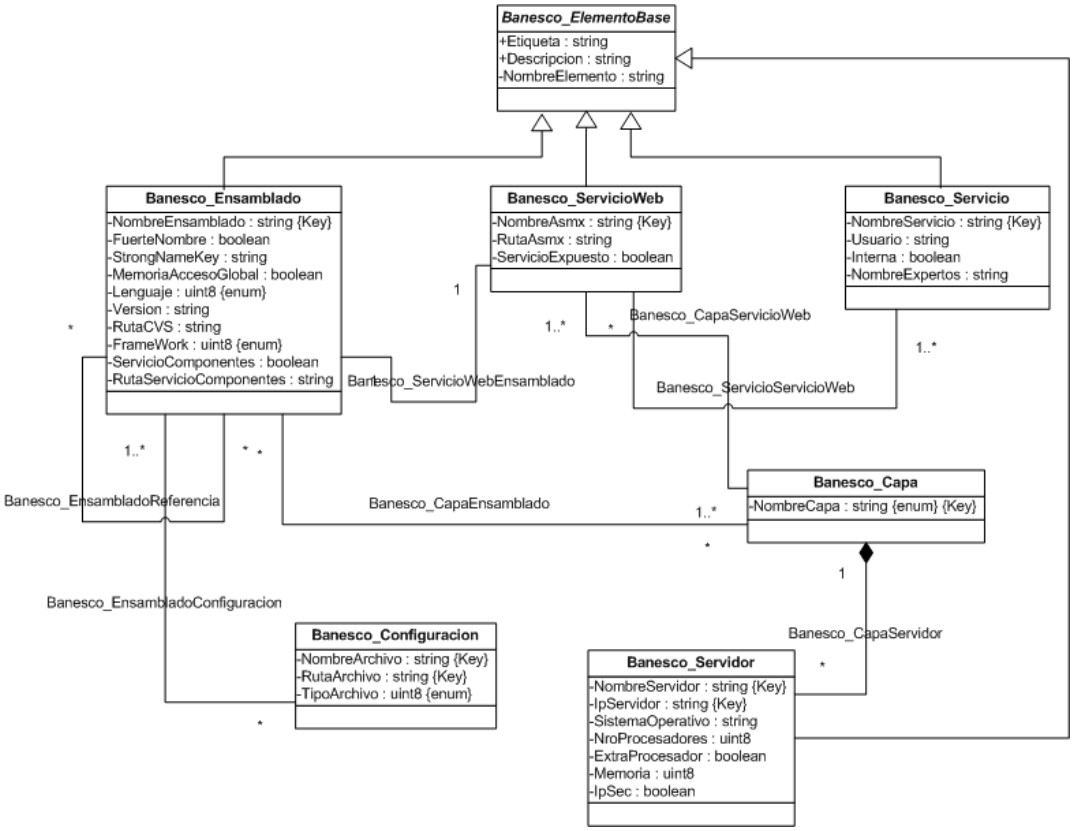


Figura 11: Representación UML clases y asociaciones del repositorio (segundo gráfico).

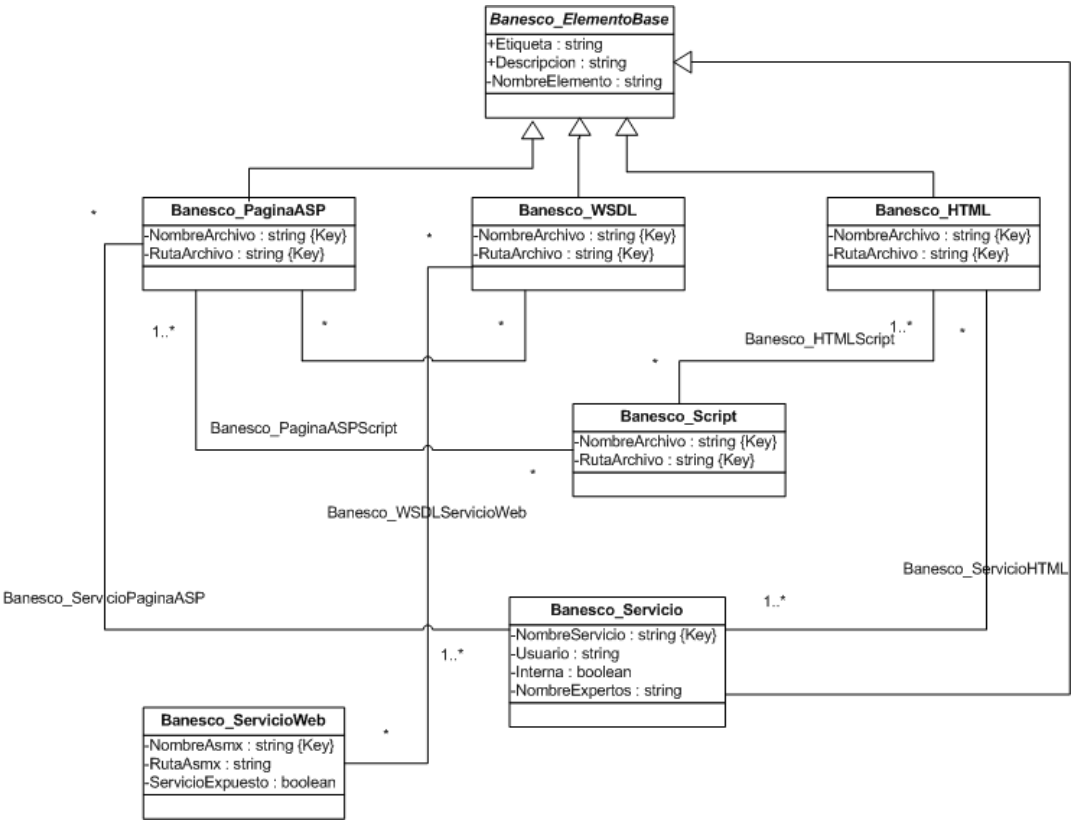


Figura 12: Representación UML clases y asociaciones del repositorio (tercer gráfico).

