Implementación de un sistema de comunicación redundante utilizando el middleware TAO

Yadiel Martinez González
Universidad de las Ciencias Informáticas, La Habana, Cuba, ymglez@uci.cu

Yanelys del Rosario Lalcebo
Universidad de las Ciencias Informáticas, La Habana, Cuba, ydelrosario@uci.cu

Yolier Galán Tassé
Universidad de las Ciencias Informáticas, La Habana, Cuba, ytasse@uci.cu

Yenier Figueroa Machado
Universidad de las Ciencias Informáticas, La Habana, Cuba, yfigueroa@uci.cu

ABSTRACT

Achieving a high level on availability of SCADA systems is a prime requirement that contributes to it´s reliability. In order to do that is necessary to have a set of replicated components which must be coordinated and synchronized, in a way that do not exist any lack of substantial information given a specific failure, what is known as redundancy.
SCADA systems are based on obtaining and processing information from scattered industrial processes and to act on them remotely through a middleware. This document describes how to implement a redundancy mechanism with TAO middleware over a SCADA system.
There were taken into account several elements to develop this work, such as the tools, technologies and methodology. Finally, some results obtained from executed tests are shown, which demonstrate the high level of availability that provides the referred mechanism to the SCADA systems.

Keywords: communication, redundancy, SCADA system

RESUMEN

Alcanzar un alto grado de disponibilidad en los sistemas SCADA[1] resulta un requisito fundamental que contribuye a la confiabilidad del mismo. Para ello, es preciso contar con un conjunto de componentes replicados que deben coordinarse y actuar sincronizadamente, de forma que no se produzca una pérdida fundamental de información ante un determinado fallo, lo que se denomina como redundancia.
Los sistemas SCADA se basan en obtener y procesar información de procesos industriales dispersos y de actuar en forma remota sobre los mismos a través de un middleware[2]. El presente documento expone cómo implementar el mecanismo de redundancia en un SCADA con el middleware TAO[3].
Para desarrollar este trabajo se tuvo en cuenta varios elementos como son las herramientas, tecnologías y los pasos a seguir. Por último se muestran algunos resultados obtenidos de las pruebas realizadas, los cuales demuestran el alto grado de disponibilidad que ofrece dicho mecanismo a los sistemas SCADA.

Palabras claves: comunicación, redundancia, sistemas SCADA

[1] SCADA: (*Supervisoy Control And Data Adquisition*)
[2] Middleware: Software de infraestructura que reside entre las aplicaciones y el sistema operativo, redes y hardware subyacentes, intentando brindar una plataforma más apropiada para el desarrollo y ejecución de los sistemas distribuidos. [1]
[3] TAO: The ACE (*Adaptive Communication Environment*) ORB (*Object Request Broker*)

1. INTRODUCCIÓN

El Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas (UCI), cuya misión es el desarrollo de productos y servicios de automatización industrial, está compuesto por varias líneas de trabajo que se especializan en los diferentes componentes de un sistema SCADA. Una de las líneas fundamentales es la de Comunicaciones, que permite el funcionamiento del sistema como un todo a través del intercambio de información entre todos los componentes distribuidos con que cuentan los SCADA desarrollados. Para ello esta línea utiliza el middleware TAO, el cual es una implementación del estándar CORBA[4] que permite la invocación de métodos remotos en sistemas distribuidos.

El medio de comunicación en un sistema SCADA debe permitir el intercambio de información de forma coordinada, confiable y segura, garantizando la efectividad y confiabilidad conque debe contar el sistema. Lo anteriormente planteado se traduce en que las comunicaciones en este tipo de sistemas, deben garantizar el mayor grado de disponibilidad posible, permitiendo que el sistema pueda funcionar, aun cuando se produzca algún fallo en alguno de sus componentes. Al originarse un fallo en un sistema de control industrial, se pueden producir resultados incorrectos, o incluso dejar de brindarse un servicio determinado, por lo que resulta de vital importancia evitar cualquier incidente de este tipo.

