Universidad de las Ciencias Informáticas

Facultad 5

Trabajo de Diploma para optar por el título de:

Ingeniero en Ciencias Informáticas

Aislamiento de procesos en la obtención de puntos calculados del SCADA UX

Autor: *Angel O. George Varela*

Tutor: *Ing. Juan Carlos González Tamayo*

Co-Tutora: *Ing. Ana Silvia Tellería Martínez*

"Año del 54 aniversario de la Revolución"

La Habana, Julio 2012

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas; así como al Centro de Informática Industrial para que hagan el uso que estimen pertinente con este trabajo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____ _____

Autor

Angel Omelio George Varela

Tutor

Ing. Juan Carlos González Tamayo

_____

Co-Tutora

Ing. Ana Silvia Tellería Martínez

Datos de contacto

Tutor: Juan Carlos Gonzáles Tamayo

Edad: 25

Ciudadanía: Cubana

Institución: Universidad de las Ciencias Informáticas

Título: Ingeniero en Ciencias Informáticas

Categoría Docente: Ninguna

E-mail: jctamayo@uci.cu

Graduado de la UCI, con 3 años de experiencia en el tema de los sistemas SCADA.

Co-Tutora: Ana Silvia Tellería Martínez

Edad: 28

Ciudadanía: Cubana

Institución: Universidad de las Ciencias Informáticas

Título: Ingeniero en Ciencias Informáticas

Categoría Docente: Instructor

E-mail: atelleria@uci.cu

Graduado de la UCI, con 6 años de experiencia en el tema de los sistemas SCADA.

Dedicatoria

A mis padres que de una forma u otra siempre han sabido estar ahí para mí en todo momento, sin ellos no sería como soy.

A mi hermano Enmanuel que siempre ha permanecido junto a mí tanto en los buenos momentos como en los malos.

A mi hermano Brian que a pesar de estar lejos nunca me olvida, como tampoco yo a él.

A todos mis abuelos, en especial a mi abuelo José Angel George Pérez y mis abuelas Carmelina y Ramona.

A mi padrino Wilfredo, que siempre me ha ayudado en mi crecimiento y formación.

A todos mis tíos, en especial a mi tío Antonio Varela.

A mi hermosa novia Enelis por todo lo que hemos compartido juntos y por el amor brindado.

Agradecimientos

A mis padres, mis hermanos y mi novia.

A mis primos, tíos, abuelos y familia en general.

A mis amigos, los antiguos, los nuevos, los olvidados, los presentes, los que ya no se encuentran en este mundo y los que se mantienen a mi lado en especial a:

Carlos Javier, Adaibel, Nelson Manuel, Bruce, Reinier, Lixander, Osvaldo, Carlos Rafael, Jansel, Juan Manuel, Enrique, Lázaro Miguel, Michel, Luisito, Eduardo, Ernesto Fuentes, Ángel Ulises, Ángel Díaz, Alian, Rosbel y tantos más que no me alcanzaría todo el documento para mencionarlos; los cuales me han enseñado tantas y tantas lecciones para vivir.

Al grupo de anime de la Mella, por todo lo que compartimos y por ayudarme una vez a la semana a romper con la rutina diaria de la universidad.

A todos los que de una forma u otra han tomado parte en mi formación tanto educacional como personal y espiritual.

A mi tutor Juan Carlos y mi co-tutora Ana Silvia, que han hecho todo lo posible porque esta tesis salga adelante y a todos los que han hecho posible de una forma u otra la realización de este trabajo.

A la UCI y la Revolución por ofrecerme la oportunidad que muchos no tuvieron ni tendrán.

Resumen

El sistema de supervisión y control SCADA UX, desarrollado en el Centro de Informática Industrial es un sistema distribuido compuesto por varios módulos, entre los que se encuentra el módulo de procesamiento. Una de las funcionalidades de dicho módulo es el procesamiento de puntos calculados. Los puntos calculados no provienen directamente de los dispositivos de medición, sino que se obtienen mediante operaciones sobre otras variables. Su configuración se realiza mediante código C++. Actualmente las funciones definidas para la obtención de estos puntos se ejecutan sin ningún tipo de limitación. Esta situación constituye una brecha de seguridad que permite la realización de distintos tipos de ataques al sistema.