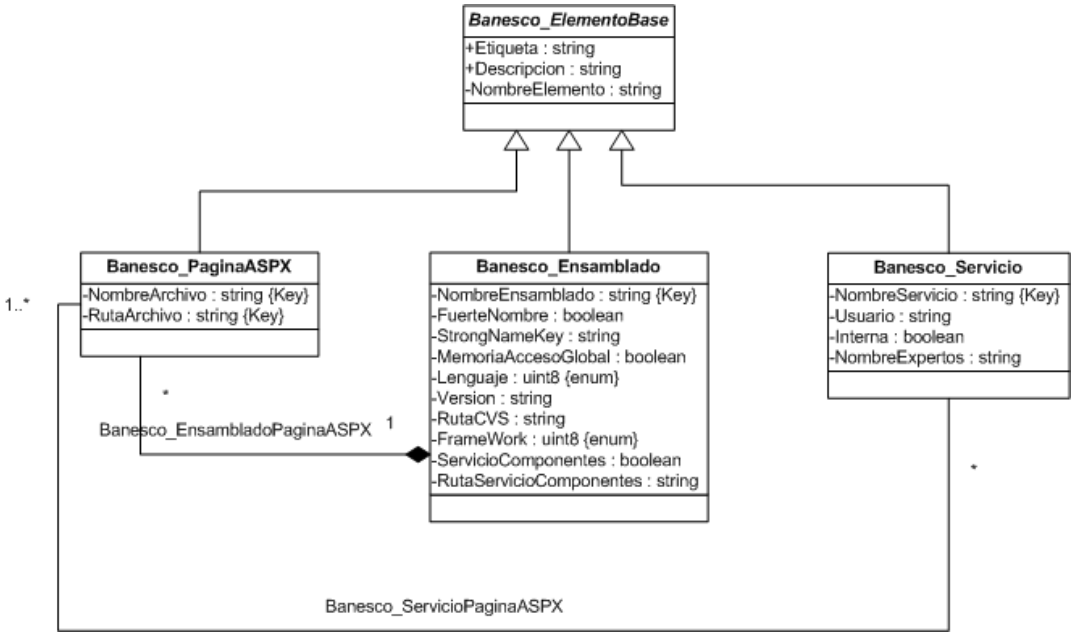
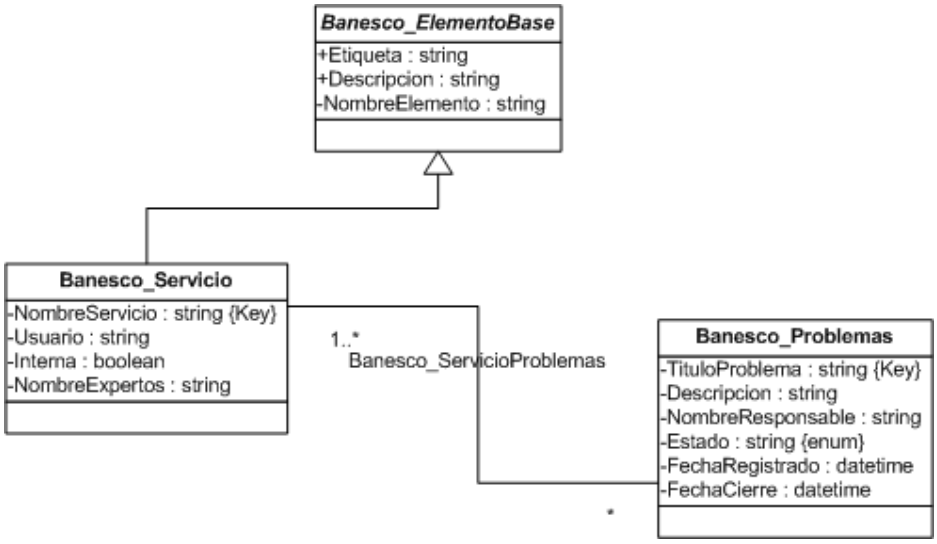


Figura 13: Representación UML clases y asociaciones del repositorio (cuarto gráfico).



Representación en Formato de Objeto Administrado del Modelo de Datos del Repositorio.

Como se ha indicado en apartados anteriores, un modelo de datos basado en CIM puede ser representado en forma textual y en forma gráfica. Para la representación textual se emplea lo que se denomina Formato de Objeto Administrado.

A continuación se presenta la definición de todos y cada uno de los elementos que forman parte de repositorio, en este caso, las definiciones tanto de las clases como de las asociaciones que las relacionan.

Las declaraciones de las clases llevan consigo los atributos que las caracterizan, el tipo de dato y la descripción para cada uno de ellos.

Las asociaciones presentan las referencias a las clases relacionadas, una breve descripción así como la cardinalidad.

- Clase Banesco_ElementoBase: Clase abstracta que sirve de base a todos los elementos de configuración dentro del repositorio.

```
class Banesco_ElementoBase
{
        [Description ("Descripción corta de la clase"), maxlen(64)]
        String Etiqueta;
        [Description ("Propiedad que provee una descripción completa"
```

```
        "de la clase")]
        String Descripcion;
        [Description ("Nombre amigable para el objeto. Esta propiedad"
        "permite a casa instancia definir un nombre amigable aparte de"
        "sus propiedades clave")]
        String NombreElemento;
};
```

- Clase Banesco_Ensamblado: Clase que representa los diferentes ensamblados desarrollados por la Gerencia de Nuevos Proyectos o la Gerencia de Soporte y Mantenimiento.

```
class Banesco_Ensamblado: Banesco_ElementoBase
{
        [Key, Description ("Nombre completo del ensamblado, Banesco.Servicios.Transaccional.BaseFinancieraLib.dll"), MaxLen(64)]
        String NombreEnsamblado;
        [Description ("Namespace asociado en el código al ensamblado"), MaxLen(64)]
        String Namespace;
        [Description ("Indica si el ensamblado es fuertemente nombrado con un SNK")]
        Boolean FuerteNombre = "false";
        [Description ("Nombre del archivo con el cual se realiza el nombrado fuerte"), MaxLen(64)]
        String StrongNameKey;
        [Description("Indica si el ensamblado se encuentra publicado en la memoria cache global de ensamblado, GAC"), Required(true)]
        Boolean MemoriaAccesoGlobal = "false";
        [Required (true), Description("Lenguaje de programación empleado para el desarrollo del ensamblado"), Values {"C#", "VisualBasic", "C"}, ValueMap{"0", "1", "2"}]
        uint8 Lenguaje;
        [Description ("Versión del ensamblado en producción"), MaxLen(24)]
        String Version;
```

```
        [Description ("Ruta del repositorio de CVS donde se encuentran los fuentes"), MaxLen(64)]
        String RutaCVS;
        [Required (true), Description ("Versión de .NET empleado para compilar el componente"), Values{"v1.0.375", "v1.1.4322", "v2.0.50727"}, ValueMap{"0", "1", "2"}]
        uint8 FrameWork;
        [Description("Indica si el ensamblado se encuentra en COM+")]
        Boolean ServicioComponentes = "false";
        [Description("Ubicación del ensamblado en COM+"), MaxLen(64)]
        String RutaServicioComponentes;
};
```