La redundancia es la clave para conseguir un sistema de alta disponibilidad, haciendo que el sistema sea resistente a fallos y además permitiendo su mantenimiento y actualización. Actualmente el mecanismo de redundancia que provee el middleware TAO para los sistemas distribuidos, depende en gran medida de servicios externos a los cuales todos los servidores redundantes deben tener acceso.

Este mecanismo de redundancia, no resulta del todo eficiente cuando se quiere garantizar un alto grado de disponibilidad. Teniendo en cuenta que se pueden producir cualquier número de fallos, ya sean eléctricos, de red, de hardware o de software, un servicio externo puede verse afectado por alguno de estos y producir una parada inesperada del sistema. Por tanto, en busca de un mejor rendimiento, el objetivo de este trabajo es la implementación de un mecanismo de comunicación redundante con el middleware TAO, de manera que sea integrado al propio sistema y no dependa de servicios externos.

2. CONTENIDO

Los sistemas informáticos que desarrollan tareas críticas en empresas e industrias como los SCADA, requieren de una alta disponibilidad que les permita estar funcionando las 24 horas del día, los 365 días del año para evitar que ocurran fallos que afecten el funcionamiento normal del sistema. Los fallos siempre van a ocurrir y más en los sistemas actuales que cuentan con muchos componentes necesarios para su funcionamiento. Sin embargo existen técnicas que ayudan a minimizar la ocurrencia de fallos en un sistema y una de estas es la de tener sistemas redundantes.

Los sistemas redundantes están compuestos por dos elementos fundamentales, una aplicación primaria que es la que se encuentra activa para garantizar el funcionamiento del sistema y otra u otras secundarias que se encuentran en Stand-by[5] con igual configuración, para activarse inmediatamente en el momento que falle la aplicación primaria.

Con el fin de garantizar lo antes planteado, en esta sección se describirán las herramientas y los pasos a seguir para desarrollar el sistema de redundancia con el middleware TAO y también se mostrarán los resultados obtenidos de algunas de las pruebas realizadas al sistema.

2.1 TECNOLOGÍA MIDDLEWARE TAO

Como se expone en la introducción del trabajo, la tecnología middleware utilizada es TAO. Este está concebido para el desarrollo de sistemas en tiempo real debido a la eficiente implementación que realiza del estándar *Real*

[4] CORBA: Estándar que establece una plataforma de desarrollo de sistemas distribuidos, facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos. [2]

[5] Stand-by: Aplicaciones que se encuentran inactivas, en espera de recibir órdenes para activarse.

Time CORBA, soportando la mayoría de sus características. TAO está disponible libremente y es de código abierto. Proporciona eficiencia, previsibilidad, escalabilidad y calidad de servicios mediante el uso de patrones estándar. Además ofrece una portabilidad extensa para un gran número de compiladores y sistemas operativos.

Entre los componentes básicos de TAO se encuentran:

- **Cliente:** Esta entidad realiza tareas de aplicación obteniendo referencias de objetos a los servidores e invocando operaciones de los servidores. El cliente no tiene conocimiento de cómo se implementa el objeto CORBA. Las operaciones que el cliente puede invocar son definidas en la interfaz de objetos como se expresa en el Lenguaje de Definición de Interfaces (IDL) de la OMG. [3]
- **Servidor:** Este componente implementa las operaciones definidas en el Lenguaje de Definición de Interfaces de la OMG. [3]
- **Compilador IDL:** Genera los *stubs* y *skeletons*, necesarios para las llamadas remotas a métodos. [3]
- ***Portable Object Adapter* (POA):** Es responsable de crear referencias a objetos, la activación de los objetos, y el envío de las solicitudes formuladas en los objetos a sus respectivos servidores. [4]
- **ORB *Core*:** El ORB *Core* entrega las solicitudes de clientes al adaptador de objetos y devuelve respuestas a los clientes. [3]
- **ACE:** TAO se implementa sobre ACE. Este es el middleware de nivel inferior que implementa los patrones básicos de concurrencia y de distribución para el software de comunicación. [3] ACE se compone de un gran conjunto de utilidades de programación en lenguaje C++ disponibles tanto en Linux como en Windows. Es un framework de libre distribución y código abierto.

