Universidad de las Ciencias Informáticas
Facultad 5

“Mecanismo de disponibilidad para el Configurador SCADA UX.”

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor

Lisandra Sánchez Elías.

Tutores

Ing. Alejandro Manuel Rubinos Carvajal.

Ing. Luis Ángel Ravelo Hernández.

Cotutora

Ing. Rosalina Puerto Sorio.

“Ciudad de la Habana, Junio de 2011“

“Año 52 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaro ser autor de este trabajo de Diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos primordiales del mismo con carácter exclusivo.

Para que así conste se firma la presente a los 22 días del mes de Junio del año 2011.

Lisandra Sánchez Elías. Ing. Alejandro Manuel Rubinos Carvajal.

_____ _____

FIRMA DEL AUTOR **FIRMA DEL TUTOR**

Ing. Luis Ángel Ravelo Hernández.

_____

FIRMA DEL TUTOR

Ing. Rosalina Puerto Sorio.

_____

FIRMA DEL CO-TUTOR

DATOS DE CONTACTO

Tutor: Ing. Alejandro Manuel Rubinos Carvajal.
Edad: 25 años.
Título: Ingeniero en Ciencias Informáticas.
Correo: amrubinos@uci.cu

Tiene dos años de graduado en la Universidad de las Ciencias Informáticas (UCI). Actualmente se desempeña como jefe de la línea Seguridad del proyecto SCADA y como arquitecto del Dominio de Supervisión y Control del Centro de Informática Industrial de la Facultad 5.

Tutor: Ing. Luis Ángel Ravelo Hernández.
Edad: 25 años.
Título: Ingeniero en Ciencias Informáticas.
Correo: laravelo@uci.cu

Tiene dos años de graduado en la Universidad de las Ciencias Informáticas (UCI). Actualmente se desempeña como Desarrollador del proyecto comunicaciones del Centro de Informática Industrial de la Facultad 5.

PENSAMIENTO

"Todo lo que enaltece y honra, implica sacrificios."

José Martí

AGRADECIMIENTOS

A mi mamá por apoyarme siempre, y darme el aliento y la fuerza necesaria cuando más lo necesito, por depositar toda su confianza en mí, por estar siempre conmigo, por ser lo que eres, te quiero, gracias mami.

A mi papá y a mi tía Julia que en vida siempre dieron de sí todo lo que tenían con tal de que yo fuera feliz, me dieron su ejemplo, su apoyo, y si hoy tuvieran la oportunidad de verme graduada estarían muy orgullosos de lo que soy.

A mi abuela Milagro, por ser mi otra madre, por aconsejarme siempre, por estar siempre ahí para mí, por su preocupación, y por su presencia.

A mi hermanito Pochi, que aún siendo un niño, se ha portado como todo un hombrecito y también siempre cuidó y apoyó a su hermana.

A mi abuela Niña por apoyarme y darme el mejor de los regaños cuando era necesario.

A mis tías Idalvis, Ana, Mayra, Josefita por estar siempre, aún cuando les era difícil.

A toda mi familia en general, que de alguna manera me apoyaron.

A mis tutores y cotutora, al profesor Argelio, que no tengo quejas de ellos, mejor no los hubiera pedido, gracias por su paciencia en todo momento, gracias por ayudarme tanto, gracias por haberme elegido, espero estén orgullosos de mí, yo estoy muy orgullosa de haberlos tenido a ustedes como guías.

A mi tonejo lindo, por soportarme aún cuando ya no le quedaban gotas de paciencia, por ayudarme tanto en el desarrollo de esta tesis, por apoyarme y estar siempre conmigo, lástima que hayamos esperado tanto tiempo para estar juntos de esta manera.

A mis amigas Yaime, Evelyn y Lisanet que cada una desde su pedacito me ha enseñado lo que es la amistad, son las hermanas hembras que no tuve, las quiero mucho.

A las chicas superpoderosas, Islema, Baby, Yelena, Aly, Annia, Yady, si de fiesta se trata, ahí sí hay piquete, sé que puedo contar con ustedes.

A mis amigos Walny, Reidel, y Hung que también me apoyaron para llegar tan lejos.

A Joaquín por ser un excelente compañero, y por demostrarme que se puede salir del bache, cualquiera que sea.

Al más reciente de mis fanáticos Haniel, en poco tiempo entendimos que valdría la pena iniciar la amistad que hoy tenemos.

Otros recientes, Raydelmis y Fernando, la parejita más cómica y guarosa, y sé que me estiman, yo también los estimo cantidad.

A la otra parte del piquete de Santiago que tanto se divierte uno con ellos, Yoel, Pavel, y Husseyn.

A mis antiguos compañeros de grupo, a Jaime por sus ocurrencias y por darme una muy buena explicación siempre, a indi, Zulia y yeisi, por tan buen año.

A mis profesores en todo el transcurso de la carrera, que de alguna manera me enseñaron como ser mejor cada día.

A la universidad por darme la mejor de las oportunidades.

A la Revolución Cubana por existir.

GRACIAS.

DEDICATORIA

A mi mami, la mamá más bella y buena sobre la faz de la tierra, que me ha enseñado que con sacrificio todo en la vida se puede.

A mi papá que sé estaría muy orgulloso de la hija que tiene.

A mi abuelita Milagro que le ha dado sentido a todo lo que soy.

A mi hermanito, lo que más quiero en la vida.

A mi novio Edilberto, que ha hecho de mí la ingeniera que hoy soy.

RESUMEN

La Universidad de las Ciencias Informáticas (UCI) es uno de los pilares en Cuba en el desarrollo de aplicaciones software, es paradigma en la investigación de tecnologías novedosas y la creación de soluciones informáticas. En la universidad existe un centro de desarrollo de software en el campo de la automatización de procesos industriales denominado Centro de Informática Industrial (CEDIN), una de sus líneas de productos es el desarrollo de sistemas SCADA[1]. Actualmente se prevé crear un mecanismo que posibilite conocer para cada uno de los módulos del SCADA UX la disponibilidad de los mismos, además permitir siempre que fuera necesario a operadores y administradores del sistema el inicio, reinicio y parada de los servicios de forma gráfica y remota.

La presente investigación persigue como objetivo desarrollar un mecanismo que permita monitorear y controlar la disponibilidad de los módulos del configurador SCADA UX, para lo cual es usada la biblioteca ICE como principal tecnología. C++ como lenguaje de programación y como framework de desarrollo QT.

En el desarrollo del mecanismo se realiza un estudio y valoración de la tecnología ICE[2], además de los componentes a reutilizar y algoritmos más complejos a implementar, para la validación se aplicaron técnicas de prueba de software que permitieron comprobar la eficacia de la solución propuesta. Se detallaron los casos de prueba, que permitieron detectar la ocurrencia de errores y solución.

PALABRAS CLAVE

Disponibilidad, ICE, SCADA, tecnología.

[1] Supervisory Control and Data Adquisition, traducido al español Supervisión, Control y Adquisición Datos.

[2] Internet Engine Communication, traducido al español como motor de comunicación Internet.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La revolución de las Tecnologías de la Información y las Comunicaciones (TIC) es también la revolución de las redes. No se trata de la era de las computadoras y de la informática, sino de las computadoras funcionando en red.

En el amplio campo de la informática, y las TICs, tiene lugar la automatización de procesos industriales, que no es más que el uso de sistemas o elementos computarizados y electromecánicos para controlar maquinarias y/o procesos industriales sustituyendo a operadores humanos. La automatización y supervisión de procesos industriales nace de la necesidad entre otras, de recoger, almacenar y visualizar la información, además de las limitaciones de la visualización de los sistemas de adquisición y control; de manera que surgen sistemas que permiten supervisar y controlar variables de proceso a distancia, denominados SCADA.

Un SCADA es una aplicación de software, especialmente diseñada para funcionar sobre ordenadores en el control de la producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, entre otros.). Además, envía la información generada en el proceso productivo a diversos usuarios, tanto del mismo nivel como hacia otros supervisores dentro de la empresa, es decir, que permite la participación de otras áreas como por ejemplo: control de calidad, supervisión, mantenimiento, entre otros. Dentro de las características de estos sistemas se encuentra su fácil adaptación a múltiples entornos; así como la posibilidad de integración con otros sistemas, ya sean gerenciales, de gestión u otros.

En sus orígenes, los sistemas SCADA eran controlados físicamente por empleados con distintos grados de acceso según la criticidad del sistema (lo cual se sigue manteniendo por cuestiones de seguridad y privilegios). Actualmente, y debido a las nuevas tecnologías, los sistemas SCADA se han ido introduciendo poco a poco en nuevas tendencias tecnológicas, entre ellas la más famosa y de mayor auge, Internet. De esta forma es posible controlar los sistemas remotamente aprovechando las comunicaciones TCP/IP, consiguiendo por tanto, reducir costes y mejorando funcionalidades y eficiencia en los diferentes procesos de cada sistema en concreto. No solamente se han aprovechado las bondades de la comunicación que hoy en día ofrece la red, sino también de los sistemas operativos más comunes como pudieran ser Microsoft Windows o Linux entre otros. Debido a la inclusión de sistemas de amplio uso sobre sistemas aislados como eran los sistemas SCADA, se conoce que

comienzan a derivar problemas para los que los sistemas SCADA no se encontraban preparados y no eran tan frecuentes ni sofisticados como los que se encuentran en la red, como malware [3]en general, así como los crackers y hackers que se encuentran detrás de muchos ataques que se dan hoy en día en Internet.

Los sistemas SCADA fueron originalmente desarrollados en una época en la que las tecnologías de la información estaban centradas en grandes mainframes[4]. Las redes de computadoras eran algo anecdótico. En este escenario los primeros SCADA se concibieron como aplicaciones autónomas, sin prácticamente conectividad externa. Se crearon redes WAN (Wide Area Network) específicas para conectar con los RTU[5]s. Si se requería redundancia en el sistema la solución típica era duplicar cada componente, dejando el sistema secundario en segundo plano, monitorizando al principal para tomar el control en caso de error.

Debido a las funciones que desempeñan los sistemas SCADA, algunos de éstos deben operar 24x7 ofreciendo alta disponibilidad de manera que el servicio no se vea interrumpido, este es el caso de procesos como la gestión de una central nuclear o los de una central eléctrica entre otros muchos. Al introducir elementos nuevos como Internet en los sistemas SCADA, se requieren actuaciones propias a tener en cuenta para adaptar los sistemas antiguos a los nuevos. Es por esto que se requieren de medidas como Firewalls, IDS[6], antivirus, auditorías y revisiones de controles de privilegios, que precisan un mantenimiento que puede dificultar o diferir con el objetivo que tienen los sistemas SCADA.

La naciente industria cubana del software cuenta con numerosos retos económicos y sociales, para acelerar la informatización de la sociedad cubana. La Universidad de las Ciencias Informáticas (UCI) forma parte de este desarrollo informático, cuya misión es formar profesionales comprometidos con la Revolución, partiendo de un modelo pedagógico flexible, que vincula dinámicamente el estudio con la producción y la investigación.

[3] Malware ,del inglés malicious software, también llamado badware, software malicioso o software malintencionado.

[4] Una computadora central o mainframe es una computadora grande, potente y costosa usada principalmente por una gran compañía para el procesamiento de una gran cantidad de datos.

[5] Unidad de Terminal Remota.

[6] Intruse Detection System, traducido al español sistema de detección de intrusos.

La universidad promueve la investigación de tecnologías novedosas, la puesta en práctica de metodologías de trabajo reconocidas internacionalmente y la creación de soluciones informáticas en la producción de software, donde el Centro de Informática Industrial (CEDIN) toma participación y protagonismo; siendo una de sus líneas de productos el desarrollo de sistemas SCADA *(Supervisory Control And Data Adquisition).*

La solución configurador SCADA UX desarrollada en el CEDIN posee como principales módulos:

- ✓ Adquisición: Realiza la recolección de datos desde dispositivos de campos a través de los manejadores.
- ✓ Seguridad: Desarrolla componentes que garanticen la seguridad de entornos relacionados con sistemas de supervisión y control de datos.
- ✓ Visualización: Muestra los datos, alarmas, eventos.
- ✓ Histórico: Guarda datos que persistan en el tiempo.
- ✓ Comunicación: Provee una interfaz OPC (OLE[7] for Process Control) para el acceso a los datos del SCADA, con los sistemas de control externos existentes.

Actualmente esta solución no cuenta con un mecanismo que permita la emisión de notificaciones o señales de vida por parte de cada uno de los módulos de modo que se pueda conocer la disponibilidad de éstos, no siendo posible para aquellos módulos que dependen de otros conocer si éstos se encuentran iniciados, así como posibilitar a operadores y administradores el inicio, parada, y reinicio de servicios de manera gráfica y remota. Es por ello que se formuló el siguiente **problema científico:** ¿Cómo garantizar la disponibilidad de los módulos del configurador SCADA UX?

A su vez el **objeto de estudio** se centra en: mecanismos de disponibilidad y redundancia en redes distribuidas delimitando como **campo de acción**: mecanismo de monitoreo y control de la disponibilidad para el configurador del proyecto SCADA UX.

El **objetivo** de este trabajo consiste en: desarrollar un mecanismo que permita monitorear y controlar la disponibilidad de los módulos del configurador SCADA UX.

[7] Object Linking and Embedding, traducido al español (Vinculación e Incrustación de objetos) para Control de Procesos Industriales

Como **preguntas científicas** se denotaron:

1. ¿Cuáles son los antecedentes históricos sobre el desarrollo de mecanismos de monitoreo y control de disponibilidad?
2. ¿Cuáles son el estado y la tendencia actual para el desarrollo de mecanismos de monitoreo y control de la disponibilidad?
3. ¿Cómo desarrollar un mecanismos de monitoreo y control de la disponibilidad para el sistema Configurador SCADA UX?

De las cuales se derivan las siguientes **tareas investigativas**:

1. Estudio de los antecedentes históricos sobre el desarrollo de mecanismos de monitoreo y control de la disponibilidad.
2. Estudio y caracterización de las tendencias actuales para el desarrollo de mecanismos de monitoreo y control de disponibilidad.
3. Estudio de las diferentes metodologías, lenguajes de programación y herramientas que existen para el desarrollo del mecanismo de disponibilidad para el configurador SCADA UX.
4. Documentación y justificación de la selección de las herramientas y tecnologías a utilizar para desarrollar el mecanismo.
5. Modelaje para el desarrollo de las funcionalidades relacionadas con el mecanismo de disponibilidad para el SCADA UX.
6. Desarrollo y prueba del mecanismo.

Después de obtener toda la información relacionada con el tema del trabajo se plantea como **posible resultado**: Obtención de un mecanismo que facilitará el monitoreo y control de la disponibilidad de los módulos del SCADA UX.

Métodos Científicos a Utilizar:

Teóricos:

- ✓ **Analítico Sintético:** Admite el análisis de todo el contenido de características específicas del mecanismo de disponibilidad y a su vez llegar a conceptos y generalizaciones, de manera que permita sintetizar todo lo obtenido como un todo.

- ✓ **Inductivo-Deductivo**: Permite como proceso lógico del pensamiento apreciar la necesidad existente en el proyecto SCADA UX de la implementación del mecanismo, además de tener presente la ventaja que ofrece el mecanismo de disponibilidad para el proyecto con el procesamiento y obtención de los resultados.
- ✓ **Modelación:** Cede a la elaboración de los diferentes diagramas necesarios para el entendimiento y mantenimiento del producto en cuestión.
- ✓ **Generalización:** Permite sistematizar en cada capítulo de la investigación las pautas más significativas de la misma, así como llegar a conclusiones más objetivas y explícitas para cada caso.
- ✓ **Revisión Documental:** Admite la consulta de la bibliografía confiable y más completa referente al tema de la investigación.

El presente Trabajo de Diploma está estructurado de la siguiente manera: Introducción, cuatro capítulos de contenido, Conclusiones, Recomendaciones, Referencias Bibliográficas, Bibliografía, Anexos, y Glosario de Términos.