- Clase Banesco_ServicioWeb: Clase asociada a los dll's expuestos como servicio web.

```
class Banesco_ServicioWeb: Banesco_ElementoBase
{
        [Key, Description("Nombre del archivo asmx"), MaxLen(64)]
        String NombreAsmx;
        [Required, Description("Ruta del archivo asmx"), MaxLen(128)]
        String RutaAsmx;
        [Description ("Indica si el servicio esta expuesto para su consumo directo por algún cliente de la institución")]
        Boolean ServicioExpuesto = "false";
};
```

- Clase Banesco_Servicio: Clase representativa de los diferentes servicios/aplicaciones desarrollados por la gerencia de Nuevos Proyectos o Soporte y Mantenimiento.

```
class Banesco_Servicio: Banesco_ElementoBase
{
        [key, Description ("Nombre del servicio/aplicativo"), maxlen(64)]
        String NombreServicio;
        [Required (true), Description ("Nombre del usuario principal de la aplicación"), maxlen(64)]
        String Usuario;
```

[Description ("Indica si es un aplicativo de uso exclusivo de la organización")]

Boolean Interna = "false";

[Description ("Nombre del personal con mayor experticia con el aplicativo"), maxlen(128)]

String NombreExpertos;

};

- Clase Banesco_HTML: Clase que representa los documentos de hipertexto que forman parte de un servicio/aplicativo

class Banesco_HTML: Banesco_ElementoBase

{

[Key, Description ("Nombre del archivo/documento de hipertexto"), maxlen(32)]

String NombreArchivo;

[Key, Description("Ubicación física del archivo de hipertexto"), maxlen(64)]

String RutaArchivo;

};

- Clase Banesco_PaginaASP: Clase que representa los archivos de página activa de servidor.

class Banesco_PaginaASP: Banesco_ElementoBase

{

[Key, Description ("Nombre del archivo de página activa de servidor"), maxlen(32)]

String NombreArchivo;

[Key, Description("Ubicación física del archivo de página activa de servidor"), maxlen(64)]

String RutaArchivo;

};

- Clase Banesco_PaginaASPX: Clase que representa los archivos de página activa de servidor desarrollados con ASP.NET.

class Banesco_PaginaASPX: Banesco_ElementoBase

{

[Key, Description ("Nombre del archivo de página activa de servidor desarrollada con ASP.NET"), maxlen(32)]

String NombreArchivo;

[Key, Description("Ubicación física del archivo de página activa de servidor desarrollada con ASP.NET"), maxlen(64)]

String RutaArchivo;

};

- Clase Banesco_Script: Representa las librerías script empleadas por los documentos de hipertexto, páginas activas de servidor, etc.

class Banesco_Script

{

[Key, Description ("Nombre de la librería script"), maxlen(32)]

String NombreArchivo;

[Key, Description("Ubicación física de la librería script"), maxlen(64)]

String RutaArchivo;

};

- Clase Banesco_Configuracion: Clase que representa los archivos de configuración empleados por los ensamblados. Los valores que poseen afectan el funcionamiento de los ensamblados.

class Banesco_Configuracion: Banesco_ElementoBase

{

[Key, Description("Ubicación física del archivo"), MaxLen(64)]

String RutaArchivo;

[Key, Description("Nombre del archivo"), MaxLen(32)]

String NombreArchivo;

[Description("Tipo de archivo de configuración"), Required(true), values{"Web.Config", "Machine.Config", "Email.Config", "Xml", "Otro"}, ValueMap{"0", "1", "2", "3", "4"}]

uint8 TipoArchivo;

};

- Clase Banesco_Capa: Representa las diferentes capas donde se distribuyen los servicios.

class Banesco_Capa

{

[Key, Values{"Capa Web", "Capa Transaccional", "Capa Biztalk"}, ValueMap{"Web", "Trx", "Biz"}, Description("Nombre de la capa ")]

String NombreCapa;

[Required("true"), Description ("Direcciones IP de los distintos servidores que componen la capa"), ArrayType("Indexed")]

String IpServidores[];

};

- Clase Banesco_Problemas: Representa los eventos que afectan la normal operación de los servicios/aplicativos.

class Banesco_Problemas

{

[Key, Description("Título descriptivo del problema"), Maxlen(128)]

String TituloProblema;

[Required(true), Description("Descripción completa/detalles del problema"), Maxlen(1024)]

String Descripcion;

[Required(true), Description("Nombre del responsable en aplicar la solución"), Maxlen(64)]

String NombreResponsable;

[Values{"En Diagnostico", "En Implantacion", "Cerrado"}, ValueMap{"Diag", "Imp", "Cerr"}, Description("Estado en que se encuentra el problema")]

String Estado;

[Required(true), Description("Fecha de registro en el repositorio")]

Datetime FechaRegistrado;

[Description("Fecha de implantación de la solución al problema")]

datetime FechaCierre;

};

- Clase Banesco_Servidor: Representa los servidores que conforman la infraestructura donde residen las aplicaciones/servicios

class Banesco_Servidor

{

[Key, Description ("Nombre con el cual se identifica al servidor"), maxlen(32)]

String NombreServidor;

[Key, Description("Dirección IP del servidor"), maxlen(16)]

String IPServidor;

[Required (true), Description("Nombre del sistema operativo instalado en el equipo"), maxlen(64)]

String SistemaOperativo;

[Required (true), Description("Número de procesadores físicos del servidor")]

uint8 NroProcesadores;

[Description("Indica si se encuentra activo el programa que permite simular el doble de procesadores físicos")]

Boolean ExtraProcesador = "false";

[Required (true), Units("GigaBytes"), Description("Tamaño de la memoria RAM instalada en el servidor")]

uint8 Memoria;

[Description("Indica si la tarjeta de encriptamiento se encuentra instalada y activa")]

Boolean TarjetaIpSec = "false";

};

- Clase Banesco_WSDL: Representa los archivos de descripción del servicio web que permiten su consumo por parte de la páginas activas de servidor

class Banesco_WSDL: Banesco_ElementoBase

{

[key, Description ("Nombre del archivo de descripción del servicio web"), maxlen(32)]