2.2 PROPUESTA DE SOLUCIÓN PARA EL SISTEMA DE REDUNDANCIA UTILIZANDO TAO

A continuación se explican los pasos a seguir para darle solución al sistema de comunicación redundante utilizando TAO. Para ello se describen las posibilidades que ofrece la tecnología y las modificaciones realizadas para dar cumplimiento al objetivo propuesto.

TAO implementa la mayoría de los servicios de CORBA, entre ellos el *Naming Service*, el cual asigna nombres a referencias de objetos. El servicio de nombres es el principal mecanismo por el cual la mayoría de los clientes pueden localizar los objetos que se vayan a utilizar [3]. Es importante destacar además, que la misma referencia de objeto se puede almacenar varias veces con nombres diferentes, pero cada nombre identifica exactamente una referencia.

Para obtener un sistema de comunicación redundante con TAO es necesario garantizar primero que todo, la redundancia del servicio de nombres anteriormente mencionado, teniendo en cuenta que este es el servicio que permite la localización entre clientes y servidores. La redundancia que propone TAO para el *Naming Service* está basada en un sistema de ficheros que almacena las referencias a objetos por cada contexto de nombre y al cual deben tener acceso tanto el servicio primario como el secundario. Por tanto, para evitar el trabajo engorroso que conlleva el depender de un sistema de ficheros, y tratando de obtener un mejor rendimiento, se propone para este trabajo que el intercambio de información entre los *Naming Service* activos se realice en tiempo de ejecución y sin necesidad de acceder a un sistema de ficheros.

El segundo paso a seguir es lograr que los servidores sean objetos persistentes. La persistencia en las conexiones garantiza que siempre que haya una interrupción, la conexión entre los elementos (clientes y servidores) sea automáticamente establecida tan pronto como se recupere el servicio. Esto evita que se tengan que reiniciar los componentes para reconocerse nuevamente. TAO permite crear objetos persistentes a través de las políticas *LifespanPolicy* con la opción *PERSISTENT* e *IdAssignmentPolicy* con la opción *USER_ID*. El uso de estas políticas permite que las referencias persistentes se mantengan en el mismo objeto si el servidor es apagado y reiniciado, es decir, que el servidor sigue utilizando la misma dirección y el mismo puerto por donde se había iniciado anteriormente en caso de ser reiniciado.

Por último, es necesario poder tener varias referencias de objetos servidores con el mismo nombre registrados en el *Naming Service*, característica que no es permitida por dicho servicio, pero que es precisa para permitir la réplica de los servidores, garantizando así la redundancia de los mismos. FT_CORBA es una especificación de tolerancia a fallas compuesta por varios servicios que permite la redundancia para TAO, entre ellos se encuentran: *Replication Manager*, *Fault Notifier*, *Fault Detector Factories* y *Replica Factories*. Para poder lograr la redundancia entonces, es necesario iniciar cada uno de estos servicios, característica, que teniendo en cuenta los objetivos del trabajo, no es factible, porque si llega a fallar uno de ellos ya se perdería la redundancia.

Ante esta situación, se estudió la forma de poder replicar los servidores y obtener el mecanismo de la redundancia sin necesidad de depender de varios servicios. Para lograr el objetivo, se utilizó una de las bibliotecas de TAO: la *IORManipulation* que es utilizada también por el servicio *Replication Manager* antes mencionado. El uso de esta biblioteca en la implementación de los servidores, permite la creación y administración de los IOGRs (*Interoperable Object Group References*). Su objetivo es establecer un servidor principal y un conjunto de réplicas del servidor, y ver si el cliente en su primera invocación desea ponerse en contacto con el principal. Cuando el servidor principal falla, entonces el cliente establece la conexión con el secundario en la próxima invocación.