- **Capítulo 1:** "Fundamentación Teórica", en este capítulo se analiza con mayor profundidad el objeto de estudio. Se explica además la metodología y algunas herramientas de desarrollo; así como las tecnologías actuales a considerar para el desarrollo del mecanismo de disponibilidad.
- **Capítulo 2**: "Herramientas y Tecnologías", en el mismo se explican detalladamente partiendo del estudio realizado en el capítulo anterior las herramientas, metodologías y tecnologías actuales definidas para su uso en la implementación del mecanismo, así como se documentan algunas de las características que las hacen relevantes en el desarrollo del mecanismo en cuestión.
- **Capítulo 3:** "Propuesta de la Solución", en esta sección se describe la arquitectura que se emplea en el desarrollo del mecanismo de disponibilidad. También se muestra el diagrama de clases de diseño, y las descripciones de las clases que forman parte de este diagrama de clases del diseño.

- **Capítulo 4:** “Implementación y Prueba”, en este capítulo se muestran los diagramas de despliegue, de componentes, pruebas realizadas al mecanismo de disponibilidad y se obtiene además una versión del producto final.

CAPÍTULO 1 "FUNDAMENTACIÓN TEÓRICA"

1.1 Introducción

En el presente capítulo se analizan y describen los sistemas SCADA y se explican las principales características de los mecanismos que garantizan disponibilidad, además se realiza un estudio y valoración de las tendencias que son más utilizadas actualmente en el desarrollo de los mecanismos de disponibilidad.

1.2 Sistemas Distribuidos

Un sistema distribuido no es más que un conjunto de elementos, tanto de procesamiento como de almacenamiento que están conectados a través de una red y que para un usuario del sistema se comporta como un único computador.

Existen seis características que son responsables de la utilidad de los sistemas distribuidos: compartir recursos, apertura, concurrencia, tolerancia a fallas y transparencia. Se hace notar que estas características no son consecuencia automática de la distribución; el software del sistema y de las aplicaciones debe ser cuidadosamente diseñado para asegurar que ellas sean conseguidas.

Por su estructura y sus características los sistemas distribuidos ofrecen grandes ventajas a los usuarios. La primera de ellas es que se adecuan a un concepto común en la vida de los seres humanos: la distribución. Es decir, el hombre comparte información y recursos sin necesidad de complejos protocolos de comunicación u otros requerimientos, simplemente se comunica. Así que los sistemas distribuidos se acercan más al concepto que tiene el hombre de compartir recursos**. (Ravelo Hernández, 2010)**

1.3 Sistemas SCADA

Uno de los ejemplos de aplicación de sistemas distribuidos son los sistemas de automatización industrial. Uno de estos sistemas se denomina SCADA, que es el acrónimo de Supervisory Control And Data Adquisition (Supervisión, Control y Adquisición de Datos).

Los sistemas SCADA son aplicaciones de software para la adquisición, supervisión y control de datos mediante la comunicación con los dispositivos de campo, con la finalidad de controlar procesos automatizados desde la pantalla del ordenador. Estos sistemas automatizados están formados por diferentes programas de software que corren en una computadora y cuyo objetivo es visualizar todos los

datos que se miden y que permita el control de los mismos de manera simple y efectiva. Provee información del proceso a diversos usuarios como son: operadores, supervisores de control de calidad, mantenimiento. Los sistemas SCADA interactúan con ordenadores, unidades remotas, sistemas de comunicación que efectúan las tareas de supervisión, gestión de los datos y el control total de los procesos. **(Guerra Garayta, 2009)**

Los SCADA utilizan la computadora y tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales. Estos sistemas son partes integrales de la mayoría de los ambientes industriales complejos o muy geográficamente dispersos ya que pueden recoger la información de una gran cantidad de fuentes rápidamente, y la presentan a un operador en una forma amigable. Los sistemas SCADA mejoran la eficacia del proceso de monitoreo y control proporcionando la información oportuna para poder tomar decisiones operacionales apropiadas. Ver ilustración 13 en los anexos.

1.3.1 Módulos de un SCADA

Los módulos o bloques software que permiten las actividades de adquisición, supervisión y control son los siguientes:

- ✓ Configuración: Permite al usuario definir el entorno de trabajo de su SCADA adaptándolo a la aplicación particular que se desea desarrollar.
- ✓ Interfaz gráfico del operador: Proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado en el SCADA o importados desde otra aplicación durante la configuración del paquete.
- ✓ Módulo de proceso: Ejecuta las acciones de mando preprogramadas a partir de los valores actuales de variables leídas.
- ✓ Gestión y archivos de datos: Almacena y procesa ordenadamente los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.
- ✓ Comunicaciones: Proporciona la transferencia de información entre la planta y la arquitectura hardware que soporta el SCADA, y entre ésta y el resto de elementos informáticos de gestión.

1.2.2 Funcionalidades de los sistemas SCADA

Como su nombre lo indica, estos se encargan de la adquisición, refiriéndose a recolectar, procesar y almacenar la información; la supervisión, para observar la evolución del proceso desde un monitor; y el control de los procesos para modificarlos. Estos sistemas son capaces de recolectar información rápidamente de una gran variedad de fuentes, y presentarla a un operador de forma sencilla de interpretar. A continuación se muestran sus principales funcionalidades **(Valenciano Odio, 2010):**

- **Adquisición y almacenamiento de datos:** Es la recolección, procesamiento y almacenamiento de la información recibida, en forma continua y confiable.

- **Supervisión de procesos:** Es el monitoreo del funcionamiento del proceso, procesamiento estadístico de los datos y confección de reportes. Además se brindan notificaciones al operador del sistema sobre cambios detectados en la instalación. Estas notificaciones se clasifican principalmente en alarmas y eventos. Las alarmas se basan en la vigilancia de los parámetros de las variables del sistema. Son los sucesos no deseables, porque su aparición puede dar lugar a problemas de funcionamiento. Este tipo de sucesos requieren de la atención de un operario para su solución antes de que se llegue a una situación crítica que detenga el proceso. El resto de las situaciones normales, tales como puesta en marcha, paro, cambios de consignas de funcionamiento, consultas de datos, entre otras, serán los denominados eventos del sistema o sucesos. Los eventos no requieren de la atención del operador del sistema, registran de forma automática todo lo que ocurre en el sistema. También será posible guardar estos datos para su posterior consulta.

- **Control de procesos:** Es el control de los procesos, actuando sobre los reguladores autónomos básicos como eventos y alarmas, o directamente sobre el proceso mediante las salidas conectadas.

- **Transmisión de datos:** Es la transmisión de información entre dispositivos de campo y computadoras de control o entre dispositivos ubicados a un mismo nivel.

- **Presentación de la información:** Es la representación gráfica de los datos. Interfaz del Operador o HMI (Human Machine Interface).

1.2.3 Beneficios del uso de los Sistemas SCADA

Los SCADA son utilizados para dirigir cualquier tipo de equipamiento. Son empleados para automatizar un proceso industrial complejo. En dicho proceso, el control manual es muy poco práctico y difícil, ya que tiene tantos controles y accesorios que al humano le resultaría difícil de manipular y dirigir.

A continuación se mencionan algunos de los beneficios del uso de los sistemas SCADA **(Valenciano Odio, 2010)**:

- A través del monitoreo de los diferentes procesos, haciendo uso extensivo de los avances tecnológicos, es posible identificar los problemas y encontrarles solución antes del acaecimiento de un detenimiento no deseado y perjudicial de las actividades monitoreadas.

- La captura, procesamiento y presentación de extensas cantidades de datos a un operador pueden ser realizadas de forma más efectiva a través de un SCADA.

- Es posible operar remotamente con la mayoría de estos sistemas de control sobre los servicios o procesos productivos, la restauración tras una interrupción puede ser efectuada a distancia con mayor velocidad.

- El SCADA también ayuda a determinar la localidad de donde proviene la interrupción, la severidad de la misma y el número de personas afectadas, informaciones importantes para garantizar la asignación apropiada de los recursos necesarios para la recuperación. Sus reportes permiten analizar las causas de las diversas problemáticas.

- A través de la integración de una gran variedad de dispositivos electrónicos de control y red que facilitan el monitoreo y administración, los SCADA posibilitan la obtención de niveles de fiabilidad y robustez nunca antes pensados.

- Mediante el despliegue de un SCADA centralizado es posible reducir los costos de operación y mantenimiento debido a que se requiere menos personal para monitorear equipamientos de localidades remotas, resultando en un incremento de la efectividad del operador y menor número de viajes de mantenimiento.

- La habilidad de monitorear y controlar los procesos desde una única localidad, y al mismo tiempo recolectar y compartir en tiempo real los datos de los procesos, se ha convertido en una herramienta invalorable para el mejoramiento de las operaciones.

- Permiten brindar respuestas frente a las situaciones de emergencia de una forma mucho más rápida y efectiva porque los datos son analizados antes, durante y después de cada evento.

- Automatizaciones de muchas otras funciones específicas brindan eficiencias adicionales. El sistema pudiera incluso ser reiniciado remotamente en momentos que sea necesario por situaciones críticas o de alarma.

1.4 Middleware

Una parte muy importante de los Sistemas Distribuidos es su Subsistema de Comunicación. Éste es el encargado de gestionar los mensajes y el traslado de información de un componente del sistema a otro. Esto permite que se organice el trabajo entre todos los elementos que lo componen alcanzándose un mayor uso de los recursos y obteniéndose resultados concretos en un corto período de tiempo. La mayoría de los Sistemas SCADA que más se comercializan en la actualidad, utilizan como Subsistema de Comunicación un Middleware.

El Middleware es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El Middleware abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API[8] para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funciones necesarias, serán útiles diferentes tipos de servicios de middleware. **(Ravelo Hernández, 2010)**

1.5 Disponibilidad

Por disponibilidad de un sistema informático se entiende la medida en la que sus parámetros de funcionamiento se mantienen dentro de las especificaciones de diseño.

[8] Application Programming Interface, Interfaz de Programación de Aplicaciones (API).

De una forma genérica se podría hablar del término "disponibilidad" aplicado a un sistema computacional del que se requiere un funcionamiento continuo.

La disponibilidad se aplica a toda la gama de soluciones que sostienen los sistemas de información de las empresas: bases de datos, cortafuegos, servidores web. Si bien, los mecanismos que se emplean pueden ser distintos en función del entorno.

Se encuentran entonces distintas configuraciones:

Activo-Pasivo.- Se trata de disponer de un nodo [9]funcionando, contando con todos los servicios que componen el sistema de información al que denominaremos Activo, y el otro nodo que se denominará Pasivo, en el que se encuentran duplicados todos estos servicios, pero detenidos a espera de que se produzca un fallo.

Activo-Activo.- La configuración de "disponibilidad" en activo-activo es muy similar a la de activo-pasivo, aunque en este caso los dos nodos comparten los servicios de una manera activa, normalmente balanceados, consiguiendo una disponibilidad mayor ya que los servicios se entregan antes.

Granja de servidores.-Normalmente orientado a servicios web, servicios computacionales que se entregan de forma masiva, como puedan ser servicios terminales. En estas configuraciones no solo es importante la fiabilidad, también es importante contar con un sistema muy disponible por lo que se suele colocar un gran número de máquinas haciendo una tarea común. Esta configuración siempre va a permitir que en caso de que un nodo deje de hacer su función otro asuma su rol.

1.5.1 Principios básicos de disponibilidad.

Los principios básicos de la "alta disponibilidad", (también se encuentra representado frecuentemente con las siglas H.A.), asocia los siguientes conceptos:

- ✓ Fiabilidad.- Marca la medida en la que un dispositivo computacional se mantiene activo.
- ✓ Disponibilidad.- La medida en la que un sistema de información está preparado para su uso.
- ✓ Confiabilidad.-Grado de eficacia del sistema de información.
- ✓ Failover.- Configuración de mínimo dos nodos en el que, en un momento dado y debido a cierta particularidad, un nodo continua activo en vez del otro.

[9] En este contexto, un nodo es una computadora que contiene los diferentes servicios, funciona como servidor.

- ✓ TakeOver.- Failover automático, cuando un fallo es detectado a partir de una monitorización.

1.5.2 Sistemas de disponibilidad y sistemas tolerantes a fallos

En un sistema tolerante a fallos, cuando se produce un fallo hardware, el hardware asociado a este tipo de sistema es capaz de detectar el subsistema que falla y obrar en consecuencia para restablecer el servicio en segundos (o incluso décimas de segundo). El cliente del servicio no notará ningún tiempo de fuera de servicio. En los sistemas de disponibilidad existen los tiempos de fuera de servicio; son mínimos pero existen, van desde 1 minuto o menos hasta 5 o 10 minutos, según sea el caso. En teoría esta es la única diferencia entre ambos, pero en los últimos años, se ha ido acercando la idea de disponibilidad a la idea de tolerancia a fallos, debido al abaratamiento de hardware, y de ciertas tecnologías que han ido surgiendo. Estas tecnologías han evolucionado de tal forma, que han logrado que subsistemas donde había que recurrir a la disponibilidad ahora, se puede lograr tolerancia a fallos a bajo precio.

En una configuración de High Availability (alta disponibilidad o HA), se toman dos servidores separados y se unen parar formar un *cluster*[10], combinados con un arreglo de discos protegidos, el sistema tiene todos sus componentes duplicados. Si un componente falla, éste puede ser automáticamente o manualmente reemplazado con su duplicado.

Algún tiempo de caída ocurre en las configuraciones HA, pero en la mayoría de los casos ésta tiene un tiempo limitado de duración. Tiempos de operación que excedan el 99.98 por ciento pueden ser consideradas en este nivel. Para el diseño de sistemas configurados en disponibilidad se requieren diseño de red, consideraciones de soporte de sistema, administración de sistema y red, y mecanismos de auditoría que aseguren que se tienen cubiertos todos los escenarios. Una desventaja en la implementación de este nivel de disponibilidad es que existe un cierto grado de complejidad que se agrega al ambiente, y la complejidad provoca que sea más difícil su administración. La otra desventaja es el costo. Se está agregando redundancia, y esto requiere dinero para los sistemas, redes, tiempo de los ingenieros, soporte y configuración.

La tolerancia a fallos, tal y como se conoce hoy en día, se basa fundamentalmente en un concepto: redundancia. La mejor forma de asegurar la disponibilidad de los equipos y los servicios que ellos suministran de manera fiable y sin interrupción las 24 horas del día durante siete días a la semana, es la

[10] Un cluster de ordenadores es, básicamente, un sistema distribuido en paralelo que consiste en dos o más servidores interconectados compartiendo sus recursos y que son vistos como si se tratase de uno solo.

duplicación de todos sus elementos críticos y la disposición de los elementos software y hardware necesarios para que los elementos redundantes actúen cooperativamente, bien sea de forma activa-activa o activa-pasiva, pero siempre de forma transparente para el usuario final.

1.6 Sistemas SCADA con alta disponibilidad existentes

La falta de previsión en la implementación de estos sistemas suele acabar con toda una serie de "remiendos" añadidos que, generalmente, provocarán fallos de funcionamiento o de seguridad donde menos se espere.

Hoy en día, los sistemas de control ya prevén situaciones de este tipo, y por ejemplo, permiten distribuir los datos "delicados" entre varios equipos que se van actualizando de forma automática, permitiendo que cualquiera de ellos tome el control si uno falla.

En la imagen siguiente, servidores redundantes del SCADA WinCC[11], de Siemens, proporcionan una elevada disponibilidad de red ante fallos. Tanto en los servidores, como los autómatas, pueden configurarse como elementos redundantes gracias a la herramienta adecuada (WinCC/Redundancy). Ver ilustración 14 en anexos.

SIMATIC WinCC, como es también conocido, es considerado en la actualidad el número uno en el terreno mundial de la automatización. Este hecho en parte también se debe a que SIMATIC ofrece las seis propiedades del sistema típicas de Totally Integrated Automation[12]:

• Ingeniería
• Comunicación
• Diagnóstico
• Seguridad en máquinas
• Protección de datos
• Robustez

1.6.1 Características destacadas del WinCC

Dentro de las características más relevantes del sistema de monitoreo y control SCADA WinCC que lo hacen de gran reconocimiento global se puede mencionar:

[11] Windows Control Center bajo Microsoft Windows

[12] Automatización totalmente integrada

- Utilización universal.