String NombreArchivo;

[Key, Description ("Ruta del archivo de descripción del servicio web"), maxlen(64)]

String RutaArchivo;

};

- Asociación Banesco_ServicioWebEnsamblado: Asociación que permite identificar un ensamblado que es expuesto como servicio web.

class Banesco_ServicioWebEnsamblado

{

```
        [Description ("Servicio web"), Max(1), Min(1)]
        Banesco_ServicioWeb ref ServicioWeb;
        [Description ("Ensamblado expuesto como servicio web"), Key]
        Banesco_Ensamblado ref Ensamblado;
};
```

- Asociación Banesco_ServicioServicioWeb: Asociación para conocer que servicios web con consumidos por los diferentes servicios/aplicaciones desarrollados por la gerencia de Nuevos Proyectos o Soporte y Mantenimiento.

```
class Banesco_ServicioServicioWeb
{
        [Description ("Servicio/Aplicación"), Min(1), Key]
        Banesco_Servicio ref Servicio;
        [Description ("Servicio web consumido por Servicio/Aplicación"), Key]
        Banesco_ServicioWeb ref ServicioWeb;
};
```

- Asociación Banesco_EnsambladoReferencia: Asociación que representa las relaciones existentes entre los ensamblados.

```
class Banesco_EnsambladoReferencia
{
        [Key, Description ("Ensamblado referenciado")]
        Banesco_Ensamblado ref Referenciado = NULL;
        [Key, Description ("Ensamblado que referencia")]
        Banesco_Ensamblado ref Ensamblado = NULL;
};
```

- Asociación Banesco_CapaServicioWeb: Asociación de los servicios web expuestos en las diferentes capas de servicio.

```
class Banesco_CapaServicioWeb
{
        [Description ("Capa de servicio"), Key]
        Banesco_Capa ref Antecedente;
        [Description ("Servicio web publicado en la capa de servicio"), Key]
        Banesco_ServicioWeb ref Referenciado;
};
```

- Asociación Banesco_CapaEnsamblado: Asociación de los ensamblados localizados en las diferentes capas de servicio.

```
class Banesco_CapaEnsamblado
{
        [Description ("Capa de servicio"), Key]
        Banesco_Capa ref Capa;
        [Description ("Ensamblado ubicado en la capa de servicio"), Key]
        Banesco_Ensamblado ref Ensamblado;

        [Required (true), Description("Ubicación del ensamblado en los servidores pertenecientes a una capa de servicio"), MaxLen(128)]
        String RutaUbicacion;
};
```

- Asociación Banesco_CapaServidor: Asociación encargada de indicar los servidores que conforman las capas de infraestructura de servicios/aplicativos.

```
class Banesco_CapaServidor
{
        [Description ("Referencia a la capa"), Min(1), Max(1)]
        Banesco_Capa ref CapaServicio;
        [Description ("Referencia al servidor que conforma la capa"), Key]
        Banesco_Servidor ref Servidor;
};
```

- Asociación Banesco_HTMLScript: Representa las diferentes librerías scripts empleadas por los documentos de hipertexto.

```
class Banesco_HTMLScript
{
        [Description ("Documento de hipertexto"), Key, Min(1)]
        Banesco_HTML ref DocumentoHTML;
        [Description ("librerías Script"), Key]
        Banesco_Script ref Libreria;
};
```

- Asociación Banesco_ServicioHTML: Representa los documentos de hipertexto relacionados a un servicio.

```
class Banesco_ServicioHTML
{
        [Description ("Servicio/aplicativo"), Key, Min(1)]
        Banesco_Servicio ref Servicio;
        [Description ("Documento de hipertexto"), Key]
        Banesco_HTML ref DocumentoHTML;
};
```

- Asociación Banesco_PaginaASPScript: Representa las diferentes librerías scripts empleadas por las páginas activas de servidor.

```
class Banesco_PaginaASPScript
{
        [Description ("Archivo de página activa de servidor"), Key, Min(1)]
        Banesco_PaginaASP ref PaginaASP;
        [Description ("librerías Script"), Key]
        Banesco_Script ref Libreria;
};
```

- Asociación Banesco_ServicioPaginaASP: Representa las páginas activas de servidor relacionadas a un servicio.

```
class Banesco_ServicioPaginaASP
{
        [Description ("Servicio/aplicativo"), Key, Min(1)]
        Banesco_Servicio ref Servicio;
        [Description ("Página activa de servidor"), Key]
        Banesco_PaginaASP ref PaginaASP;
};
```

- Asociación: Banesco_ServicioPaginaASPX: Representa las páginas activas de servidor desarrolladas con ASP.NET relacionadas a un servicio.

```
class Banesco_ServicioPaginaASPX
{
        [Description ("Servicio/aplicativo"), Key, Min(1)]
        Banesco_Servicio ref Servicio;
```

```
        [Description ("Página activa de servidor"), Key]
        Banesco_PaginaASPX ref PaginaASPX;
};
```

- Asociación Banesco_EnsambladoPaginaASPX: Asociación que permite identificar el ensamblado que contiene a una página activa de servidor desarrollada con ASP.NET.

```
class Banesco_EnsambladoPaginaASPX
{
        [Description ("Referencia al ensamblado"), Min(1), Max(1), Key]
        Banesco_Ensamblado ref Ensamblado;
        [Description ("Referencia al archivo de página activa de servidor desarrollada con ASP.NET"), Key]
        Banesco_PaginaASPX ref PaginaASPX;
};
```

- Asociación Banesco_WSDLServicioWeb: Representa los archivos de descripción de servicio web relacionados a un servicio web.

```
class Banesco_WSDLServicioWeb
{
        [Description ("Servicio Web"), Key]
        Banesco_ServicioWeb ref ServicioWeb;
        [Description ("Archivo de descripción de servicio web"), Key]
        Banesco_WSDL ref ArchivoWSDL;
};
```

- Asociación Banesco_WSDLPaginaASP: Asociación que permite conocer los servicios web consumidos por una página activa de servidor

```
class Banesco_WSDLPaginaASP
{
        [Description ("Archivo de descripción de servicio web"), Key]
        Banesco_WSDL ref ArchivoWSDL;
        [Description ("Página activa de servidor"), Key]
        Banesco_PaginaASP ref PaginaASP;
};
```

- Asociación Banesco_EnsambladoConfiguracion: Asociación que permite conocer los archivos de configuración empleados por distintos ensamblados.

```
class Banesco_EnsambladoConfiguracion
{
        [Description ("Ensamblado que hace uso de los valores presentes en el archivo de configuración"), Min(1), Key]
        Banesco_Ensamblado ref Ensamblado;
        [Description ("Archivo de configuración"), Key]
        Banesco_Configuracion ref Configuracion;
};
```

- Asociación Banesco_ServicioProblemas: Representación de los eventos que afectan el normal funcionamiento de los servicios/aplicativos.

```
class Banesco_ServicioProblemas
{
        [Description ("Servicio/aplicativo"), Min(1), Key]
        Banesco_Servicio ref Servicio;
        [Description ("Evento que afecta la operación normal del servicio/aplicativo"), Key]
        Banesco_Problemas ref Problema;
};
```

EVALUACIÓN DEL PROYECTO.