A continuación se muestra gráficamente la propuesta de solución. En la figura 1, se representa el comportamiento que debe tener el sistema de redundancia a través de la federación de *Naming Service* que propone TAO. Este se basa en un modo *full-connected* donde todos los clientes tienen acceso a todas las aplicaciones desde cualquier localización del *Naming Service*. La diferencia en este caso, es que en lugar de utilizar un sistema de ficheros para almacenar la información referente a los clientes y servidores que tienen conectados, se la trasmiten en tiempo de ejecución actualizando el espacio de memoria de cada uno, es decir, al ser modificado el espacio de nombres de un *Naming Service*, este actualiza al resto de los *Naming Service* redundantes pasándole las nuevas modificaciones ocurridas.

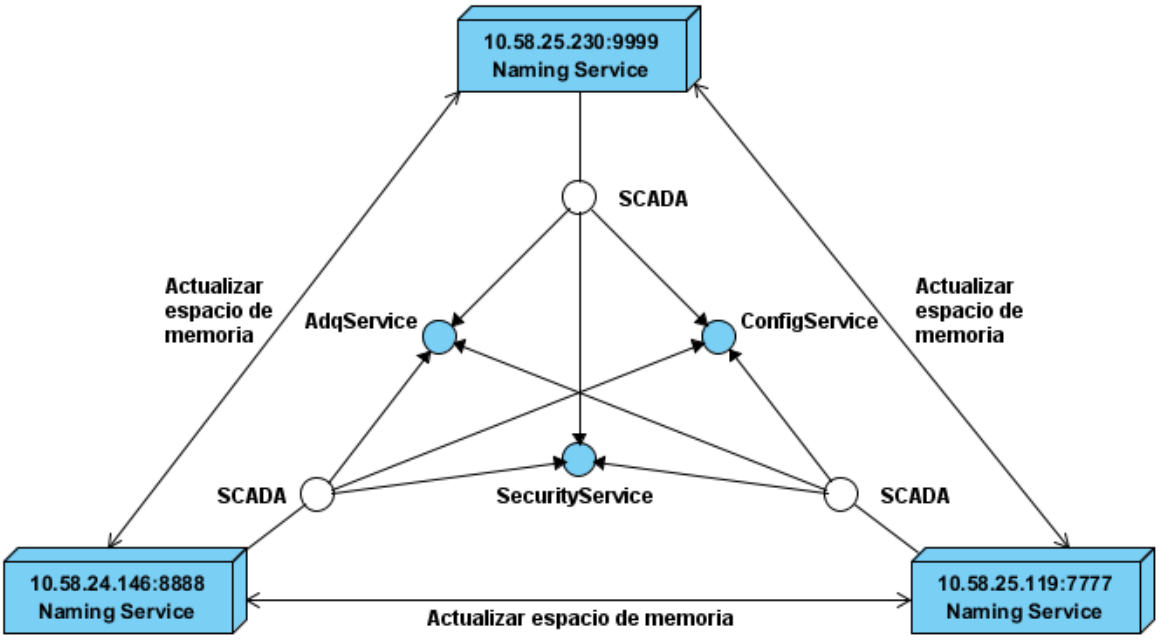

Figura 1: Comportamiento *full-connected* del *Naming Service*.

En la figura 2, se representa el funcionamiento de la redundancia entre los servidores, la cual se obtiene a través de la manipulación de los IOGRs como se explicó anteriormente. Estos servidores intercambian su estado constantemente, es decir, la última información que envió, los clientes que puedan tener conectados, de manera que no existan pérdidas de datos en la comunicación.