  - ✓ Soluciones para todos los sectores industriales.
  - ✓ Configuración multilingüe para su utilización en todo el mundo.
  - ✓ Integrable en todas las soluciones de automatización.

- Integradas todas las funciones de mando y supervisión.

El equipamiento básico del sistema incluye funciones de mando y supervisión diseñadas a la medida de las necesidades industriales para la visualización gráfica completa de los procedimientos y estados del proceso, señalización y confirmación de alarmas, archivo de los valores de medida y mensajes, listado de todos los datos de proceso y de archivo, gestión de los usuarios y sus autorizaciones de acceso. Los procedimientos y eventos relevantes en materia de calidad son registrados continuamente y así pueden ser seguidos fácilmente.

- Configuración sencilla y eficiente.

Las sofisticadas funciones de configuración reducen drásticamente el tiempo y el trabajo de ingeniería y de formación: editor gráfico cómodo y orientado al objeto (adaptable y ampliable individualmente utilizando Visual Basic for Applications), extensas bibliotecas, modularidad, modificaciones rápidas por medio de la configuración online, herramienta de configuración para el tratamiento de grandes cantidades de datos, transparencia gracias a la lista de referencias cruzadas.

a. Escalable de modo continuo, también vía Web.

WinCC ofrece esta escalabilidad continua y homogénea, desde la pequeña solución mono puesto hasta una arquitectura cliente / servidor redundante con servidor central y puestos de operador en la Web.

b. Estándares abiertos para una integración sencilla.

El sistema WinCC se basa en los más altos niveles de apertura y capacidad de integración: controles ActiveX para ampliaciones en tecnologías o sectores específicos, comunicación no propietaria con el proceso vía OPC, interfaces estándares para el acceso externo a la base de datos (WinCC OLE-DB y OPC HDA), lenguaje de scripts estándar integrado (VBScript y ANSI-C), acceso a los datos y las funciones del sistema vía Application Programming Interface (API) utilizando la opción Open Development Kit (WinCC/ODK), ampliaciones personalizadas de los editores WinCC a través de Visual Basic for Applications (VBA).

1.6.1.1 Gamas de Producto de WinCC

Son variadas las gamas de producto del sistema que se analiza, solo se hará mención de algunas que incluye la de disponibilidad como la más importante en cuestión. Se podrían mencionar:

Configuración de instalaciones escalables

WinCC/Server – Sirve para la expansión de una solución mono puesto a un potente sistema cliente-servidor (distribuido) con hasta 12 servidores WinCC y 32 clientes.

Ampliaciones del sistema

WinCC/IndustrialX – Configuración de objetos personalizados en técnica ActiveX, lo que permite la estandarización, aplicación universal y modificación de forma centralizada.

Mayor disponibilidad

WinCC/Redundancy – Aumenta la disponibilidad del sistema mediante estaciones WinCC o servidores redundantes que se vigilan mutuamente y aseguran de esta forma la manejabilidad de la instalación y una adquisición de datos sin lagunas.

WinCC/ProAgent – Permite llevar a cabo un diagnóstico rápido y directo del proceso en máquinas e instalaciones.

SIMATIC Maintenance Station – Visualiza la información necesaria para el mantenimiento de toda la automatización.

SIMATIC WinCC Posibilita el diseño redundante del sistema adaptándolo al proceso del usuario. La escalabilidad en este caso es prácticamente ilimitada, va desde controladores redundantes hasta conexiones periféricas redundantes, pasando por sistemas de bus. El rasgo particular del mismo es que la programación se realiza en el mismo entorno y la arquitectura redundante en la estación periférica se resuelve a través de componentes estándar. Con los controladores de la línea es además posible el “hot-stand-by”: en caso de falla, el controlador redundante asume el control en el término de 100 ms. De ser necesario, los controladores de alta disponibilidad pueden posicionarse a 10 km unos de otros.

1.6.1.2 Herramienta WinCC/Redundancy

WinCC Redundancy ofrece un claro incremento de la disponibilidad de WinCC y de la instalación en su conjunto gracias al funcionamiento paralelo de dos PC servidores acoplados entre sí. Ver ilustración 15 en los anexos.

Mientras dura la avería de un servidor, el otro servidor sigue archivando todos los avisos y datos de proceso del proyecto WinCC. Cuando el servidor averiado vuelve a ponerse en funcionamiento se copian en él automáticamente los contenidos de todos los archivos de avisos, archivos de valores de proceso y ficheros de usuario, rellenando así las lagunas que se habían creado en los datos del servidor mientras éste estaba fuera de funcionamiento. A esta operación también se le denomina sincronización tras el restablecimiento.

La opción de WinCC Redundancy ofrece al usuario:

- ✓ La sincronización automática de los ficheros de avisos, de valores de proceso y de usuario tras el establecimiento de un servidor que había fallado.
- ✓ La sincronización automática de los ficheros de avisos, de valores de proceso y de usuario tras fallar el acoplamiento al proceso.
- ✓ La sincronización online de avisos internos.
- ✓ La sincronización online de ficheros de usuario.
- ✓ La conmutación automática de los clientes entre los servidores redundantes en caso de fallar uno de éstos, entre otros.

Para WinCC Redundancy se han de cumplir los siguientes requisitos:

- ✓ Los servidores WinCC redundantes con servicio multipuesto requieren el empleo de ordenadores con un sistema operativo para servidores de Windows 2000.
- ✓ Un requisito de Redundancy es que los servidores tengan la hora sincronizada. Sin embargo, conviene prever la sincronización horaria para toda la instalación (ordenadores WinCC, autómatas programables). Esto puede efectuarse mediante la opción WinCC "Timesynchronization".
- ✓ Los valores de proceso, avisos y bloqueos de avisos activos procedentes de los autómatas programables subordinados se emiten simultáneamente a ambos servidores.
- ✓ Cada uno de los dos servidores tiene la opción Redundancy.
- ✓ Los servidores Redundancy tienen que tener configuraciones funcionalmente idénticas.

1.7 Conclusiones del capítulo

En el caso particular del trabajo de investigación y desarrollo del mecanismo de disponibilidad para el configurador SCADA UX, partiendo de la teoría aplicada en el SCADA WinCC reconocido mundialmente y número uno en su tipo, de su implementación de disponibilidad, no se adopta como solución la herramienta WinCC/Redundancy debido a que uno de los requisitos que se tiene que cumplir para ésta es que los servidores WinCC redundantes requieren el empleo de ordenadores con un sistema operativo para servidores de Windows 2000 y para el desarrollo del mecanismo se prevé el uso de herramientas libres. Teniendo en cuenta toda la información recopilada y documentada anteriormente acerca de los sistemas de disponibilidad y tolerantes a fallos; los principios básicos de disponibilidad, las distintas configuraciones existentes: activo-activo, activo-pasivo, y granja de servidores y por último los conocimientos expuestos acerca de los sistemas SCADA así como la necesidad de la existencia del mecanismo que disponibilidad en el proyecto SCADA UX, se exponen a continuación las herramientas y metodología seleccionada para el desarrollo del mecanismo en cuestión.

Capítulo 2: "Herramientas y Tecnologías"

2.1 Introducción

En el presente capítulo se definen las tecnologías y herramientas que serán empleadas para la construcción del mecanismo, así como algunas de sus principales características que las hacen relevantes ante otras de su tipo. Para algunos casos la selección estuvo dada debido a que son herramientas usadas por el proyecto SCADA UX en la implementación de sus productos y funcionalidades.

2.2 Tecnologías libres para desarrollar mecanismo de disponibilidad.

2.2.1 Sistema Operativo: GNU/Linux

El sistema operativo es el programa (o software) más importante de un ordenador. Para que funcionen los otros programas, cada ordenador de uso general debe tener un sistema operativo. Los sistemas operativos realizan tareas básicas, tales como reconocimiento de la conexión del teclado, enviar la información a la pantalla, no perder de vista archivos y directorios en el disco, y controlar los dispositivos periféricos tales como impresoras, escáner, entre otros.

En sistemas grandes, el sistema operativo tiene incluso mayor responsabilidad y poder, es como un policía de tráfico, se asegura de que los programas y usuarios que están funcionando al mismo tiempo no interfieran entre ellos. El sistema operativo también es responsable de la seguridad, asegurándose de que los usuarios no autorizados no tengan acceso al sistema.

GNU/Linux es un sistema operativo que ha sido desarrollado bajo los principios del software libre. Software libre se refiere a la libertad de los usuarios para ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. De modo más preciso, se refiere a cuatro libertades de los usuarios del software:

- ✓ La libertad de usar el programa, con cualquier propósito (libertad 0).
- ✓ La libertad de estudiar cómo funciona el programa, y adaptarlo a tus necesidades (libertad 1). El acceso al código fuente es una condición previa para esto.
- ✓ La libertad de distribuir copias, con lo que puedes ayudar a tu vecino (libertad 2).

- ✓ La libertad de mejorar el programa y hacer públicas las mejoras a los demás, de modo que toda la comunidad se beneficie (libertad 3). El acceso al código fuente es un requisito previo para esto. **(Puerto Sorio, 2010)**

Se selecciona el sistema operativo GNU/Linux que por su exclusividad de ser software libre, resulta un sistema de alta calidad tecnológica con menos errores que los demás sistemas comerciales, a un costo bajo o muy bajo y con la disponibilidad del código fuente que permite aprender, modificar o ayudar al desarrollo del sistema.

2.2.2 Metodología de Desarrollo: RUP

Una metodología de desarrollo es un conjunto de procedimientos, técnicas y herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software.

Una metodología puede seguir uno o varios modelos de ciclo de vida, es decir, el ciclo de vida indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo. La metodología indica cómo hay que obtener los distintos productos parciales y finales.

El propósito de una metodología de desarrollo es garantizar la eficacia (p. ej. cumplir los requisitos iniciales) y la eficiencia (p. ej. minimizar las pérdidas de tiempo) en el proceso de desarrollo de software. Se selecciona RUP como metodología de desarrollo debido a sus características de ser: dirigido por casos de uso, iterativo e incremental y centrado en la arquitectura.

- ✓ Dirigido por caso de uso[13]: Significa que el proceso de desarrollo sigue una trayectoria a través de flujos de trabajo generados por casos de uso.
- ✓ Iterativo e Incremental: Posibilita que se pueda fragmentar el trabajo en partes más pequeñas o en mini proyectos, permitiendo la proporción entre casos de uso y arquitectura durante cada mini proyecto. Cada parte se puede ver como una iteración de la cual se obtiene un incremento, provocando un acrecentamiento del producto.
- ✓ Centrado en la Arquitectura: Provee la visión del sistema completo. La arquitectura describe los componentes del modelo que son más importantes para su construcción, o sea, los orígenes del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente.

[13] Casos de uso: Describen la funcionalidad del sistema, en función de su importancia para el usuario.

El RUP divide el proceso de desarrollo en fases, obteniendo un producto final al culminar cada fase.

En cada fase se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso. Además, se define la arquitectura del sistema cuyas decisiones se toman sobre la base entendimiento del mismo y de los requisitos (funcionales y no funcionales) identificados de acuerdo al alcance definido. Por último, se obtiene un producto listo para su manejo y correctamente documentado. Ver 16 en anexos.

2.2.3 Lenguaje de modelado: UML

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software.

Debido a que la metodología de desarrollo seleccionada (RUP) utiliza UML como lenguaje para la modelación, se decide utilizar el mismo como lenguaje de modelado del mecanismo de disponibilidad.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional), pero no especifica en sí mismo qué metodología o proceso usar.

2.2.4 Lenguaje de Programación: C++

Un lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina. **(Puerto Sorio, 2010)**

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales.

El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

Se decide utilizar C++ como lenguaje de programación para el desarrollo del mecanismo debido a:

- ✓ Es un lenguaje orientado a objetos.
- ✓ Es muy potente en lo que se refiere a creación de sistemas complejos debido a su robustez.
- ✓ Actualmente, puede compilar y ejecutar código de lenguaje C, ya que viene con bibliotecas incluidas para realizar esta labor.
- ✓ Es muy utilizado en el mundo por lo que existe abundante información sobre su uso.
- ✓ Existen muchos algoritmos cuyos pseudocódigos se encuentran ya desarrollados en C++, de manera que pueden tomarse y amoldarse a la solución deseada.

2.2.5 Biblioteca: ICE

ICE

ICE es un middleware, es decir, un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas[14] con soporte para C++, C#, Java, Python, Ruby, PHP, y Visual Basic. ICE proporciona herramientas, APIs, y soporte de bibliotecas para construir aplicaciones cliente/servidor orientadas a objetos. Los clientes y los servidores pueden escribirse en diferentes lenguajes de programación, pueden ejecutarse en distintos sistemas operativos y en distintas arquitecturas, y pueden comunicarse empleando diferentes tecnologías de red.

ICE no es más que una plataforma para desarrollo de aplicaciones de comunicación para Internet de alto rendimiento que incluye varias capas de servicios y plugins[15]. Mantiene una semántica orientada a objetos. Soporta múltiples interfaces. Es independiente de la máquina. Es independiente del lenguaje de programación. Es independiente de la implementación, es decir, el cliente no necesita conocer como el servidor implementa sus objetos. Es independiente del sistema operativo. Es independiente del protocolo de transporte empleado. **(Vallejo Fernández, 2006)**

14 Plataformas con distintos sistemas operativos, que usan distintos protocoles de red e incluso involucran distintos lenguajes de programación en la aplicación distribuida.

15 Módulo de hardware o software que añade una característica o servicio específico a un sistema más grande.

2.2.6 Arquitectura de ICE

Replicación

El principal objetivo de la replicación es el de proporcionar redundancia ejecutando el mismo servidor en distintas máquinas. Si en una de ellas se produce un error, el servidor permanece disponible en el resto. ICE soporta una forma limitada de replicación cuando un proxy[16] especifica múltiples direcciones para un objeto. El núcleo de ejecución de ICE selecciona una de esas direcciones de manera aleatoria para su intento de conexión inicial, e intenta contactar con el resto en caso de fallo. Por ejemplo, considere este proxy:

SimplePrinter:tcp -h server1 -p 10001:tcp -h server 2 -p 10002

El proxy indica que el objeto identificado por SimplePrinter está disponible utilizando dos direcciones TCP, una en la máquina server1 y otra en la máquina server2. Se supone que los administradores del sistema garantizan que el servidor está en ejecución tanto en la máquina server1 como en la máquina server2.

Grupos de réplica

Además de la replicación basada en proxies comentada en la sección anterior, ICE soporta una forma más útil de replicación conocida como grupos de réplica que requieren el uso de un servicio de localización.
Un grupo de réplica tiene un identificador único y consiste en un número determinado de adaptadores de objetos. Un adaptador de objetos puede ser miembro de al menos un grupo de réplica. Dicho adaptador se considera como un adaptador de objetos replicado.

Después de haber establecido un grupo de réplica, su identificador puede utilizarse como un proxy indirecto en lugar de un identificador de adaptador. Por ejemplo, un grupo de réplica identificado por PrinterAdapters puede utilizarse como un proxy de la siguiente forma:

SimplePrinter@PrinterAdapters

Un grupo de réplica es tratado por el servicio de localización como un adaptador de objetos virtual. El comportamiento del servicio de localización cuando resuelve un proxy indirecto que contiene el identificador de un grupo de réplica es un aspecto relacionado con la implementación Por ejemplo, el

[16] En redes de computadoras , un servidor proxy es un servidor (un ordenador o una aplicación) que actúa como intermediario en las solicitudes de los clientes en busca de recursos de otros servidores.

servicio de localización podría tomar la decisión de devolver las direcciones de todos los adaptadores de objetos del grupo, en cuyo caso el núcleo de ejecución de la parte del cliente podría seleccionar una de esas direcciones de manera aleatoria utilizando la forma limitada de replicación comentada en la anterior sección. Otra posibilidad para el servicio de localización sería la de devolver una única dirección seleccionada en función de una determinada heurística.

Sin tener en cuenta cómo el servicio de localización resuelve un grupo de réplica, el principal beneficio es la indirección: el servicio de localización puede añadir más inteligencia al proceso de resolución actuando como intermediario.

Propiedades

La mayor parte del núcleo de ejecución de ICE se puede configurar a través de las propiedades. Estos elementos son parejas clave-valor, como por ejemplo Ice.Default.Protocol=tcp.