El objetivo principal del proyecto orientado a diseñar, desarrollar e implantar un repositorio de información donde se muestran las relaciones existentes entre distintos elementos de configuración fue logrado en su totalidad, toda vez que realizado el diseño de datos de la CMDB, desarrollado e implantado, se comprueba que refleja la manera en que los diferentes EC interactúan entre si, dando cabida al registro de información útil al momento de evaluar el alcance de un cambio en algún EC, o bien al momento de evaluar el esfuerzo requerido para el desarrollo de un nuevo servicio o aplicativo.

Una de las limitaciones establecidas al inicio del proyecto tiene que ver con la selección de una herramienta compatible con el sistema operativo instalado en los equipos de trabajo, a la cual se pudiera exportar el modelo de datos del repositorio, y facilitar el uso de la CMDB por parte de los usuarios finales.

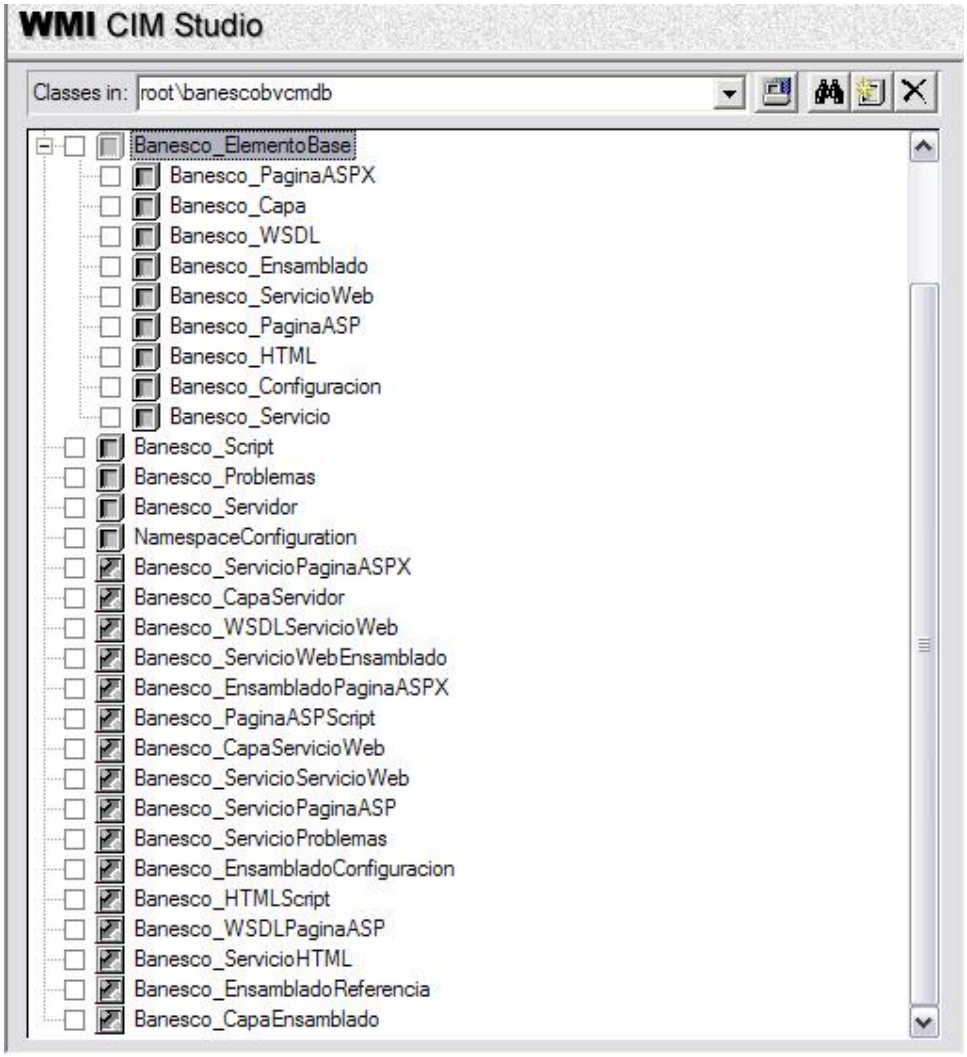
Para este fin fue seleccionada la Instrumentación Administrativa de Windows (WMI por sus siglas en inglés), API del sistema operativo Windows que permite controlar y administrar a los equipos de una red. Mediante el empleo de CIM, el WMI permite la definición de clases, atributos y asociaciones que conforman el modelo de datos. Así mismo permite crear fácilmente instancias de las clases que representan los diferentes EC que forman parte del repositorio. WMI no requiere compra de ninguna licencia de software para su uso.

Un aspecto clave que apoya el éxito del proyecto es la capacidad que tiene el WMI para mostrar con el uso de gráficos las relaciones existentes en los EC y los roles que tiene cada uno de ellos. De igual manera la facilidad que tiene para permitir a los usuarios consultar y actualizar información de cada uno de los elementos de configuración hace de esta herramienta una buena escogencia de cara a los usuarios.

El uso de un estándar para el diseño del modelo de datos del repositorio resulta beneficioso en la evolución que se realice del proyecto. Gracias a ello las labores de desarrollo de una interfaz personalizada no necesitarán ningún tipo de adaptación especial del modelo, en virtud que el lenguaje empleado en su definición es perfectamente entendido por distintas tecnologías de desarrollo de software.

En la figura 14 de observa como queda expuesto el modelo de clases del repositorio en el WMI.