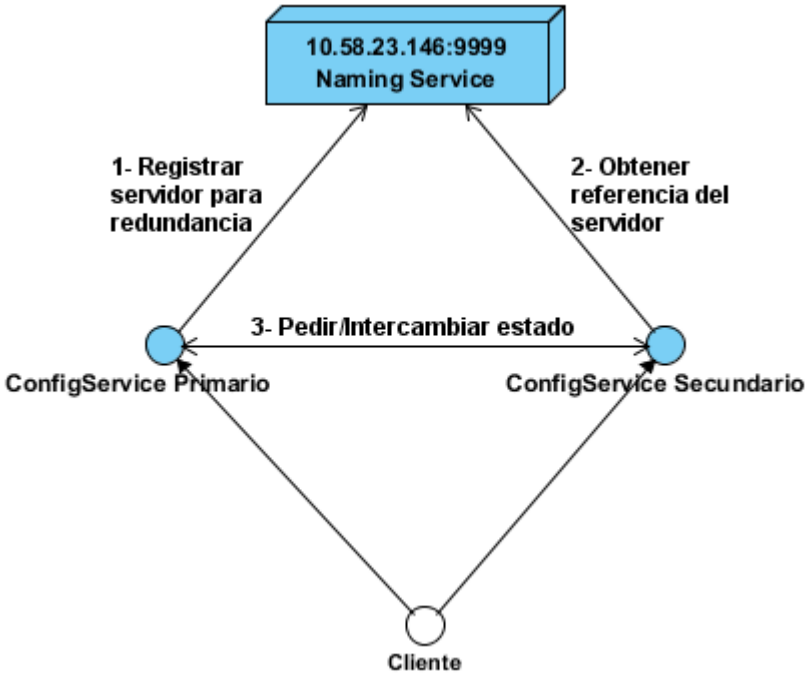

Figura 2: Redundancia de los servidores.

Encapsulando todo el funcionamiento de lo que es el sistema de redundancia implementado, en la siguiente figura se muestra el diagrama de paquetes que agrupa todas las funcionalidades del *Naming Service* en la biblioteca *lib_fed_naming*, mientras que la redundancia de los servidores se agrupa en la biblioteca *lib_redundance_manager*.

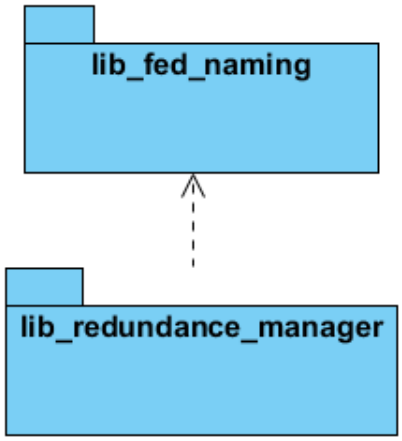

Figura 3: Diagrama de paquetes del sistema de redundancia.

2.3 PATRONES DE DISEÑO UTILIZADOS

El uso de patrones contribuye a reutilizar diseño, identificando aspectos claves de la estructura de un diseño que puede ser aplicado en una gran cantidad de situaciones. La reutilización del diseño provee numerosas ventajas: reduce los esfuerzos de desarrollo y mantenimiento, mejora la seguridad, eficiencia y consistencia de los diseños. Además aumenta la flexibilidad, modularidad y extensibilidad, factores internos e íntimamente relacionados con la calidad percibida por el usuario.

***Adapter*:** Este patrón consiste en tener una clase adaptadora que contiene la misma interfaz que la clase adaptada permitiendo añadir nuevos comportamientos. En este caso se utiliza para la clase *NamingContext* que es implementada originalmente por TAO pero para añadir los nuevos comportamientos en cuanto a redundancia se

refiere, es necesario crear un adaptador de esta clase. A continuación se muestra un fragmento de código donde se emplea dicho patrón.

***Observer*:** Este patrón se basa en tener observadores de una clase y cada vez que ocurra un cambio en esta, se notifica a todos los observadores. Teniendo en cuenta que para implementar la redundancia, todos los *Naming Service* deben actualizar su estado con las últimas modificaciones ocurridas, se hace necesario el uso de este patrón.