Dichas propiedades están normalmente almacenadas en ficheros de texto y son trasladadas al núcleo de ejecución de ICE para configurar diversas opciones, como el nivel de traceado, y muchos otros parámetros de configuración.

2.2.7 Servicios ofrecidos por ICE

ICE maneja una serie de servicios, estos servicios se implementan como servidores ICE de forma que la aplicación desarrollada actúe como un cliente. Ninguno de estos servicios utilizan características internas a ICE ocultas a la aplicación que se desarrolle, por lo que en teoría es posible desarrollar servicios equivalentes por parte del desarrollador.

Sin embargo manteniendo estos servicios disponibles como parte de la plataforma permite al desarrollador centrarse en el desarrollo de la aplicación en lugar de construir la infraestructura necesaria en primer lugar. Además, la construcción de dichos servicios no es un esfuerzo trivial, por lo que es aconsejable conocer y usar los servicios disponibles en lugar de “reinventar la rueda” en cada aplicación.

Dentro de los servicios que ofrece ICE se pueden mencionar:

IceBox: IceBox es un único servidor de aplicaciones que permite gestionar el arranque y la parada de un determinado número de componentes de aplicación.

IceStorm: IceStorm es un servicio de publicación-subscripción que actúa como un distribuidor de eventos entre servidores y clientes.

IcePatch2: IcePatch2 es un servicio que permite la fácil distribución de actualizaciones de software a los clientes. Éstos simplemente han de conectarse al servidor IcePatch2 y solicitar actualizaciones para una determinada aplicación.

Glacier2: Glacier2 es el servicio de cortafuegos de ICE, el cual permite que tanto los clientes como los servidores se comuniquen de forma segura a través de un cortafuegos sin comprometer la seguridad.

Específicamente en el desarrollo del mecanismo que garantice monitorear la disponibilidad de los módulos del SCADA "UX" puede resultar de utilidad el uso del servicio IceGrid ya que brinda elementos como nodos, registros, réplicas y otros que servirían para la implementación del mecanismo en cuestión.

IceGrid

IceGrid es la implementación de un servicio de localización ICE. Tiene características relevantes para la implementación del mecanismo como son:

- IceGrid permite el registro de servidores para un arranque automático, es decir, habilita la activación de servidores bajo demanda (servidores que se activan cuando el cliente emite una solicitud).
- IceGrid soporta la replicación y el balanceado de carga.
- IceGrid proporciona un servicio que permite a los clientes obtener proxies para los objetos en los que están interesados.

ICE introduce una serie de conceptos técnicos que componen su propio vocabulario, como ocurre con cualquier nueva tecnología, por lo que es necesario que se entiendan los términos en IceGrid de: registro, réplica, nodo, adaptador y grupo de réplica detallado en secciones anteriores.

Adaptador

El adaptador de objetos es una parte de la API de ICE específico al lado del servidor: sólo los servidores utilizan los adaptadores de objetos. Un adaptador de objetos tiene varias funciones, como traducir las peticiones de los clientes a los métodos específicos al lenguaje de programación empleado, asociarse a uno o más puntos finales de transporte, o crear los proxies que pueden pasarse a los clientes.

Nodo

Se identifica a un computador físico del sistema. Esto no es del todo cierto, ya que un mismo computador puede tener varios nodos. Sin embargo, para el objetivo del trabajo de investigación es suficiente pensar que un nodo es un computador del grid.

Registro

Nodo del sistema que se encarga de la transparencia de localización y funciona como "DNS[17]" de todo el sistema. Por ello, debe conocer la estructura completa del sistema y de todos los recursos. Debe de ser único, puede haber otro registro en el sistema, pero con funciones de respaldo.

Réplica

Nodo del sistema que cumple las funciones de respaldo, es decir, es el registro esclavo que está en espera de que ocurra un fallo en la conexión con el registro máster para tomar su lugar y servir entonces de intermediario en la comunicación aplicación cliente y servidor.

2.2.8 Herramienta CASE[18]: Visual Paradigm

Una herramienta CASE es un conjunto de métodos, utilidades y técnicas que proporcionan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases. Las herramientas CASE engloban todos los pasos del desarrollo del software, y también aquellas actividades generales que se aplican a lo largo del proceso.

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Se decide utilizar Visual Paradigm como herramienta CASE debido a:

- ✓ Posee capacidades de ingeniería directa e inversa.

[17] Domain Name Server, servidor de nombre de dominio traducido al español.

[18] Siglas en inglés: Computer Aided Software Engineering (Ingeniería de Software Asistida por Ordenador).

- ✓ Posee disponibilidad de múltiples versiones para cada necesidad.
- ✓ Ofrece un diseño centrado en casos de uso y enfocado al negocio.
- ✓ Puede integrarse a los principales entornos de desarrollo.

2.2.9 Framework para el desarrollo de interfaces gráficas de usuarios: QT

QT es una biblioteca multiplataforma, que incluye aproximadamente 400 clases C++ de forma nativa, las cuales brindan las funcionalidades GUI[19], Base de Datos, XML[20], de red, Open GL, Multihilo; para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas como herramientas de la consola y servidores.

Con QT Creator, el IDE[21] necesario para trabajar con las librerías QT, se puede desarrollar grandes aplicaciones para Windows, Linux y Mac, además es gratuito.

QT Creator es un IDE (Entorno de Desarrollo Integrado) creado por Trolltech para el desarrollo de aplicaciones con las bibliotecas QT. Los sistemas operativos que soporta en forma oficial son: GNU/Linux. Además, hay una versión para Mac OS X, Windows XP y Vista.

Incluye un montón de herramientas (indentación[22] y resaltado del código, soporte de sesiones, múltiples proyectos, integración con servicios de control de versiones, diferentes configuraciones de compilación para un mismo proyecto y más).

Dentro de las características esenciales de QT Creator que la definen como herramienta de desarrollo del mecanismo se pueden listar las siguientes:

- Funciona bastante bien con cualquier proyecto basado en C++, esto incluye proyectos construidos con autotools.
- Indexación de 200.000 líneas de código que ocurre en alrededor de un minuto.

[19] Interfaz Gráfica de usuario.

[20] Siglas en inglés: eXtensible Markup Language (Lenguaje de marcado extensible). Diseñado especialmente para la web.

[21] Integrated Development Environment - Entorno integrado de desarrollo

[22] Espacio o sangría que se pone a la derecha de cada línea de código.

- Proporciona métodos rápidos para encontrar archivos, los símbolos, y ver donde los símbolos se usan.

2.3 Conclusiones del Capítulo

Orientado en el estudio de las diferentes herramientas y tecnologías existentes y su desarrollo actual, se define que es más recomendable utilizar las mencionadas y listadas en este capítulo por su singularidad de pertenecer a la categoría de software libre. Además, con el uso de las mismas se está siendo consecuente con las políticas trazadas por el Centro de Informática Industrial.

Capítulo 3: "Propuesta de la Solución"

3.1 Introducción

El presente capítulo tiene como objetivo dar a conocer las características principales del mecanismo a desarrollar, siguiendo detalladamente el problema por el cual fue concebido. Además, se especifican las funcionalidades que se implementarán para que el mecanismo cumpla con las especificaciones para el que fue desarrollado. También se describen los diferentes procesos que se llevan a cabo en el sistema, es decir el flujo normal de los eventos que ocurren en el mismo, así como se detalla información de diseño del mecanismo en cuestión y se da a conocer la propuesta de solución que se realiza que para el caso de la investigación es el desarrollo de una biblioteca que permita monitorear la disponibilidad de los módulos del configurador SCADA UX.

3.2 Descripción de las Funcionalidades

Partiendo de la situación problémica expuesta con anterioridad y el estudio realizado referente a la tecnología que se utiliza (ICE) en el desarrollo del mecanismo de monitoreo y control de la disponibilidad de los módulos, se definen como principales requisitos funcionales de la biblioteca los listados a continuación, que permitirán que está garantice redundancia en los servicios y balanceo de carga de los mismos, así como permitir conocer la disponibilidad de los servidores y a su vez llevar a cabo las acciones de inicio, reinicio y parada de los servicios:

Un requisito funcional define el comportamiento interno del software: cálculos, manipulación de datos y otras funcionalidades específicas. Son complementados por los requisitos no funcionales, que se enfocan en cambio en el diseño o la implementación.

Funcionalidad 1: Conocer grado de disponibilidad.

Funcionalidad 2: Permitir inicio, reinicio y parada de los servicios.

Funcionalidad 3: Permitir redundancia y balanceo de carga.

Para el desarrollo de la biblioteca los requisitos funcionales son analizados como funcionalidades ya que ésta no es una aplicación de escritorio con la que un usuario en particular interactúa, sino que se trata de un sistema que define las principales acciones necesarias para conocer disponibilidad de servidores, éstas son explicadas a continuación:

Funcionalidad 1: Conocer grado de disponibilidad.

Resumen:

Para que la biblioteca permita conocer el grado de disponibilidad de los servicios brindados para satisfacer peticiones de los clientes se lleva a cabo el proceso que se describe a continuación:

La biblioteca para conocer el estado de disponibilidad de los servidores utiliza información proveniente de la biblioteca IceGrid. El proceso consiste en una traducción de los estados devueltos por IceGrid en estados definidos por la biblioteca en cuestión. Esta traducción se debe a que los estados definidos por IceGrid son solo para los servidores ICE por lo que el estado devuelto para los servidores que no son de ICE no estará en correspondencia con el estado real de estos. En la siguiente tabla se muestra dicha traducción:

Estados IceGrid	**Estados Biblioteca de Disponibilidad**
Inactive	INACTIVE
Active, Activating or ActivationTimeout	ACTIVE
Deactivating	DEACTIVATING
Destroying	DESTROYING
Destroyed	DESTROYED
Estado por defecto : UNKNOW	

Tabla 1 Estados de IceGrid y Biblioteca de disponibilidad.

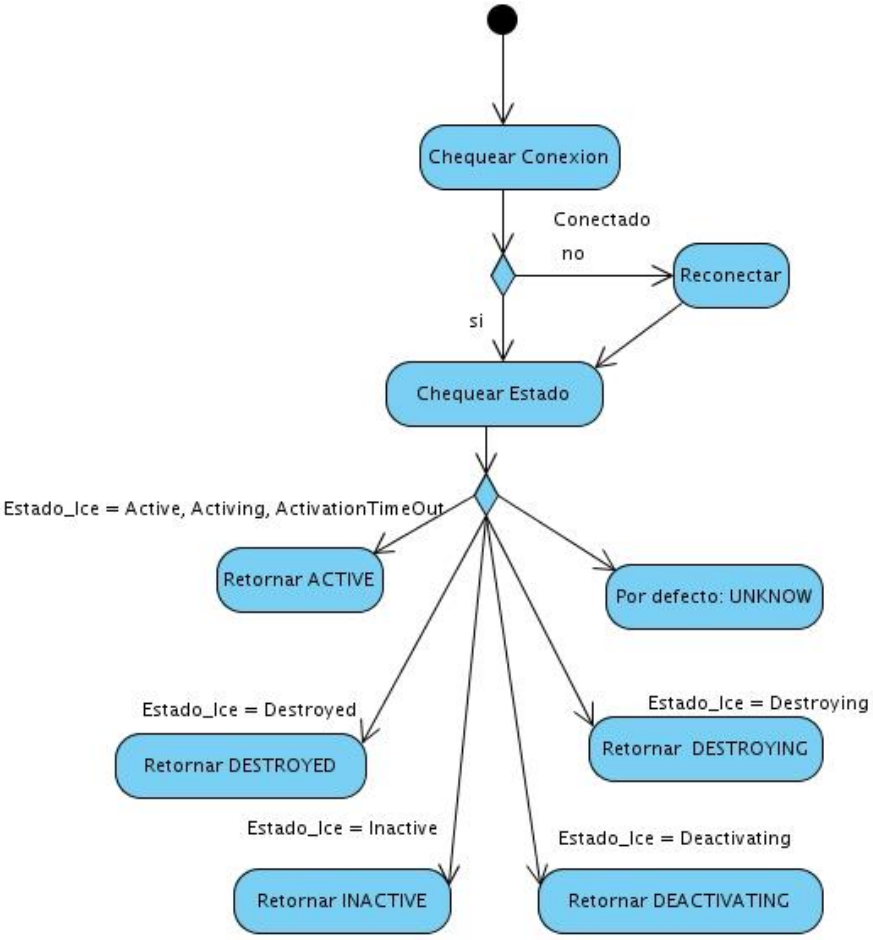


Ilustración 1 Diagrama del proceso Conocer grado de disponibilidad

Funcionalidad 2: Permitir inicio, reinicio y parada de los servicios.

Resumen:

Para el caso en que la biblioteca permita las acciones de inicio, reinicio y parada de servicios, estos procesos se llevan a cabo de forma independiente atendiendo a cuál es que solicita el cliente, aunque son procesos similares, con la diferencia de los estados a chequear para cada una las acciones.

En caso de que sea solicitada la acción de inicio primero se verifica que el servidor que se desea iniciar esté conectado al registro máster, en caso contrario se intenta conectar a otro registro esclavo que esté disponible, una vez conectado se procede a verificar el estado del servidor, si está activo entonces se termina el proceso y en caso contrario se procede a iniciar el servidor. Condicionalmente después de la inicialización del servidor pudiese ocurrir la captura de alguna excepción que para realizar la acción

correspondiente se chequea el estado del servidor, y si éste continúa en el mismo estado previamente analizado se lanza una excepción indicando la inexistencia del servidor.

Para el caso de que se solicite la acción de parada de los servicios que corren en el servidor se lleva un proceso similar al de inicio, con la salvedad de que el estado a verificar es que el servidor no esté inactivo o desactivándose.

Por último, para llevar a cabo la acción de reinicio, solo es necesario ejecutar las funciones de parada de servicios y luego el inicio de los mismos.

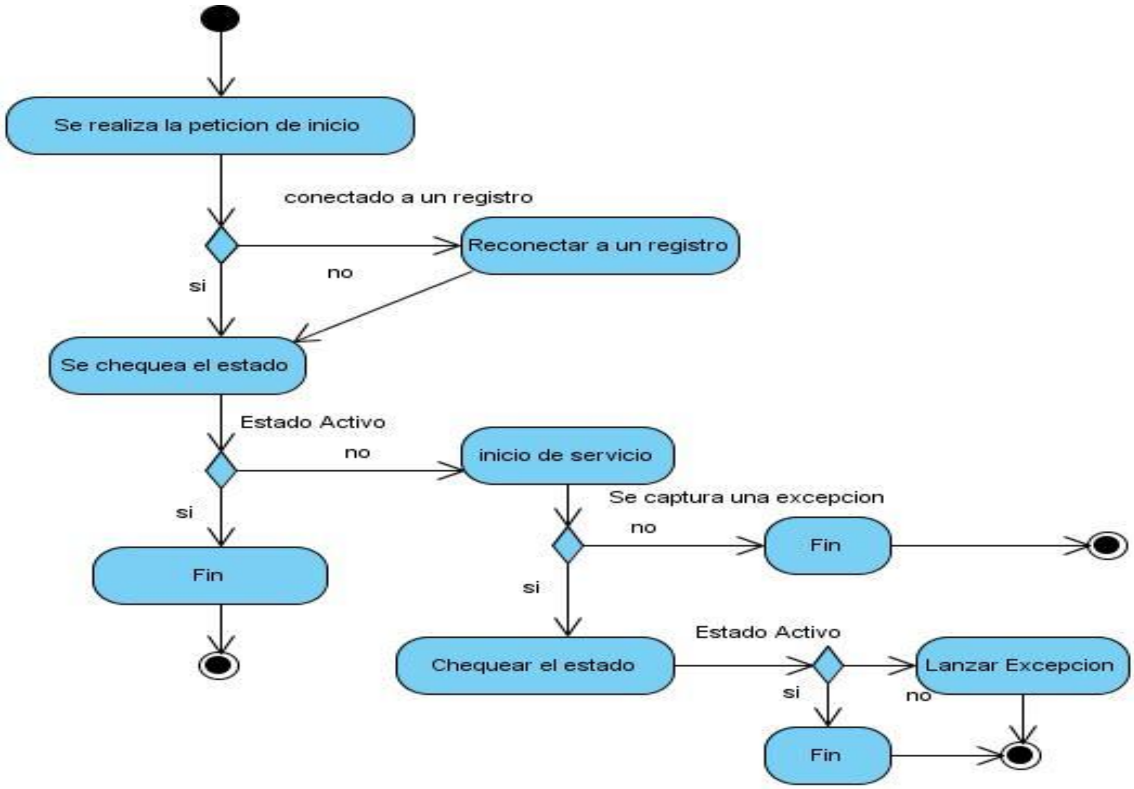


Ilustración 2 Diagrama del proceso Inicio de los servicios.