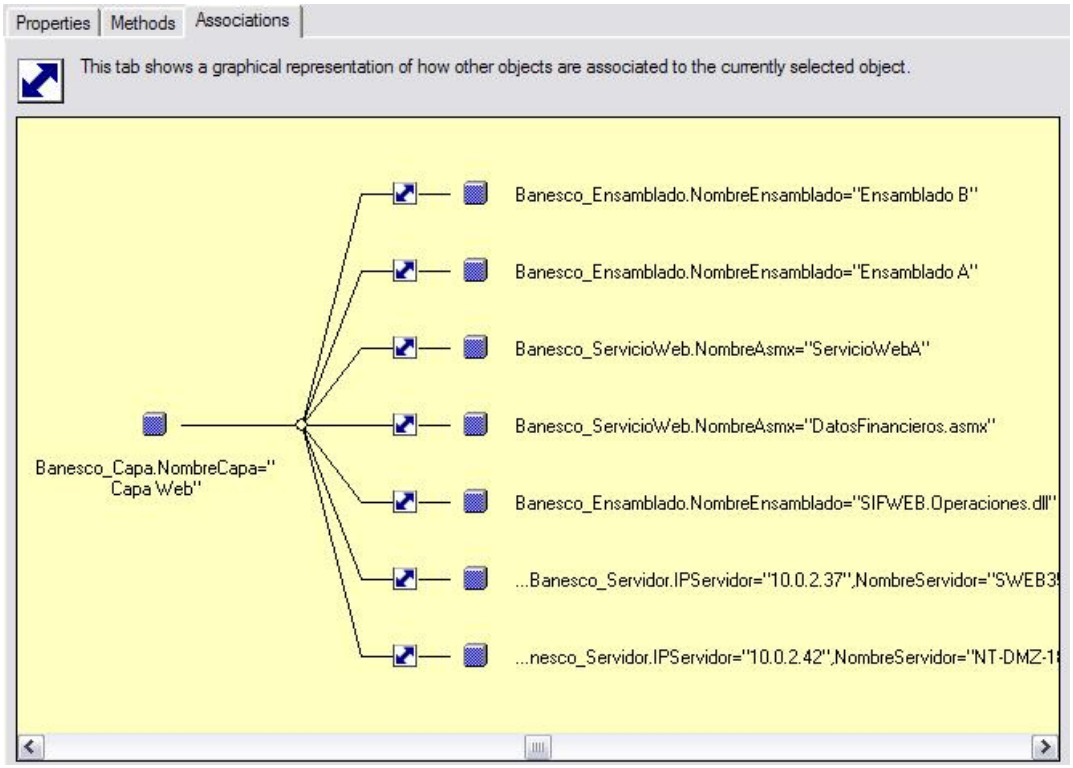
Figura 14: Representación del modelo de clases en WMI.



Nótese en la figura anterior que en WMI quedan expuestas tanto las clases como las asociaciones que las relacionan.

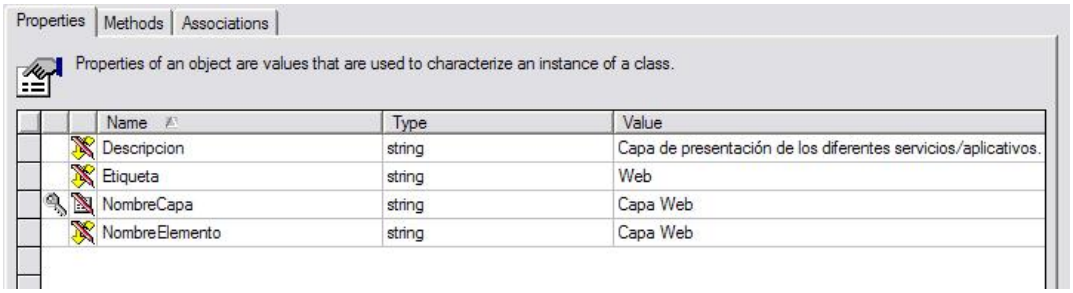
En la figura 15 se expone un ejemplo de cómo gráficamente, WMI despliega las relaciones existentes entre un conjunto de EC. Es de resaltar que al mostrar cada una de las instancias de las clases, se muestra su clave de identificación única.

Figura 15: Representación gráfica en WMI de las relaciones entre EC.



En la figura 16 se observa como WMI muestra información relacionada con las instancias de las clases definidas en el repositorio (EC).

Figura 16: Despliegue de atributos de un EC en WMI.



CONCLUSIONES Y RECOMENDACIONES.

Toda vez que el repositorio de configuración está diseñado, desarrollado y puesto a disposición para su uso por parte del personal de la Gerencia de Banca Virtual, se ratifica la importancia de contar con una herramienta de apoyo en las labores de gestión de configuración, permitiendo evaluar más eficientemente el impacto en los servicios que tiene el constante mantenimiento (preventivo, perfectivo y correctivo) de los componentes de software.

Sin embargo, existe la posibilidad que todo el esfuerzo invertido en lograr el desarrollo del repositorio se pierda si este no es frecuentemente actualizado, y como consecuencia, el repositorio puede reflejar una imagen incorrecta del estado de los elementos de configuración almacenados en él.

Esta situación puede acarrear peores consecuencias si se compara con un escenario donde no se cuente con el repositorio de configuración. Por ejemplo, al surgir la necesidad de actualizar un EC, es posible que la CMDB indica de manera equivocada otros elementos o servicios que se tiene que modificar para que el cambio sea efectivo. Esto lleva a una situación donde además de modificar indebidamente un elemento (con el riesgo de afectar su funcionamiento) se pierde un esfuerzo importante en tiempo y dinero en modificaciones que no son necesarias.

En vista del escenario antes descrito, se recomienda tener especial cuidado en asegurar que la información contenida sea adecuadamente actualizada, de manera que refleje siempre los últimos cambios realizados en todos los EC que la conforman. Adicional a esto, se recomienda promover su uso, asegurando así que los usuarios propongan mejoras que hacen con el tiempo que el repositorio se convierta en una fuente más confiable y completa de información.

REFERENCIAS BIBLIOGRÁFICAS.