A continuación se muestra un fragmento de código donde se puede ver el empleo de estos patrones. En el caso del *adapter* se muestra la implementación de la misma interfaz de *NamingContext* y haciendo uso del *observer* se muestra la notificación que realiza por cada cambio ocurrido.

naming_context_adapter_impl.cpp

```
void NamingContextAdapter::notify()
{
    observer_.notify();
}
//-----------------------------------------------------------------------
void NamingContextAdapter::bind (const CosNaming::Name & n,
                  CORBA::Object_ptr obj)
{
    this->nc_->bind(n,obj);

    observer_.notify();
}
//-----------------------------------------------------------------------
void NamingContextAdapter::rebind (const ::CosNaming::Name & n,
                    CORBA::Object_ptr obj)
{
    this->nc_->rebind(n,obj);

    observer_.notify();
}
//-----------------------------------------------------------------------
void NamingContextAdapter::bind_context (const ::CosNaming::Name & n,
                          CosNaming::NamingContext_ptr nc)
{
    this->nc_->bind_context(n,nc);

    observer_.notify();
}
//-----------------------------------------------------------------------
CosNaming::NamingContext_ptr NamingContextAdapter::bind_new_context (const CosNaming::Name & n)
{
    CosNaming::NamingContext_ptr new_ctx = this->nc_->bind_new_context(n);

    observer_.notify();

    return new_ctx;
}
//-----------------------------------------------------------------------
```

Figura 4: Utilización de los patrones *Adapter* y *Observer*.

2.4 METODOLOGÍA, LENGUAJE Y HERRAMIENTAS

La metodología, lenguaje y herramientas utilizadas para el desarrollo del sistema de redundancia, no han sido objeto de selección en este caso, pues ya han sido analizadas y valoradas por analistas y arquitecto del centro. A continuación se detallan cada uno de los elementos a tener en cuenta para la utilización de las mismas.

Como metodología de desarrollo se empleará RUP (por sus siglas en inglés *Rational Unified Process*). RUP divide el proceso de desarrollo en ciclos, teniendo un producto final al culminarse cada una de las iteraciones. Es más apropiada para proyectos de gran envergadura, dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas, por lo que la hace ideal para los sistemas SCADA desarrollados en el CEDIN. El lenguaje de modelado a utilizar es UML (por sus siglas en inglés *Unified Modeling Language*) el cual es un lenguaje de modelado visual que se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema de software. Como herramienta CASE se utiliza *Visual Paradigm* la cual propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación, ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de

diagramas [5]. El entorno de desarrollo será Eclipse y el lenguaje de programación a utilizar es C++, este ofrece recursos muy útiles en la rama de la automatización y además es el lenguaje base de TAO.

2.5 PRUEBAS REALIZADAS

En esta sección se presentan los resultados obtenidos de algunos casos de prueba desarrollados para evaluar el comportamiento del sistema y constatar que permite contar con una alta disponibilidad.

Caso de prueba 1:

- Se inician 2 *Naming Service* localizados en 2 subredes diferentes y por los puertos 9999 y 8888 respectivamente.
- Se inicia un servidor con el nombre "config1" en el *Naming Service* del puerto 9999.
- Un cliente solicita la configuración del sistema al servidor de nombre "config1".

Resultado Esperado:

- El cliente recibe la configuración del sistema.

Resultado:

- Satisfactorio. El cliente recibe la configuración del sistema.

Caso de prueba 2:

- Se inician 2 *Naming Service* localizados en 2 subredes diferentes y por los puertos 9999 y 8888 respectivamente.
- Se inicia un servidor con el nombre "config1" en el *Naming Service* del puerto 9999.
- Se desconecta la red cableada donde corre el *Naming Service* del puerto 9999.
- Un cliente solicita la configuración del sistema al servidor de nombre "config1".

Resultado Esperado:

- El cliente recibe la configuración del sistema.