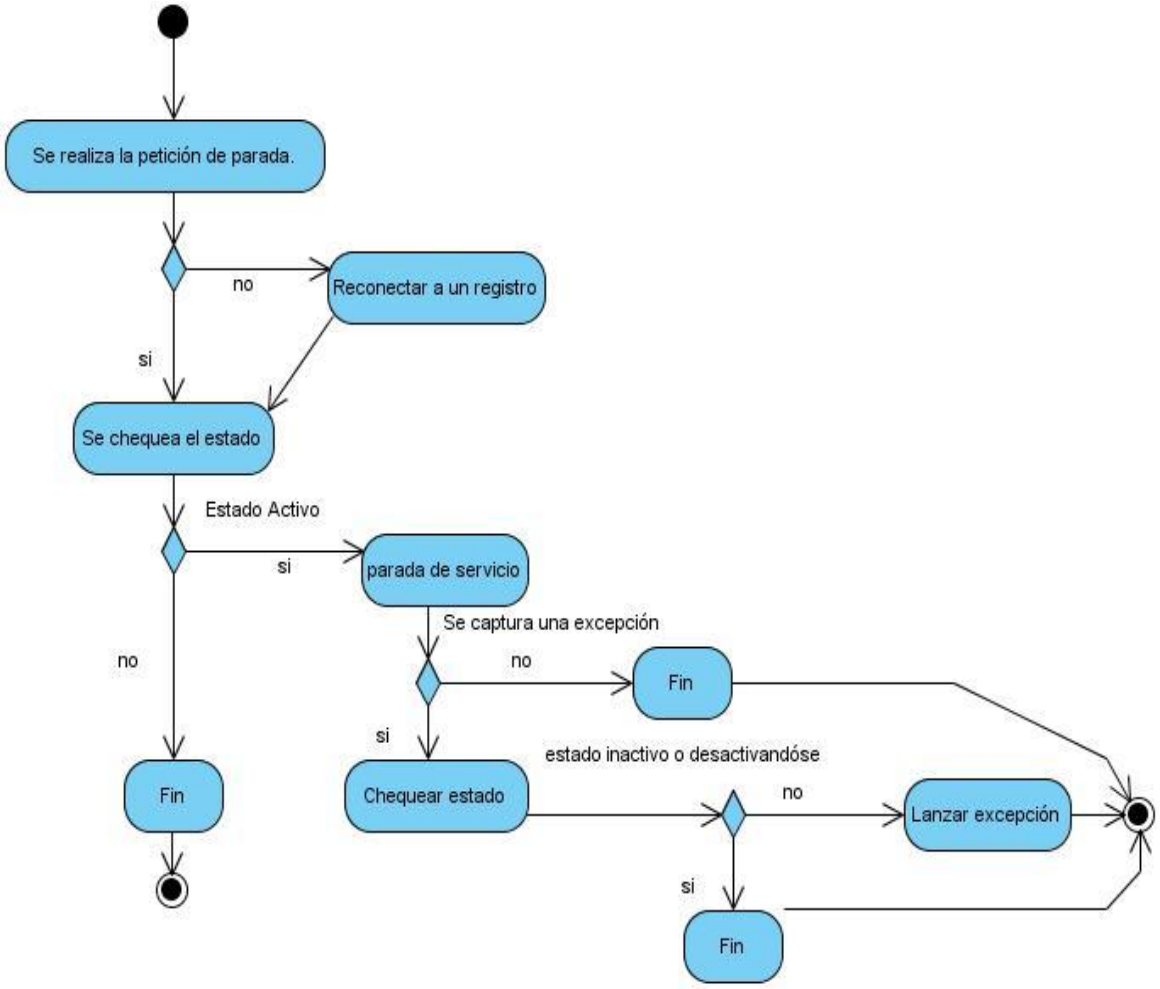


Ilustración 3 Diagrama del proceso Parada de los servicios

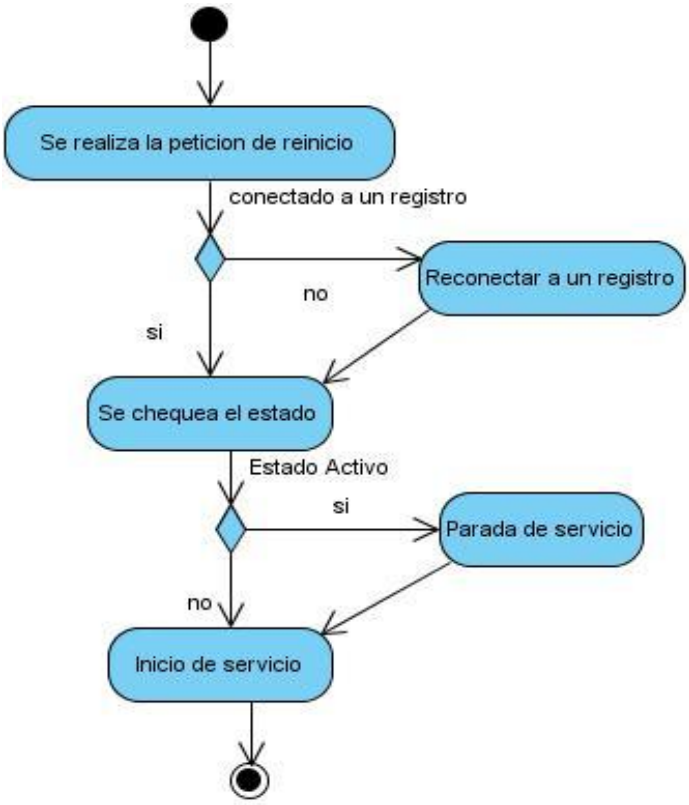


Ilustración 4 Diagrama del proceso Reinicio de los Servicios

Funcionalidad 3: Permitir redundancia y balanceo de carga.

Resumen:

La redundancia se implementa en la biblioteca haciendo uso de las configuraciones de los servidores y nodos donde estos últimos contienen uno o más servidores. Para la obtención de la redundancia los servicios existen en más de un servidor, es decir, existen múltiples servidores que brindan los mismos servicios.

Por otra parte el balanceo de carga se garantiza haciendo uso de los grupos de réplica. Los adaptadores de objetos que pertenecen a un mismo grupo de réplica son aquellos adaptadores que pertenecen a servidores que proveen los mismos servicios por lo que en la configuración del descriptor se les define un atributo *replica-group* indicando el nombre del grupo de réplica al que pertenece. Al grupo de réplica se le define un objeto *load-balancing* que tiene como atributo *type* para especificar el tipo de balanceo que utilizará dicho grupo de réplica. Entre los tipos de balanceo de carga posibles está *ordered*, *adaptive*, *round robin* y *random*. Para más detalles remitirse a la documentación oficial de ICE.

3.3 Requisitos No Funcionales

Las características o requisitos no funcionales (RNF) del sistema, son propiedades o cualidades que el producto debe tener. Son las características que hacen al producto atractivo, usable, rápido y confiable. Estos requisitos son muy importantes para que los clientes y operadores puedan valorar características no funcionales del software como: seguridad, confiabilidad y usabilidad.

RNF1. Software

El mecanismo debe ser compatible con sistema operativo GNU/Linux distribución Debian.

RNF2. Escalabilidad

El sistema debe estar en capacidad de permitir, en el futuro, el desarrollo de nuevas funcionalidades de disponibilidad después de su construcción y puesta en marcha inicial.

RNF3. Fiabilidad

Debe existir una comunicación segura y eficiente, garantizando no existan pérdidas de información ni acceso indebido de usuarios o aplicaciones.

RNF4. Soporte

El desarrollo del sistema debe brindar alta interoperabilidad para que aplicaciones del SCADA UX implementadas en diferentes lenguajes y sistemas operativos, puedan comunicarse de manera rápida y eficiente.

Prestar servicios de instalación, configuración y mantenimiento de la aplicación.

RNF6. Portabilidad

La solución adoptada debe poder ser implantada en cualquier sistema distribuido que necesite intercambiar información entre los subsistemas o módulos que lo componen.

3.4 Diseño de la biblioteca

En la etapa de diseño del sistema es cuando se traducen los requisitos funcionales y no funcionales en una representación de software, con el objetivo de producir un modelo o representación de una entidad

que se va a construir posteriormente, en el siguiente acápite se muestran los diagramas de componentes y de clases del diseño de la biblioteca de disponibilidad.

Diagrama de Componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes.

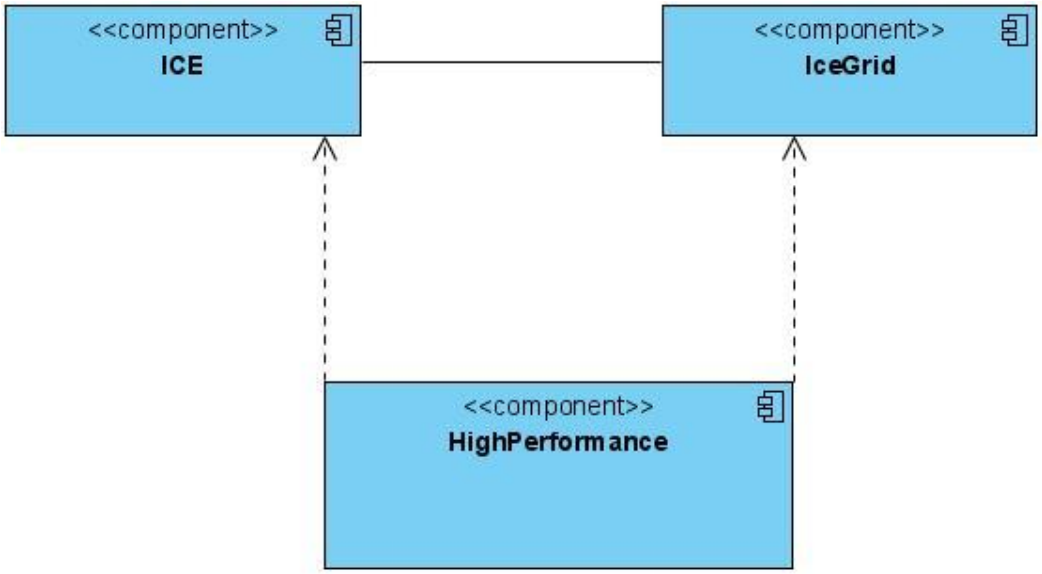


Ilustración 5 Diagrama General de Componentes.

En este diagrama se muestra la organización y las dependencias entre el conjunto de componentes externos que se utilizan para la biblioteca de disponibilidad (HighPerformance). En este caso la misma utiliza las bibliotecas de ICE y IceGrid.

No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realiza por partes. Cada diagrama describe un apartado del sistema. El siguiente diagrama muestra por tanto una vista interna de los componentes, o sea, las clases e interfaces que se utilizan por la biblioteca de disponibilidad, como son: núcleo de disponibilidad, manejador de sesión, manejador de excepciones y los componentes registro y estado de los servidores.

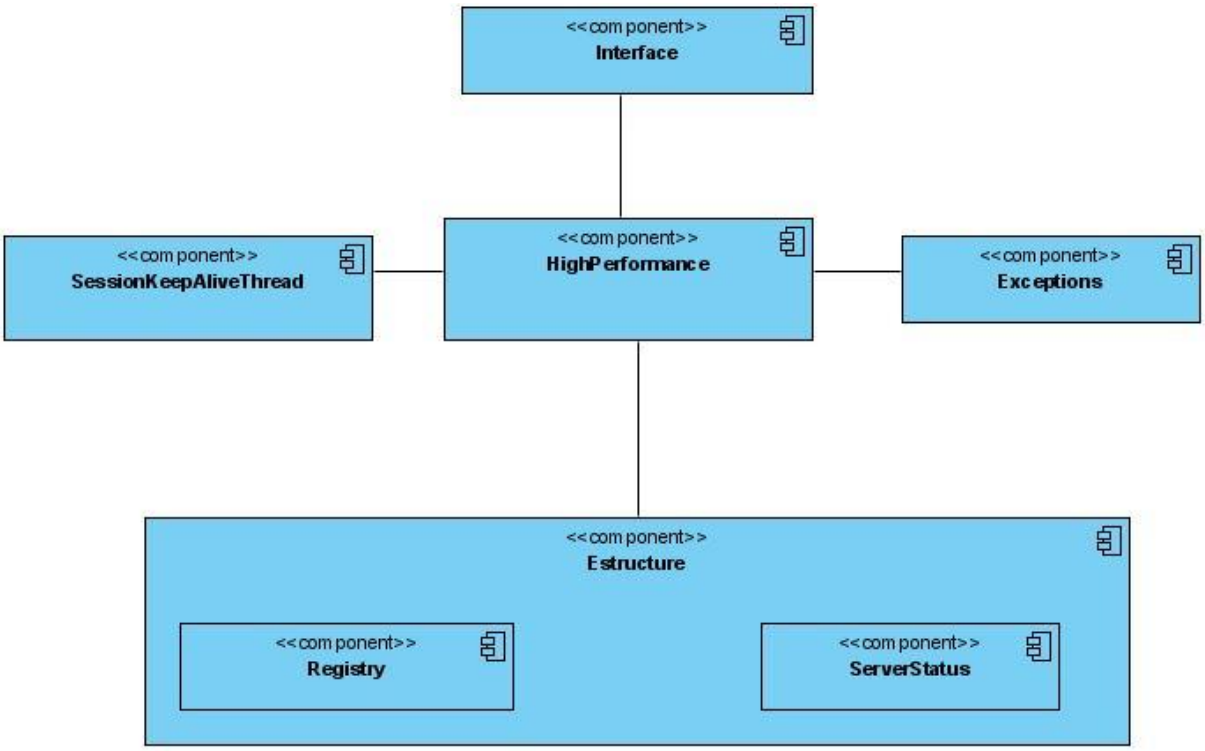


Ilustración 6 Diagrama Específico de Componentes.

Diagrama de Clases del diseño

El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. El siguiente diagrama muestra las clases definidas para la implementación de la biblioteca en cuestión y sus relaciones entre ellas. La clase principal HighPerformance (Alta disponibilidad) es la clase controladora que contiene las principales funcionalidades.

SessionKeepAliveThread (Mantenedor de sesiones): es la clase que se encarga de enviar pulsos de actividad a la clase principal para que la sesión abierta que permite administrar todos los servidores desde IceGrid no se cierre.

Registry: clase que contiene todos los atributos de un registro como: nombre, host, puerto y tiempo de conexión para los registros esclavos.

ServerStatus: es un enumerativo que contiene los estados definidos por la biblioteca para conocer el grado de disponibilidad de los servidores.

Para el caso de la clase **ConectionException** es la clase padre que maneja los posibles errores de conexión, que pueden ser errores de conexión al servidor, para lo cual se define la clase que hereda de esta **ConectionServerException** y la clase **ConectionRegistryException** que se usa para el manejo de los errores de tipo conexión al registro.

Y por último la clase **FileException** que se encarga de operar errores de archivo, archivo que contiene información de configuración del registro, y errores del tipo, por ejemplo al abrir archivo, leer archivo y errores en los datos almacenados en el archivo. Para ello se definen las clases **OpenFileExecption**, **ReadFileException** y **DataFileException** respectivamente que heredan de la clase **FileException**.