BMC Software (2005). What do you need from a configuration management database (CMDB)?. Recuperado en Junio 4, 2006, de http://www.bmc.com/USA/Corporate/BSM/attachments/BMC_CMDB_wp_en.pdf

BMC Software (2006). *Configuration management database (CMDB). Cornerstone of ITIL – based integrated service support*. Recuperado en noviembre 26, 2006, de http://documents.bmc.com/products/documents/66/93/56693/56693.pdf

Combalia, P. (2005). *Conceptos generales de ITIL Descripción de los conceptos básicos sobre ITIL, alcance, beneficios y otros temas relacionados*. Recuperado en Diciembre 28, 2005, de http://www.itsmf-argentina.com.ar/IMG/pdf/ConceptosGralesITIL_v_01_2_.pdf

Combalia, P. (2005). *Visión general sobre soporte y prestación de servicios IT*. Recuperado En Diciembre 28, 2005, de http://www.itsmf-argentina.com.ar/IMG/pdf/ConceptosGralesSoporteyPrestacionServicios_v_01_2_.pdf

Cossío~Ortiz, S.G., y Palomino~Martínez, D.F. (2005). *ITIL: servicios de tecnologías de información*. Recuperado en Diciembre 28, 2005, de la Univeridad Nacional Autónoma de México Web site http://www.enterate.unam.mx/Articulos/2005/noviem/itil.htm

Del Rosario, Z. y Peñaloza, S. (2003). *Guía para la elaboración formal de reportes de investigación*. Caracas: Publicaciones UCAB.

Directory of ITIL, ITSM & security services & software. (2005). *ITIL change management*. Recuperado en Diciembre 30, 2005, de http://www.itil-itsm-world.com/itil-3.htm

Distributed Management Task Force (2005). *Common Information Model (CIM) Infrastructure Specification*. Recuperado en Agosto 15, 2006, de http://www.dmtf.org/standards/published_documents/DSP0004V2.3_final.pdf

Duthie, G.A (2002). *Aprenda Ya Microsoft ASP.NET*. España: McGraw-Hill.

Enterprise Management Associates (2005). *Planning for CMDB Design and Adoption: An Industry Colloquium*. Recuperado en Enero 3, 2006, de http://www.nlayers.com/pdf/CMDB_Design_Adoption_by_EMA.pdf

Foro HelpDesk. (2006). *ITIL: Administración de la configuración – Modelo de datos de la administración de servicios de IT*. Recuperado en Agosto 6, 2006, de http://www.foro-helpdesk.com/index.php?publicaciones_white_papers=1&publicacion_id=226

Foro HelpDesk. (2006). *ITIL Implementación de la administración de la configuración*. Recuperado en Agosto 6, de 2006, de http://www.foro-helpdesk.com/index.php?publicaciones_white_papers=1&publicacion_id=285

Information technology infrastructure library. (2005). Recuperado en Enero 3, 2006, de http://en.wikipedia.org/wiki/Information_Technology_Infrastructure_Library

Hobbs, C (2004). *A practical approach to WBEM/CIM management*. USA: Auerbach

Landeau, R (2005). *Manual de investigación. Trabajos de grado*. Caracas: Ediciones Torán, C.A

Mueller, D (2005). *Taking de configuration management database to the next level: The federated data model*. Recuperado en enero 3, 2006, de http://www.computerworld.com/managementtopics/management/story/0,10801,101081,00.html?SKC=management-101081

Salischiker, D (2005). *Glosario de términos ITIL Diccionario que contiene los términos mas utilizados en el marco de trabajo de ITIL*. Recuperado en Diciembre 28, 2005, de http://www.itsmf-argentina.com.ar/IMG/pdf/ItilGlosario_v.01.pdf

ANEXO A
Notación Formal para la Sintaxis del Formato de Objeto Administrado

Este anexo contiene una notación formal de la sintaxis del formato de objeto administrado. Se deben tomar en consideración los siguientes puntos:

1. Una lista de propiedades vacía equivale a “*”.
2. Todas las palabras clave son insensibles al uso de letras mayúsculas y minúsculas.
3. Un valor de tipo cadena de caracteres puede contener comillas (“) precedidos siempre por un carácter backslash (\).
4. El tipo IDENTIFIER es empleado para nombrar clases, propiedades, calificadores, métodos y espacios de nombre (namespace).

```
mofSpecification      =  *mofProduction

mofProduction         =  compilerDirective    |
                         classDeclaration     |
                         assocDeclaration     |
                         indicDeclaration     |
                         qualifierDeclaration |
                         instanceDeclaration

compilerDirective     =  PRAGMA pragmaName  "(" pragmaParameter ")"

pragmaName            =  IDENTIFIER

pragmaParameter       =  stringValue

classDeclaration      =  [ qualifierList ]
                         CLASS className  [ superClass ]
                         "{" *classFeature "}" ";"

assocDeclaration      =  "[" ASSOCIATION *( "," qualifier ) "]"
                         CLASS className  [ superClass ]
                         "{" *associationFeature "}" ";"

                         // Context:
                         // The remaining qualifier list must not include
                         // theASSOCIATION qualifier again. If the
                         // association has no super association, then at
                         // least two references must be specified! The
                         // ASSOCIATION qualifier may be omitted in
                         // sub-associations.

indicDeclaration      =  "[" INDICATION *( "," qualifier ) "]"
                         CLASS className  [ superClass ]
                         "{" *classFeature "}" ";"

className             =  schemaName "_" IDENTIFIER   // NO whitespace !

                         // Context:
                         // Schema name must not include "_" !

alias                 =  AS aliasIdentifer

aliasIdentifer        =  "$" IDENTIFIER   // NO whitespace !

superClass            =  ":" className

classFeature          =  propertyDeclaration | methodDeclaration
```

```
associationFeature     =  classFeature | referenceDeclaration

qualifierList          =  "[" qualifier *( "," qualifier ) "]"

qualifier              =  qualifierName [ qualifierParameter ] [ ":" 1*flavor ]

qualifierParameter     =  "(" constantValue ")" | arrayInitializer

flavor                 =  ENABLEOVERRIDE | DISABLEOVERRIDE | RESTRICTED |
                          TOSUBCLASS | TRANSLATABLE

propertyDeclaration    =  [ qualifierList ] dataType propertyName
                          [ array ] [ defaultValue ] ";"

referenceDeclaration   =  [ qualifierList ] objectRef referenceName


                          [ defaultValue ] ";"

methodDeclaration      =  [ qualifierList ] dataType methodName
                          "(" [ parameterList ] ")" ";"

propertyName           =  IDENTIFIER

referenceName          =  IDENTIFIER

methodName             =  IDENTIFIER

dataType               =  DT_UINT8 | DT_SINT8 | DT_UINT16 | DT_SINT16 |
                          DT_UINT32 | DT_SINT32 | DT_UINT64 | DT_SINT64 |
                          DT_REAL32 | DT_REAL64 | DT_CHAR16 |
                          DT_STR | DT_BOOL | DT_DATETIME

objectRef              =  className REF

parameterList          =  parameter *( "," parameter )

parameter              =  [ qualifierList ] (dataType|objectRef) parameterName
                          [ array ]
parameterName          =  IDENTIFIER

array                  =  "[" [positiveDecimalValue] "]"

positiveDecimalValue   =  positiveDecimalDigit *decimalDigit

defaultValue           =  "=" initializer

initializer            =  ConstantValue | arrayInitializer | referenceInitializer

arrayInitializer       =  "{" constantValue*( "," constantValue)"}"

constantValue          =  integerValue | realValue | charValue | stringValue |
                          booleanValue | nullValue

integerValue           =  binaryValue | octalValue | decimalValue | hexValue

referenceInitializer   =  objectHandle | aliasIdentifier

objectHandle           =  stringValue
                          // the(unescaped)contents of which must form an
                          // objectName; see examples

objectName                [ namespacePath ":" ]  modelPath

namespacePath             [ namespaceType "://" ] namespaceHandle

namespaceType             One or more UCS-2 characters NOT including the sequence
                          "://"
```

```
namespaceHandle        =  One or more UCS-2 character, possibly including ":"
                          // Note that modelPath may also contain ":" characters
                          // within quotes; some care is required to parse
                          // objectNames.

modelPath              =  className "." keyValuePairList
                          // Note: className alone represents a path to a class,
                          // rather than an instance

keyValuePairList       =  keyValuePair *( "," keyValuePair )

keyValuePair           =  ( propertyName "=" constantValue ) | ( referenceName "="
                          objectHandle )

qualifierDeclaration   =  QUALIFIER qualifierName qualifierType scope
                          [ defaultFlavor ] ";"

qualifierName          =  IDENTIFIER

qualifierType          =  ":" dataType [ array ] [ defaultValue ]

scope                  =  "," SCOPE "(" metaElement *( "," metaElement ) ")"

metaElement            =  CLASS | ASSOCIATION | INDICATION | QUALIFIER
                          PROPERTY | REFERENCE | METHOD | PARAMETER | ANY

defaultFlavor          =  "," FLAVOR "(" flavor *( "," flavor ) ")"

instanceDeclaration    =  [ qualifierList ] INSTANCE OF className [ alias ]
                          "{" 1*valueInitializer "}" ";"

valueInitializer       =  [ qualifierList ]
                          ( propertyName | referenceName ) "=" initializer ";"
```