Resultado:

- Satisfactorio. El cliente recibe la configuración del sistema.

Caso de prueba 3:

- Se inician 2 *Naming Service* localizados en 2 subredes diferentes y por los puertos 9999 y 8888 respectivamente.
- Se inicia un servidor con el nombre "config1" en el *Naming Service* del puerto 9999.
- Se apaga la máquina donde corre el *Naming Service* del puerto 9999.
- Un cliente solicita la configuración del sistema al servidor de nombre "config1".

Resultado Esperado:

- El cliente recibe la configuración del sistema.

Resultado:

- Satisfactorio. El cliente recibe la configuración del sistema.

Caso de prueba 4:

- Se inician 2 *Naming Service* localizados en 2 subredes diferentes y por los puertos 9999 y 8888 respectivamente.
- Se inicia un servidor primario con el nombre "config1" en el *Naming Service* del puerto 9999.
- Se inicia un servidor secundario con el nombre "config1" en el *Naming Service* del puerto 9999.
- Se apaga la máquina donde corre el servidor primario.
- Un cliente solicita la configuración del sistema al servidor de nombre "config1".

Resultado Esperado:

- El servidor secundario se activa como primario con el mismo estado que tenía el servidor primario.
- El cliente recibe la configuración del sistema.

Resultado:

- Satisfactorio. El servidor secundario se activa como primario y el cliente recibe la configuración del sistema.

Estas pruebas simulan varios tipos de fallos que pueden ocurrir en las instalaciones donde se encuentran los servidores. Los resultados dejan claro, que se cuenta con un sistema de comunicaciones altamente disponible, lo cual permite darle cumplimiento al objetivo de esta investigación.

3. CONCLUSIONES

El desarrollo del sistema de comunicación redundante utilizando el middleware TAO, deviene en favorables resultados que se pueden concluir a continuación:

- El estudio realizado permitió la implementación del mecanismo de redundancia de manera que garantice una alta disponibilidad de los servicios que ofrecen los SCADA desarrollados por el CEDIN.
- Se diseñó e implementó el sistema de forma que no dependa de servicios externos para lograr la redundancia.
- Se realizaron pruebas al sistema de comunicación redundante a través de varios escenarios de fallas y se obtuvo resultados satisfactorios.

BIBLIOGRAFÍA

Henning, Michi y Vinosky, Steve. Advanced CORBA® Programming with C++. 12/02/1999.

Henning, Michi y Vinosky, Steve. Advanced CORBA® Programming with Student Workbook. Septiembre 2000.

Rodrigo Elgueta. Sistemas redundantes de Alta Disponibilidad, junio 2007. Disponible en: http://www.emb.cl/electroindustria/articulo.mvc?xid=725&tip=7

Wilson, Dale y Totten Steve. Fault Tolerant (FT) CORBA Services. Disponible en: http://www.dre.vanderbilt.edu/~schmidt/DOC_ROOT/TAO/docs/releasenotes/ftcorba_services.html

Página Oficial de TAO. (2013). Disponible en: http://www.theaceorb.com/

REFERENCIAS

Schmidt, D.C. Middleware for Distributed Systems. (2005). Disponible en: http://www.cs.wustl.edu/~schmidt/PDF/middleware-encyclopedia.pdf.

Zanabria Gálvez, Aldo Hernán. Common Object Request Broker Architecture "Modelo de Objetos Distribuido". Conceptos. (22/11/2010). Disponible en: http://es.scribd.com/doc/43672117/CORBA.

Página Oficial de TAO. (2013). Disponible en: http://www.theaceorb.com/product/abouttao.html.

Javier, I.M.P., Tesis "Selección de la tecnología adecuada para implementar el Middleware V 2.0." Universidad de las Ciencias Informáticas. (2009)

Portal EcuRed. Visual Paradigm. 2012. Disponible en: http://www.ecured.cu/index.php/Visual_Paradigm.

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.