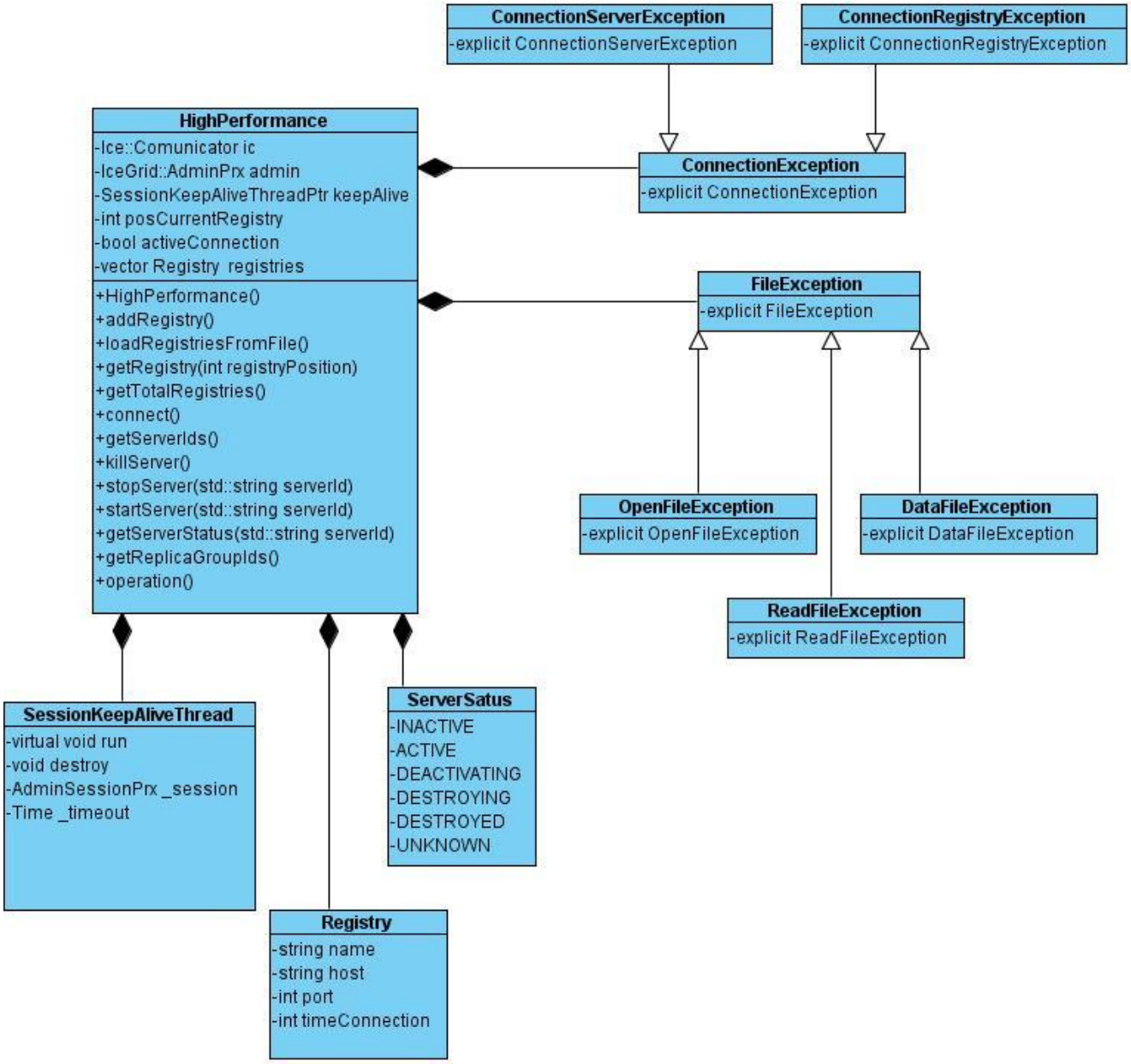


Ilustración 7 Diagrama de clases del diseño.

3.5 Arquitectura de la biblioteca de disponibilidad

La arquitectura en capas constituye una especialización de la arquitectura cliente-servidor, donde la carga se distribuye en partes con un reparto claro de las funciones y donde cada capa tiene relación con la siguiente. En el diseño de la biblioteca, se utiliza una arquitectura por paquetes, similar a la arquitectura en capas.

Esta arquitectura al igual que la arquitectura en capas simplifica la comprensión y la organización del desarrollo de sistemas complejos, reduciendo así las dependencias de forma que las capas más bajas no son conscientes de ningún detalle de las capas superiores. **(Páez Castillo, 2009)**

En el diseño de la biblioteca se utiliza la arquitectura en tres paquetes, ya que el modelo de diseño de la misma está organizado en los paquetes de Control, Estructura y Excepción como se muestra a continuación:

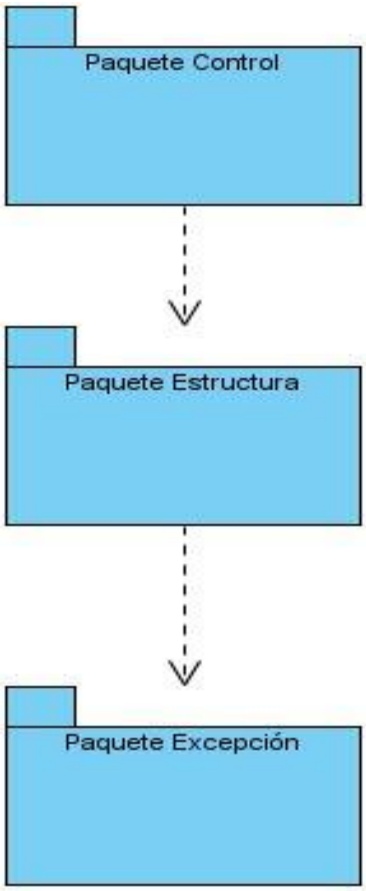


Ilustración 8 Modelo de Diseño de Paquetes de Biblioteca HighPerformance.

- **Paquete de Control**

En el paquete de Control se define e implementa la clase principal HighPerformance que proporciona los métodos necesarios para adicionar registros y conectarse a éstos, y a partir de ahí administrar los distintos servidores que existan en el registro, además posibilita que en caso de que falle un registro pueda conectarse a otro (en caso de existir), de forma transparente al usuario.

- **Paquete de Estructura**

El paquete de Estructura contiene las clases con información relevante y necesaria para el funcionamiento de la biblioteca de disponibilidad como son las clases: Registry, clase que permite almacenar la información de un registro, por ejemplo: nombre y puerto por el que se va a realizar la conexión. ServerStatus es la clase que contiene los distintos estados en los que se puede encontrar un servidor, algunos de los estados son: INACTIVE, ACTIVE. Y la clase SessionKeepAliveThread encargada de enviar pulsos de actividad al registro para que la sesión creada no sea cerrada.

- **Paquete de Excepción**

El objetivo del paquete de Excepción es almacenar las clases que manejan las excepciones en caso de ocurrir un error a la hora de realizar alguna operación de disponibilidad en la biblioteca. Éstas están resumidas en dos clases principales: FileException de la cual heredan OpenFileException, ReadFileException, DataFileException encargadas de manejar errores del archivo XML que tiene toda la configuración del registro, como pueden ser errores de datos incorrectos, o cuando se está leyendo el archivo. La otra clase principal es ConnectionException, encargada de manejar errores de conexión al registro o conexión al servidor, de la misma heredan: ConnectionServerException, y ConnectionRegistryException quienes manejan independientemente los errores antes mencionados.

3.6 Patrones

Un patrón es una solución a un problema en un contexto determinado, aplicando el conocimiento específico acumulado por la experiencia en un dominio, es decir, una solución a un problema de diseño que aparece con frecuencia. A continuación se explica cómo fueron utilizados en la biblioteca desarrollada.

3.6.1.1 Patrones GRASP

En diseño orientado a objetos, **GRASP** son patrones generales de software para asignación de responsabilidades. Es el acrónimo de "General Responsibility Assignment Software Patterns".

Aunque se considera que más que patrones propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

En la implementación de la biblioteca de disponibilidad se hace uso de los patrones listados a continuación:

- **Controlador**: El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. En la solución propuesta se llama interfaz a la biblioteca de forma general, específicamente a la clase HighPerformance. Muchas de las funcionalidades que existen en ésta, por ejemplo: startServer, son ejecutadas por la clase IceGrid con la salvedad de que en dependencia de los resultados HighPerformance devuelve el mismo de forma que se pueda realizar las operaciones para los servidores de ICE y para los que no lo son. Siendo la biblioteca la intermediaria entre la peticiones que recibe y la biblioteca IceGrid.
- **Alta cohesión y bajo acoplamiento**: Los conceptos de cohesión y acoplamiento están íntimamente relacionados. Un mayor grado de cohesión implica uno menor grado de acoplamiento. La información que almacenan cada una de las clases implícitas en la implementación de la biblioteca de disponibilidad son coherentes y están (en la medida de lo posible) relacionadas con la clase. Además las clases están lo menos ligadas entre sí de tal forma que en caso de producirse una modificación en alguna de ellas, tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases, para mejor entendimiento del uso del patrón dirigirse al diagrama de clases del diseño propuesto.

3.7 Conclusiones del Capítulo

En este capítulo se abordó las principales características de la biblioteca propuesta, el flujo principal de los procesos que se realizan por la misma para conocer grado de disponibilidad, así como realizar las acciones necesarias como son: inicio, reinicio y parada de los servicios según sea necesario. Además de ello se listan todos los requisitos del software ya sean funcionales (denominados en el caso particular como funcionalidades) o no funcionales.

Capítulo 4: "Implementación y Prueba"

4.1 Introducción

En este capítulo se explica el estándar de codificación y documentación utilizado para el desarrollo del código de la biblioteca de disponibilidad. También se explica mediante un diagrama de despliegue la distribución física del hardware utilizado y diferentes pruebas de sistema realizadas a la biblioteca así como una breve explicación del manual de uso realizado.

4.2 Vista General del manual de usuario

Para un mejor entendimiento, dominio y uso de la biblioteca desarrollada, se realizó un manual de usuario o manual de uso, el cual aborda lo referido a la definición de los ficheros donde se especifica la configuración de los registros, réplicas y nodos. Además, se explica el contenido del XML[23] en el cual se definen los elementos que contendrá el registro, haciendo énfasis en lo que más probablemente sea necesario usar en la biblioteca. Igualmente se expone cómo se inician el registro, las réplicas y los nodos, así cómo se pueden adicionar elementos al registro desde un XML, actualizar esta información o eliminarla. Por último, se especifica las características del fichero desde el cual el módulo lee la información del lugar donde se encuentran ubicados el registro y la(s) réplica(s).

A continuación se explica brevemente las configuraciones de nodos, registros y réplicas.

Especificación de archivo de configuración del registro de IceGrid

En este archivo se hace necesario especificar los siguientes parámetros:

Ice.Override.ConnectTimeout = [Tiempo en milisegundos]

Ice.Override.Timeout = [Tiempo en milisegundos]

IceGrid.Registry.Client.Endpoints = [Protocolo de comunicación] -p [Puerto]

IceGrid.Registry.Server.Endpoints = [Protocolo de comunicación]

IceGrid.Registry.Internal.Endpoints = [Protocolo de comunicación]

[23] **Extensible Markup Language** o Lenguaje de marcado extendible. Norma recomendada para definir nuevos tipos de documentos, permitiendo al usuario definir sus propias etiquetas de marcado para extender las capacidades del HTML.

IceGrid.Registry.AdminPermissionsVerifier = IceGrid/NullPermissionsVerifier

IceGrid.Registry.PermissionsVerifier = IceGrid/NullPermissionsVerifier

IceGrid.Registry.Data = [Dirección donde se creará la base de datos del registro]

Ice.Override.ConnectTimeout: Tiempo que esperará el registro para tratar de hacer una conexión.

Ice.Override.Timeout: Tiempo que esperará el registro para tratar de recibir respuesta de un pedido.

IceGrid.Registry.Client.Endpoints: En este parámetro se especifica el protocolo y el puerto que se usarán en la comunicación con los clientes.

IceGrid.Registry.Server.Endpoints: Parámetro que especifica el protocolo que se usará en la comunicación con los servidores.

IceGrid.Registry.Internal.Endpoints: Parámetro donde especifica el protocolo que se usará en la comunicación interna.

IceGrid.Registry.AdminPermissionsVerifier: Este parámetro se usa para especificar que no se requerirá loguearse para crear sesiones de administración.

IceGrid.Registry.PermissionsVerifier: Este parámetro se usa para especificar que no se requerirá loguearse para crear sesiones.

IceGrid.Registry.Data: Parámetro con el cual se especifica la dirección donde se guardará la base de datos del registro.

Ver ilustración 17 de ejemplo en los anexos.

Inicialización del registro

Para iniciar el registro hay que ejecutar: icegridregistry --Ice.Config = [Dirección donde se encuentra la configuración del registro].

- ✓ Ejemplo

user@nautilus:~/Desktop/Prueba$ icegridregistry --Ice.Config=Registry.conf

En este caso se está ubicado en la misma dirección donde se encuentra el archivo de configuración del registro cuyo nombre es Registry.conf.

Especificación de archivo de configuración de la réplica de IceGrid

En este archivo se hace necesario especificar los siguientes parámetros:

Ice.Override.ConnectTimeout = [Tiempo en milisegundos]

Ice.Override.Timeout = [Tiempo en milisegundos]

Ice.Default.Locator=IceGrid/Locator:[Protocolo de comunicación] -h [host] -p [Puerto]

IceGrid.Registry.Client.Endpoints=[Protocolo de comunicación] -p [Puerto]

IceGrid.Registry.Server.Endpoints=[Protocolo de comunicación]

IceGrid.Registry.Internal.Endpoints=[Protocolo de comunicación]

IceGrid.Registry.AdminPermissionsVerifier=IceGrid/NullPermissionsVerifier

IceGrid.Registry.PermissionsVerifier=IceGrid/NullPermissionsVerifier

IceGrid.Registry.Data=[Dirección donde se creará la base de datos de la réplica]

IceGrid.Registry.ReplicaName=[Nombre de la réplica]

Ice.Override.ConnectTimeout: Tiempo que esperará el registro para tratar de hacer una conexión.

Ice.Override.Timeout: Tiempo que esperará el registro para tratar de recibir respuesta de un pedido.

Ice.Default.Locator: En este parámetro se especifica el protocolo que se usará en la comunicación con el Máster, la dirección donde se encuentra el máster y el puerto que se usará en la comunicación.

IceGrid.Registry.Client.Endpoints: En este parámetro se especifica el protocolo y el puerto que se usarán en la comunicación con los clientes.

IceGrid.Registry.Server.Endpoints: Parámetro que especifica el protocolo que se usará en la comunicación con los servidores.

IceGrid.Registry.Internal.Endpoints: Parámetro donde especifica el protocolo que se usará en la comunicación interna.

IceGrid.Registry.AdminPermissionsVerifier: Este parámetro se usa para especificar que no se requerirá loguearse para crear sesiones de administración.

IceGrid.Registry.PermissionsVerifier: Este parámetro se usa para especificar que no se requerirá loguearse para crear sesiones.

IceGrid.Registry.Data: Parámetro con el cual se especifica la dirección donde se guardará la base de datos del registro.

IceGrid.Registry.ReplicaName: Parámetro usado para especificar el nombre de la Réplica. Ver ilustración 18 de ejemplo en los anexos.

Inicialización de la réplica

Para iniciar una réplica se realiza lo mismo que para iniciar un registro con la única diferencia (como es lógico) de que habría que especificar la dirección del archivo donde se encuentra la configuración de la réplica. Para poder realizar esta acción se hace necesario que el registro haya sido iniciado.

- ✓ Ejemplo

user@nautilus:~/Desktop/Prueba$ icegridregistry --Ice.Config=Replica1.conf.

En este caso se está ubicado en la misma dirección donde se encuentra el archivo de configuración de la réplica cuyo nombre es Replica1.conf.

Especificación de archivo de configuración del nodo de IceGrid

En este archivo se hace necesario especificar los siguientes parámetros:

IceGrid.Node.Name = [Nombre del nodo]

IceGrid.Node.Endpoints = [Protocolo de comunicación]

IceGrid.Node.WaitTime = 0

IceGrid.Node.Data = [Dirección donde se creará la base de datos del nodo]

Ice.Default.Locator = IceGrid/Locator**(**:[Protocolo de comunicación] -h [Host] -p [Puerto]**)**

IceGrid.Node.Name: Parámetro donde se especifica el nombre del nodo.