Las siguientes definiciones no permiten espacios en blanco entre palabras.

```
schemaName             =  IDENTIFIER
                          // Context:
                          // Schema name must not include "_" !
fileName               =  stringValue

binaryValue            =  [ "+" | "-" ] 1*binaryDigit ( "b" | "B" )

binaryDigit            =  "0" | "1"

octalValue             =  [ "+" | "-" ] "0" 1*octalDigit

octalDigit             =  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7"

decimalValue           =  [ "+" | "-" ] ( positiveDecimalDigit *decimalDigit | "0" )

decimalDigit           =  "0" | positiveDecimalDigit

positiveDecimalDigit   =  "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

hexValue               =  [ "+" | "-" ] ( "0x" | "0X" ) 1*hexDigit

hexDigit               =  decimalDigit | "a" | "A" | "b" | "B" | "c" | "C" |
                          "d" | "D" | "e" | "E" | "f" | "F"

realValue              =  [ "+" | "-" ] *decimalDigit "." 1*decimalDigit
                          [ ( "e" | "E" ) [ "+" | "-" ] 1*decimalDigit ]

charValue              =  // any single-quoted Unicode-character, except
                          // single quotes

stringValue            =  1*( """ *stringChar """ )

stringChar             =  "\" """ | // encoding for double-quote
                          "\" "\" | // encoding for backslash
                          any UCS-2 character but """ or "\"

booleanValue           =  TRUE | FALSE

nullValue              =  NULL
```

Las siguientes palabras clave no son sensibles al uso de letras mayúsculas y minúsculas.

```
ANY                     =  "any"
AS                      =  "as"
ASSOCIATION             =  "association"
CLASS                   =  "class"
DISABLEOVERRIDE         =  "disableOverride"
DT_BOOL                 =  "boolean"
DT_CHAR16               =  "char16"
DT_DATETIME             =  "datetime"
DT_REAL32               =  "real32"
DT_REAL64               =  "real64"
DT_SINT16               =  "sint16"
DT_SINT32               =  "sint32"
DT_SINT64               =  "sint64"
DT_SINT8                =  "sint8"
DT_STR                  =  "string"
DT_UINT16               =  "uint16"
DT_UINT32               =  "uint32"
DT_UINT64               =  "uint64"
DT_UINT8                =  "uint8"
ENABLEOVERRIDE          =  "enableoverride"
FALSE                   =  "false"
FLAVOR                  =  "flavor"
INDICATION              =  "indication"
INSTANCE                =  "instance"
METHOD                  =  "method"
NULL                    =  "null"
OF                      =  "of"
PARAMETER               =  "parameter"
PRAGMA                  =  "#pragma"
PROPERTY                =  "property"
QUALIFIER               =  "qualifier"
REF                     =  "ref"
REFERENCE               =  "reference"
RESTRICTED              =  "restricted"
SCHEMA                  =  "schema"
SCOPE                   =  "scope"
TOSUBCLASS              =  "tosubclass"
TRANSLATABLE            =  "translatable"
TRUE                    =  "true"
```

ANEXO B
Tipos de Datos Intrínsecos Definidos en el Formato de Objeto Administrado

Tabla B1: *Tipos de datos intrínsecos definidos en el Formato de Objeto Administrado y su interpretación* (tomada de Distributed Management Task Force, 2005).

Tipo de dato intrínseco	**Interpretación**
uint8	Entero de 8 bit sin signo
sint8	Entero de 8 bit con signo
uint16	Entero de 16 bit sin signo
sint16	Entero de 16 bit con signo
uint32	Entero de 32 bit sin signo
sint32	Entero de 16 bit con signo
uint64	Entero de 64 bit sin signo
sint64	Entero de 64 bit con signo
string	Cadena de caracteres UCS-2
boolean	Boleano
real32	Punto flotante de 4 byte IEEE
real64	Punto flotante de 8 byte IEEE
datetime	Una cadena de caracteres conteniendo fecha/hora
<nombre de clase> ref	Referencia fuertemente tipeada
char16	Carácter UCS-2 de 16 bit