En el presente trabajo se plantea como objetivo desarrollar el procesamiento de puntos calculados del SCADA UX utilizando la tecnología *sandbox* para evitar los ataques de explotación de errores. Para dar cumplimiento al objetivo planteado se estudiaron tecnologías de aislamiento de procesos que pudieran utilizarse en el procesamiento de puntos calculados. Sobre la tecnología seleccionada se desarrollaron aplicaciones demostrativas y se realizaron pruebas de rendimiento y seguridad. Finalmente se integró la tecnología evaluada al módulo de procesamiento del SCADA UX y se realizaron pruebas para comprobar el correcto funcionamiento de la solución.

Palabras clave: Aislamiento de procesos, brecha de seguridad, explotación de errores, puntos calculados, *sandbox*, SCADA.

Índice

Introducción 1
Capítulo 1: Fundamentación Teórica 5
1.1 Introducción 5
1.2 Seguridad 5
1.2.1 Clasificación de los ataques de seguridad 5
1.2.2 Tipos de ataques 6
1.3 Aislamiento de procesos 7
1.4 Procesamiento 8
1.5 Punto 9
1.6 Punto calculado 10
1.7 Procesamiento de puntos calculados en sistemas SCADA 11
1.7.1 SCADA DDS 12
1.7.2 SCADA Eros 12
1.7.3 SCADA GALBA 13
1.7.4 SCADA UX 15
1.8 Conclusión del estudio de sistemas SCADA 17
1.9 Conclusiones parciales 17
Capítulo 2: Selección tecnológica 18
2.1 Introducción 18
2.2 Estudio de aplicaciones sandbox 18
2.2.1 Rainbow 18
2.2.2 Immunity 19
2.2.3 NaCl (Google Native Client) 19
2.2.4 Plash 19
2.2.5 Selinux 19
2.2.6 Seccomp-nurse 19

2.2.7 Shikashi 20
2.2.8 Bibliotecas sandbox (libsandbox & pysandbox) 20
2.2.9 Sandboxie 20
2.3 Selección del sandbox 21
2.4 Embebido de lenguaje 22
2.4.1 Lenguaje de programación Ruby 22
2.4.2 Embeber Ruby en C/C++ 22
2.5 Pruebas de rendimiento sobre el sandbox 24
2.6 Pruebas de seguridad sobre el sandbox 26
2.7 Ambiente de desarrollo 29
2.7.1 Sistema Operativo GNU/Linux 30
2.7.2 Lenguajes de programación 30
2.7.3 Metodología de desarrollo AUP 30
2.7.4 Herramienta CASE 31
2.7.5 Lenguaje de modelado UML 32
2.7.6 Entorno de desarrollo Eclipse 32
2.8 Conclusiones parciales 32
Capítulo 3: Análisis y diseño de la solución 33
3.1 Introducción 33
3.2 Modelo del dominio 33
3.3 Definición de requisitos 34
3.4 Diagrama de clases del diseño 36
3.4.1 Descripción de las clases 37
3.4.2 Descripción de las relaciones entre las clases 38
3.5 Conclusiones parciales 39
Capítulo 4: Implementación y pruebas 40
4.1 Introducción 40
4.2 Diagrama de componentes 40

4.3 Estilo de código....41
4.4 Pruebas....41
4.4.1 Descripción de los casos de prueba....42
4.4.2 Resultado de las pruebas....43
4.5 Discusión de resultados....43
4.6 Conclusiones Parciales....44
Conclusiones....45
Recomendaciones....46
Referencias bibliográficas....47
Bibliografía....50
Anexos....52
Glosario de términos....54

Índice de figuras

Figura 1. Interfaz gráfica del SCADA Eros. 13
Figura 2. Interfaz gráfica en Qt para el cálculo de puntos analógicos, modo de operación “Periódico”. 14
Figura 3. Interfaz gráfica en Qt para el cálculo de puntos digitales, modo de operación “por Excepción”. 15
Figura 4. Ejemplo de interfaz en la que se muestra la función de un punto calculado. 16
Figura 5. Ciclo de vida de AUP. 31
Figura 6. Modelo del dominio. 33
Figura 7. Diagrama de clases del diseño. 37
Figura 8. Diagrama de componentes. 40
Figura 10. Ejemplo de embebido de Ruby en C/C++. 52
Figura 11. Ejemplo de fichero de Ruby con sandbox Shikashi. 52
Figura 12. Ejemplo de fichero utilizado para configurar los permisos del sandbox Shikashi. 53