IceGrid.NodeEndpoints: En este parámetro se especifica el protocolo que se usará en la comunicación con los servidores.

IceGrid.Node.WaitTime: Parámetro que define el tiempo que esperará el nodo para activar o desactivar un servidor. Se pone 0 para que no se espere para detener un servidor que no sea de Ice, y se detenga apenas llega la orden.

IceGrid.Node.Data: Parámetro donde se define la dirección donde se guardará la base de datos del nodo.

Ice.Default.Locator: Parámetro donde se especifica el protocolo que se usará en la comunicación con el o los registros, la dirección donde se encuentran estos y el puerto que se usará en la comunicación.

Ver ilustración 19 de ejemplo en los anexos.

Inicialización del nodo

Para iniciar un nodo hay que ejecutar: icegridnode --Ice.Config = [Dirección donde se encuentra la configuración del nodo].

- ✓ Ejemplo

user@nautilus:~/Desktop/Prueba$ icegridnode –Ice.Config=Node1.conf

En este caso la ubicación es en la misma dirección donde se encuentra el archivo de configuración del nodo cuyo nombre es Node1.conf.

Para más información relacionada por ejemplo con: especificación del XML donde se realiza la descripción de todos los elementos que contendrá el registro; actualizar información de las aplicaciones del registro, entre otros, se puede consultar el manual oficial de uso realizado para la biblioteca adjunto como anexo y además, puede dirigirse a la documentación oficial de Ice. La versión de Ice usada en la biblioteca es la 3.3.1, que se encuentra en los repositorios de Ubuntu.

4.3 Estándar de Codificación

Es prudente establecer estándares de codificación para todos los programadores. Estos estándares consisten en estilos de codificación a la hora de escribir el código. Los aspectos para los que generalmente se establecen estándares son los siguientes:

- ✓ Identificadores
- ✓ Indentación
- ✓ Líneas y espacios en blancos

- ✓ Comentarios

En cada grupo de desarrollo se definen cuáles serán los aspectos a estandarizar y qué estilos se aplicarán a cada uno de ellos. El cumplimiento de estándares permite que todo el código tenga el sello personal del programador y en caso de que sean varios los programadores, pues se busca que todo el código parezca que ha sido implementado por la misma persona. De esta manera, se consigue mayor legibilidad y facilidad de mantenimiento. A continuación se definirá cuáles estándares usar para la implementación correspondiente a este trabajo. Están agrupados en tres conjuntos: Nombres, Manejo de errores y Codificación.

- Nombres

  - Los nombres de las clases son sustantivos singulares.
  - Los nombres deben reflejar el qué y no el cómo.
  - Los nombres no deben revelar detalles de implantación.
  - Escoger nombres lo suficientemente largos para ser expresivos, pero evitando manejar nombres que dificulten la labor de implantación.
  - Evitar nombre que permitan una interpretación subjetiva (evitar ambigüedad y asegurar abstracción).
  - Evitar redundancia no repitiendo nombre de clases en sus elementos.
  - Dado que los nombres generalmente son el producto de concatenar varias palabras, se debe emplear mayúscula para el inicio de cada palabra y minúscula para el resto de las letras para el caso de los nombres de métodos y funciones. Para el caso de los nombres de variables y atributos debe aplicarse la misma convención, con excepción de la primera letra del nombre, la cual debe ser en minúscula.
  - Los nombres de constantes deben contener solo letras mayúsculas.
  - Minimizar el uso de abreviaciones. En caso de ser requeridas, se debe ser consistente en su uso y cada abreviación debe significar solo una cosa. En general agregar a la documentación las abreviaturas.
  - Los nombres de los métodos son frases que incluyen verbos.
  - Los nombres de los atributos y parámetros son frases con sustantivos.
  - Evitar el reúso de nombres para distintos propósitos.

- Manejo de Errores

- Se pueden manejar los errores mediante mecanismos de excepciones o mediante valores de retorno, aunque esto debe ser uniforme dentro de un mismo objeto.
- Es buena práctica emplear herramientas para identificar errores en la codificación en caliente.

➢ Codificación

- Se establece un tamaño de indentación estándar de tres (3) espacios, sin tabulaciones.
- Alinear secciones del código.
- Alinear verticalmente llaves de apertura y cierre.
- Usar espacios antes y después de los operadores que el lenguaje de programación permita.
- Emplear líneas en blanco para organizar el código, permitiendo crear párrafos de código para una mejor lectura.
- Evitar colocar más de una sentencia por línea.
- Emplear constantes en sustitución de números o cadenas de caracteres literales.
- Minimizar el alcance de las variables para evitar confusión y facilitar el mantenimiento.
- Emplear cada variable y rutina solo para un propósito.
- Evitar el uso de variables públicas, sustituirlas por variables privadas y métodos que provean el valor de tal variable, para mantener el encapsulamiento.
- Minimizar el uso de conversiones de tipo forzadas (castings), cuando se requiera su uso, debe ser comentada la justificación.
- Emplear select-case o switch en sustitución de if anidados sobre la misma variables.
- Liberar apuntadores de manera explícita.
- Emplear i, j, k, l, p, q, r para contadores en ciclos.
- Comentar siempre las llaves que cierran.
- Emplear al máximo operadores del tipo: +=, *=, /=, -=, ++, --, entre otros.
- Mantener la modularidad del código bajo el criterio de la lógica que encierra, no exagerar la modularidad.
- Emplear correctamente los tipos de ciclos: si es al menos una vez usar do-while, si es ninguna o más veces usar while-do, y si se conoce el número exacto de ciclos usar for.
- Inicializar todas las variables.
- Emplear líneas en blanco para separar pasos lógicos (declaraciones, lazos).
- Siempre asignar NULL a los apuntadores luego de ser destruidos (solo aplica para C).

4.4 Estándar de documentación

Se utilizará la herramienta de auto documentación Doxygen para generar la documentación del código involucrado en el desarrollo de la biblioteca y los formatos brindados por esta herramienta los cuales se explican a continuación.

- Estilo de bloques de documentación JavaDoc.

Se adopta el estilo de bloques de documentación JavaDoc, el cual consiste de un bloque de comentario de estilo C, este bloque de comentario comienza con dos asteriscos de esta manera:

```
/**
 * Texto
 */
```

En este caso los asteriscos que se encuentran en la línea de la mitad son opcionales, por lo que también es válido que el bloque sea de esta manera:

```
/**
  Texto
 */
```

- Descripción Breve con comando \brief ó @brief.

Para hacer una descripción breve se adopta el uso del comando \brief ó @brief en el bloque de comentarios ya descrito. La acción de este comando termina al final de un párrafo, de tal manera que la descripción detallada sigue después de una línea vacía. Aquí tenemos un ejemplo:

```
/**
 * @brief descripción breve.
 * Continuación de la descripción breve.
 *
 * La descripción detallada comienza aquí, nótese
 * que se debe dejar una línea en blanco para lograr
 * tener las dos descripciones (breve y detallada)
*/
```

Nota: si no se utiliza el comando @brief Doxygen tomará la descripción hecha en el bloque como una descripción detallada y no habrá descripción breve.

- Descripción de Argumentos y Métodos.

A la hora de hacer una descripción de los argumentos de métodos y funciones esta se hará en línea, es decir, luego de la declaración de cada uno de los argumentos se da una breve descripción de cada uno de ellos, en el siguiente ejemplo se muestra el formato a utilizar:

```
Int Function (int a, /**<Primer operando de la operación*/
                int b, /** Segundo operando de la operación*/
                int c /**<Tercer operando de la operación*/);

/**
 * @brief Breve descripción del método
 * @param a Descripción del parámetro a
 * @param b Descripción del parámetro b
 * @param c Descripción del parámetro c
 * @return Retorno del método
 */
int Function ( int a, int b, int c);
```

- Documentación de tipos de datos

Para la documentación de tipos de datos definidos tales como enumerados, uniones o typedefs se pueden utilizar los formatos especificados en los apartados anteriores para describir su función y los elementos que los componen, aquí se muestra un ejemplo:

```
/**
 * @brief Enumerado de los tipos de datos admitidos
 *
 * Descripción más detallada de la función de este tipo de dato.
 */
enum DataTypes{ INTEGER, /**<Puede ser un valor de tipo entero */
                DOUBLE, /**<Puede ser un valor de tipo double */
                STRING /**<Puede ser un valor de tipo string */ };
```

Para este ejemplo Doxygen generará la siguiente salida de documentación:

Error: Macro Image (Imagen1.jpg, 100%) failed
sequence item 0: expected string, None Type found

- Formato Doxygen

El formato a utilizar a la hora de documentar el código hará que Doxygen genere una salida de documentación como se muestra en el siguiente ejemplo:

```
/**
 * @brief Breve descripción de esta función
 *
 * Aquí comienza la descripción detallada de esta función,
 * también se muestra una explicación del valor de retorno de la
 * función si es el caso para esto se utiliza el comando \@return
 * @return Devuelve un valor entero resultado de la operación
 */
int Function ( int a , /**<Primer operando de la operación */
                       int b , /**<Segundo operando de la operación */
                       int c /**<Tercer operando de la operación */ );
```

Error: Macro Image (Imagen2.jpg, 100%) failed
sequence item 0: expected string, None Type found

Se observa que se genera una descripción breve seguida de una descripción más detallada para la función, luego se explica brevemente el valor de retorno de la misma y la descripción de los parámetros usando el formato de documentación en línea.

- Comando @code y @endcode

En algunos casos es necesario mostrar en la documentación una parte del código cuando se hacen comentarios entre líneas o para mostrar algún método en particular, ya que esto puede ahorrar hacer una explicación más extensa; para realizar esto en Doxygen se utiliza el comando @code y finaliza la acción del mismo con el comando @endcode, en el siguiente ejemplo se muestra lo planteado:

```
/**
 * @brief Devuelve el contador
 *
 * Se detalla la utilidad de esta función, el código que se muestra a continuación se incluye haciendo
 * uso del comando \@code y finalizando el bloque de código con el comando \@endcode, de esta
 * manera:
 * @code
 * int getCount()

 * {
 *                 return count;
 * }
 * @endcode
 */
```

```
int getCount()
{
    return count;

}
```

Doxygen genera la siguiente salida:

Error: Macro Image(Imagen3.jpg, 100%) failed
sequence item 0: expected string, None Type found

- Comandos @author y @date

Es importante que se especifique el nombre del autor y la fecha de creación de cualquier estructura en un código, para ello se utilizan los comandos @author y @date para el nombre del autor y la fecha respectivamente, en el siguiente ejemplo se puede ver la acción de estos comandos:

```
/**@brief Descripción breve
 *
 * Aquí comienza la descripción detallada
 * @author Alejandro Rubinos amrubinos@uci.cu
 * @date 15-02-2010
 */
void voidFunction ();
```

Para este caso Doxygen genera lo siguiente:
Error: Macro Image(Imagen5.jpg, 100%) failed
sequence item 0: expected string, None Type found

- Comando @see

Existe otro comando útil que permite realizar referencias a otras clases o métodos cuando sea necesario, es importante usar este comando en la documentación a la hora de implantar los métodos o funciones propias de alguna clase, este comando es @see y su uso se muestra en el siguiente ejemplo:

```
/**
* Constructor de la clase
* @see Test::Test()
*/
Test2();
```

Esto generará en la documentación para el constructor de la clase de prueba Test2 una referencia hacia la documentación existente para el constructor de la clase de prueba Test, la salida de Doxygen es la siguiente:

Error: Macro Image(Imagen6.jpg, 100%) failed
sequence item 0: expected string, None Type found

Nota: si se desea realizar una referencia desde cualquier estructura hacia la documentación completa de una clase ya documentada solo se debe utilizar el comando @see seguido del nombre de la clase de esta forma:

```
/**
* Constructor de la clase
* @see Clase
*/
Test2();
```

La salida generada por Doxygen sería algo como esto:

Error: Macro Image(Imagen7.jpg, 100%) failed
sequence item 0: expected string, None Type found

4.5 Pruebas

La prueba del software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Es una actividad en la cual un sistema o componente es ejecutado bajo condiciones o requerimientos específicos, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.

4.5.1 Tipos de Prueba

La calidad de un sistema está determinada, entre otras cosas, por la coincidencia entre lo que se programó y los requisitos establecidos en la primera fase. Para comprobar el grado de cumplimiento de estos requisitos se usan las pruebas del sistema. La prueba de unidad es la prueba enfocada a los elementos probables más pequeños del software, consiste en una prueba estructural (o caja blanca), lo cual requiere conocer el diseño interno del componente puesto que verifica la lógica interna y el flujo de datos; y una prueba de especificación (de caja negra), basada sólo en la especificación del comportamiento externamente visible del componente.

Estas definen un conjunto amplio de acciones de comprobación que abarcan todas las características que determinan la calidad de un software. Se comprueban las funcionalidades diseñando casos de prueba que definen cómo proceder. Estos casos de prueba incluyen los juegos de datos a usar que son los válidos o

esperados y los no válidos o no esperados por el sistema. Las pruebas se deben aplicar durante todo el ciclo de vida del software e invariablemente se le debe dedicar una gran parte del esfuerzo total del desarrollo.

El objetivo de los casos de prueba es forzar al máximo el sistema en los puntos críticos para encontrar fallos y detectar defectos. Se debe escoger los tipos de prueba que se adapten mejor al sistema que se va a probar. Para esto se debe tener en cuenta el lenguaje de programación, el proceso de desarrollo, las características de los desarrolladores, el tipo de funcionalidad que se implementa, la plataforma en que se ejecutan los procesos, los errores más importantes, si la aplicación es de escritorio o web, si realiza conexiones a bases de datos, entre otras observaciones.

Los diferentes niveles de pruebas son:

- Nivel de unidad: Está enfocada al código fuente de los componentes y verifica todos los flujos de control. Este tipo de prueba pasa primero por la revisión del programador.
- Nivel de integración: Prueba los componentes combinados para ejecutar casos de uso. Tiene como objetivo descubrir errores en las especificaciones de las interfaces de las clases.
- Nivel de sistema: Prueba el software funcionando como un todo. Aceptable cuando el software se encuentra en fase de construcción.
- Nivel de aceptación: Es la prueba final antes del despliegue del sistema. Generalmente las realizan los usuarios finales y es llamada prueba piloto.

4.5.2 Pruebas de Caja Blanca

Las pruebas de Caja Blanca se nombran de esta forma porque revisan la parte interna del software, específicamente sobre el código fuente. Se basan en el examen minucioso de los detalles procedimentales. Se comprueban los caminos lógicos del sistema generando casos de prueba que ejerciten las estructuras condicionales y los bucles. Existen varios métodos que analizan diferentes partes del programa y se complementan entre sí para garantizar la calidad del sistema. La técnica del camino básico se utiliza para comprobar la complejidad lógica de un diseño procedimental, permite diseñar casos de prueba para cubrir todas las sentencias de un programa a partir de la obtención de un conjunto de caminos independientes. La prueba de los Bucles se centra en la validez de las estructuras cíclicas o bucles. El objetivo es probar el comportamiento de estas estructuras en sus valores límites de iteración. Los bucles se clasifican en cuatro tipos: Bucles simples, Bucles anidados, Bucles concatenados y Bucles no estructurados. Aunque la esencia es la misma, cada tipo se prueba de forma diferente. De lo anterior

pudiera pensarse que las pruebas de Caja Blanca logran enmendar todos los errores del programa. La desventaja de estas es entonces de tipo logística ya que resulta imposible abarcar todo el código fuente de un sistema medianamente grande. El tiempo necesario para realizarlas sería considerable y se torna compleja su aplicación sobre algoritmos críticos.

4.5.3 Pruebas de Caja Negra

Las Pruebas de Caja Negra deben su nombre a los elementos que estas revisan y las condiciones en que se hace la revisión. Estas se basan en los requerimientos funcionales del sistema y se llevan a cabo desde el exterior de la aplicación. Este tipo de prueba es importante a la hora de medir el grado de cumplimiento de los requerimientos solicitados por el cliente y se aplican sobre la interfaz de la aplicación observando las respuestas del sistema antes determinadas acciones y los datos de salida para determinados datos de entrada.

Después de un análisis de los diferentes métodos de pruebas que existen y la función de cada uno de ellos se ha llegado a la conclusión de aplicar las pruebas de Caja Negra por lo que fue el método seleccionado para comprobar la validez en las respuestas de la biblioteca.

4.5.4 Diseño de Casos de Prueba

El objetivo fundamental del diseño de casos de prueba es conseguir un conjunto de pruebas que tengan la mayor probabilidad de encontrar los defectos del software. Un caso de prueba es un conjunto de entradas de pruebas, condiciones de ejecución y resultados esperados desarrollados para verificar una función esperada. Estas pruebas serán realizadas a un demo desarrollado para comprobar las funcionalidades de la biblioteca de disponibilidad en cuestión.

4.5.4.1 Descripción del caso de prueba Conocer grado de disponibilidad

Nombre de la funcionalidad	Descripción General	Escenarios de Pruebas	Flujo del escenario
Conocer grado de disponibilidad	La biblioteca debe permitir conocer en cada caso el grado de disponibilidad de todos los servicios.	**EP 1.1**: Servidor encontrado.	• Se chequea si se está conectado a un registro. • Se solicita estado del servidor solicitado. • Se retorna estado del servidor.
		EP 1.2: Servidor no encontrado.	• Se chequea si se está conectado a un registro. • Se solicita estado del servidor solicitado. • La biblioteca devuelve estado UNKNOW por defecto porque no existe el servidor solicitado.

Tabla 2 Descripción del caso de prueba Conocer grado de disponibilidad.

Id del Escenario	Escenario	Descripción	Respuesta del sistema	Resultado de la prueba
EP 1.1	Servidor encontrado.	Disponibilidad del servidor.	Se espera que la biblioteca devuelva el estado de disponibilidad del servidor, ejemplo: ACTIVE.	La biblioteca muestra el estado de disponibilidad del servidor solicitado.
EP 1.2	Servidor no encontrado.	Disponibilidad del servidor.	Se espera que la biblioteca devuelva por defecto el estado UNKNOW.	La bilioteca muestra el estado UNKNOW para el servidor solicitado.

Ilustración 9 Resumen de los escenarios para Conocer grado de disponibilidad.

4.5.4.2 Descripción del caso de prueba "Inicio" y "Parada de los servicios"

En este caso se detallará la funcionalidad en dos acápites, es decir, se describirá en dos casos de prueba, el primero para el inicio y el otro para la parada de servicios respectivamente.

Nombre de la funcionalidad	**Descripción General**	**Escenarios de Prueba**	**Flujo del escenario**
Inicio de servicios	La biblioteca debe permitir el inicio del servidor que posee los servicios solicitados.	**EP 1.1:** Servidor solicitado ya está activo.	• Se realiza la petición de inicio. • Se chequea si se está conectado a un registro. • Se chequea estado del servidor. • Si el estado del servidor es ACTIVE la biblioteca termina la funcionalidad porque ya está iniciado el servicio.
		EP 1.2: Servidor solicitado no está activo.	• Se realiza la petición de inicio • Se chequea si se está conectado a un registro. • Se chequea estado del servidor. • Si el estado del servidor no es ACTIVE la biblioteca inicia el servidor.

Tabla 3 Descripción del caso de prueba Inicio de los servicios.

Id del escenario	Escenario	Descripción	Respuesta del sistema	Resultado de la prueba
EP 1.1:	Servidor solicitado ya está activo.	Servidor en estado activo.	Se espera que la biblioteca termine la funcionalidad ya que el servidor ya está activo.	La biblioteca ubica el servidor con estado ACTIVE.
EP 1.2:	Servidor solicitado no está activo.	Servidor en estado no activo.	Se espera que la biblioteca proceda a inicializar el servidor.	La biblioteca inicializa correctamente el servidor y muestra estado ACTIVE para el mismo.

Ilustración 10 Resumen de los escenarios para inicio de servicios.

Nombre de la funcionalidad	**Descripción General**	**Escenarios de Prueba**	**Flujo del escenario**
Parada de servicios.	La biblioteca debe permitir la parada del servidor que posee los servicios solicitados.	**EP 1.1:** Servidor solicitado inactivo.	• Se realiza la petición de parada. • Se chequea si se está conectado a un registro. • Se chequea estado del servidor. • Si el servidor está INACTIVE se termina la funcionalidad, ya está parado el servicio.
		EP 1.2: Servidor solicitado activo.	• Se realiza la petición de parada. • Se chequea si se está conectado a un registro. • Se chequea estado del servidor. • Si estado ACTIVE la biblioteca para el servidor.

Tabla 4 Descripción del caso de prueba Parada de los servicios.

Id del escenario	**Escenario**	**Descripción**	**Respuesta del sistema**	**Resultado de la prueba**
EP 1.1	Servidor solicitado inactivo.	Servidor solicitado en estado inactivo.	Se espera que la biblioteca termine la funcionalidad ya que el servidor ya está inactivo.	La biblioteca ubica el servidor con estado INACTIVE.
EP 1.2	Servidor solicitado activo.	Servidor en estado activo.	Se espera que la biblioteca proceda a detener el servidor.	La biblioteca detiene correctamente el servidor y muestra estado INACTIVE para el mismo.

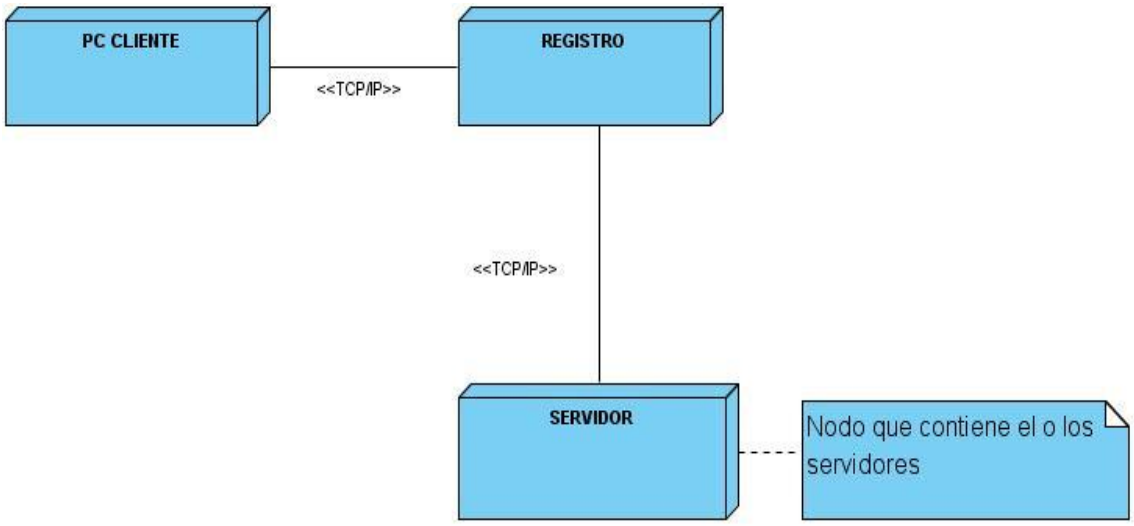
Ilustración 11 Resumen escenarios para parada de servicios.

4.5.5 Resumen de las Pruebas

Se realizaron un total de 2 casos de prueba en los que los resultados en cada uno de los escenarios que los comprende fue el esperado, es decir, no se obtuvieron no conformidades, por lo que se demuestra la validez de la biblioteca en cuanto a las especificaciones para las cuales fue desarrollada y sus principales funcionalidades.

4.6 Diagrama de Despliegue

El Diagrama de Despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes.



.

Ilustración 12 Diagrama de Despliegue.

El diagrama anterior muestra los elementos (nodo[24]) necesarios para el despliegue de la biblioteca en cuestión (Biblioteca de Disponibilidad o High Performance) así como las relaciones físicas entre estos, por tanto, para el uso de la biblioteca se necesita una PC[25] cliente de donde se realizarán las peticiones necesarias al servidor para conocer el grado de disponibilidad de los servidores o para la utilización de los servicios que brinda el mismo, una PC donde estará el registro y su respectiva configuración que será

[24] Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria.

[25] Personal Computer o Computador Personal traducido al español.

intermediario en la comunicación entre la aplicación cliente y el servidor, y por último, una PC servidor donde estará la estructura lógica nodo que contiene el o los servidores que brindan los servicios al cliente.

4.7 Conclusiones del capítulo

En el capítulo se plasmaron los estándares de documentación y codificación usados, que son aplicados en el Centro de Informática Industrial. Así como se validaron las funcionalidades de la biblioteca y se exponen los resultados obtenidos a partir de los diferentes casos de prueba efectuados a las funcionalidades para comprobar la validez de la biblioteca y que ésta a su vez cumple con el objetivo para el cual fue desarrollada, lográndose una solución viable para una primera versión del producto. Como conclusión de estas pruebas se puede plantear que de un total de 2 casos de prueba, 2 resultaron satisfactorios y ninguno fallido por lo que se concluye que el proceso de validar esta primera versión de la biblioteca de disponibilidad concluye de manera exitosa.

Conclusiones Generales

Con la culminación del presente trabajo se logró desarrollar la primera versión de la biblioteca de disponibilidad (HighPerformance) basado en tecnologías libres en su totalidad. Además, se utilizaron satisfactoriamente algoritmos viables del servicio IceGrid en el desarrollo de la biblioteca de disponibilidad, la cual a su vez constituye una base de conocimiento agregada sobre el uso de la biblioteca ICE que es usada además para las comunicaciones del SCADA. La utilización de dicha biblioteca proveerá al SCADA UX y al Centro de Informática Industrial una herramienta que facilite el control y monitorización de los estados de los módulos en cuanto a su disponibilidad, y realizar las operaciones necesarias sobre éstos como inicio, reinicio y parada de los servicios de forma gráfica y remota.

De manera general se cumplió con el desarrollo del mecanismo que permitió monitorear la disponibilidad de los módulos del configurador SCADA UX y se logró los resultados esperados con la implementación de la biblioteca de disponibilidad como solución al problema científico planteado.

Recomendaciones

- Integrar la biblioteca de disponibilidad al SCADA UX.
- Incorporar a la biblioteca una interfaz de usuario que permita visualizar las acciones de la biblioteca partiendo del demo propuesto.
- Incorporar mecanismos de seguridad necesarios para evitar vulnerabilidades en el uso de la biblioteca de disponibilidad.

Referencias Bibliográficas

1. **Ravelo Hernández, Luis Ángel.** *Módulo de Comunicación para Sistemas SCADA usando tecnologías libres.* La Habana : s.n., 2010.

2. **Guerra Garayta, Ariel.** *WebUInit, Interfaz Hombre-Maquina del SCADA en la Web.* 2009.

3. **Valenciano Odio, Lien.** *Desarrollo del Módulo de Distribución de Datos para el Sistema de Control y Supervisión de Variables Ambientales.* 2010.

4. **Puerto Sorio, Rosalina.** *Sistema de Vigilancia por Cámaras Cancerbero (SVCC).* 2010.

5. **Vallejo Fernández, David.** *Documentación de ZeroC ICE.* 2006.

6. **Páez Castillo, Douglas.** *Desarrollo de la arquitectura de la Plataforma de Televisión Informativa PRIMICIA.* 2009.

Bibliografía

1. AG, Siemens. Human Machine Interface with SIMATIC HMI. [En línea] 2011. http://www.automation.siemens.com.

2. Bakken, D. *Middleware.* 2005.

3. Carazo, Omar. [En línea] http://escert.upc.edu/userfiles/SCADA.pdf

4. Centre for the protection of national infrastructure. [En línea] http://www.cpni.gov.uk/advice/infosec/business-systems/scada/.

5. DiCicco, Ernest A**.** SCADA Solutions from ABB. [En línea] 2008. http://www.abb.com/industries.

6. División Automatismo y Control Inteligente . [En línea] http://xonet.com.ar/plc_archivos/division11.htm#scada.

7. Elers Pérez, Alberto y Sariol Pérez, Jorge Luis. *Implementación del Módulo Gestión de Actividades del Tutor Virtual de Evaluación para el Aprendizaje Autónomo de Idiomas.* 2010.

8. El Mundo. [En línea] http://www.elmundo.es/elmundo/2010/07/20/navegante/1279619263.html.

9. Fernández H, Luis. Software Guisho. [En línea] 2009. http://software.guisho.com/patrones-de-diseno.

10. Fowler, Martin & Rice, David & Foemmel, Matthew & Hieatt, Edward & Mee, Robert & Stafford, Randy. *Patterns of Enterprise Application Architecture. Addison Wesley.* 2002.

11. Grupo de Ingenieria de Fabricación. [En línea] http://fabricacion.dimf.etsii.upm.es/wikifab/images/9/99/Wincc_Plant_Intelligence.pdf.

12. Ingeniería de software I. . [En línea] 24 de Febrero de 2010. http://www.scribd.com/doc/3062020/Capitulo-I-HERRAMIENTAS-CASE..

13. Innovación y Promoción Económica. [En línea] http://www.tecnologico.deusto.es/projects/flexeo/files/E1.1-Estado_del_arte_Arq_Mon_Rem-v1.0.pdf.

14. JOHN WILEY & SONS, Buscmann, Frank & Meunier, Regine & Rohnert, Hans & Sommerlad, Peter & Stal, Michael. *PATTERN - ORIENTED SOFTWARE ARCHITECTURE. SYSTEM OF PATTERNS. .* Toronto : s.n., 2001.

15. Martín, Aurelio**.** [En línea] http://www.aslan.es/boletin/boletin39/siemens.shtml.

16. masadelante.com. [En línea] http://www.masadelante.com/faqs/sistema-operativo.

17. OPC Foundation. [En línea] 2009-2010. http://www.opcfoundation.org/.

18. Open Scada Security Project. [En línea] 2009. http://www.scadasecurity.org/index.php/Main_Page.

19. Paredes, Juan Pedro. *Alta Disponibilidad para Linux.*

20. Pilar, María del. [En línea] http://www.scribd.com/doc/13719562/Lenguaje-de-Programacion.

21. QT, sitio oficial. [En línea] 2008-2011. http://qt.nokia.com/products/developer-tools.

22. Reyes Bermúdez, Enrique. *Sistema de autenticación para el módulo Seguridad del proyecto SCADA Guardián del Alba.* 2009.

23. SearchDataCenter.com. [En línea] 2011. http://searchdatacenter.techtarget.com/definition/high-availability.

24. Siemens. [En línea] 2009. http://www.sispm.com/descargas/02%20HMI/Manuales/WinCC_V6_Opciones.pdf.

25. SlideShare. [En línea] 2005. http://www.slideshare.net/guest259ab3/sistemas-de-control-distribuido.

26. Software de vigilancia. [En línea] 11 de Marzo de 2010. http://www.antirrobo.net/vigilancia/software-de-vigilancia.html.

27. Tàvara, Alex. alextàvara. [En línea] http://alextavara.com/blog/interfaz-grafica-con-qt-en-c.

28. Vázquez, José María Morales. *Diseñando sistemas de alta disponibilidad y tolerantes a fallos, versión 1.3.*

29. *Visual Paradigm for UML.* [En línea] 17 de Febrero de 2010. http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML.

30. Webmaster. Open Source Middleware. [En línea] 2007. http://middleware.objectweb.org/.

Glosario de Términos

Arquitectura: Diseño que muestra los bloques de construcción físicos y lógicos de una aplicación distribuida (o algún otro sistema de software) y las relaciones entre ellos.

Disponibilidad: Capacidad del sistema para estar disponible cuando se necesite.

Estándar: Norma que se utiliza como punto de partida para el desarrollo de servicios, aplicaciones, protocolos, entre otros.

Framework: En el desarrollo de software es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

ICE: Es un middleware orientado a objetos, es decir, ICE proporciona herramientas, APIs, y soporte de bibliotecas para construir aplicaciones cliente-servidor orientadas a objetos.

SCADA: Un SCADA es una aplicación de software, especialmente diseñada para funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables, entre otros.) y controlando el proceso de forma automática desde una computadora.

Software: Es la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo y que “un producto de software es un producto diseñado para un usuario”.