Índice de tablas

Tabla 1. Especificación de calidad 11
Tabla 2. Resultado de prueba de rendimiento #1 25
Tabla 3. Resultado de prueba de rendimiento #2 25
Tabla 4. Resultado de prueba de rendimiento #3 25
Tabla 5. Resultado de prueba de rendimiento #4 26
Tabla 6. Resultados de pruebas de seguridad con el sandbox Shikashi 27
Tabla 7. Definición del requisito: Cargar configuración de punto calculado 35
Tabla 8. Definición del requisito: Obtener punto calculado 36
Tabla 9. Descripción de casos de prueba 42

Introducción

El aislamiento de procesos o entorno limitado (del inglés *sandbox*) es un mecanismo utilizado para prevenir fallos en la ejecución de programas que puedan poner en peligro la seguridad de un proceso (1). Esta técnica restringe el nivel de acceso o ejecución de las aplicaciones, permitiendo aislar a las mismas en un entorno controlado.

La seguridad es una característica importante a tener en cuenta en el desarrollo de un sistema SCADA.

Los sistemas de Supervisión, Control y Adquisición de Datos (SCADA por sus siglas en inglés, *Surpervisory Control and Data Adquisition*) son aplicaciones de software diseñadas con el objetivo de monitorear y controlar procesos productivos o de infraestructura automatizados (2).

A partir de los requerimientos y las necesidades de automatización de varias empresas cubanas, se desarrolla en el Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas (UCI) el sistema SCADA UX.

Este emplea una arquitectura distribuida de módulos los cuales son conectados a través de una capa de comunicaciones conocida como Middleware, encontrándose entre ellos los módulos de: recolección, procesamiento, configuración y la interfaz hombre – máquina (HMI por sus siglas en inglés, *Human Machine Interface*) (3).

En un proceso industrial supervisado mediante el SCADA UX la información de los distintos dispositivos de medición (como pueden ser un termómetro que mide la temperatura en una caldera de vapor o un metro-contador que supervisa el consumo de electricidad de un motor), es accedida por el recolector mediante el empleo de manejadores o drivers y luego es enviada al módulo de procesamiento. El módulo de procesamiento analiza y transforma esta información según las configuraciones establecidas y se realiza la publicación de los datos resultantes para ser mostrados en las consolas de operación a los usuarios del sistema.

El análisis y las transformaciones que realiza dicho módulo sobre las variables se conoce como “procesamiento de puntos” y constituye una de sus tareas fundamentales (4).

En los sistemas SCADA se maneja el concepto de punto. Un punto es un valor simple de entrada y (o) salida que es supervisado por el sistema. Constituye una representación del valor que ofrece un dispositivo de medición y se obtiene al realizar operaciones de lectura sobre la memoria de dicho

dispositivo. Un ejemplo de punto sería el valor que provee un manipulador de válvula indicando el flujo que se permite pasar por una tubería (5).

Existe un tipo de punto al cual se le llama punto calculado, debido a que toma su valor por la evaluación de fórmulas matemáticas que involucran el valor de uno o más puntos ya sean calculados o no (4).

Actualmente como parte del procesamiento de los puntos calculados es necesario obtener su valor mediante la ejecución de funciones de bibliotecas compartidas. Estas son programadas en las estaciones HMI clientes mediante el lenguaje C++ por los usuarios de la aplicación. La ejecución de dichas bibliotecas en un ambiente sin ningún tipo de restricción constituye una brecha de seguridad; puesto que en la edición de dichos puntos no se realiza comprobación del código escrito. Esto permite a un usuario malintencionado desarrollar código que vulnere al sistema SCADA, así como al equipo en el que se ejecuta.

Aprovechando esta vulnerabilidad pudiera asociarse la siguiente instrucción a un punto calculado: “system ("shutdown -R now");”. Al evaluar el punto calculado, dicha instrucción pudiera provocar una caída del sistema, causando una interrupción indefinida en la supervisión. Otra de las acciones que puede realizarse utilizando esta brecha es el acceso irrestricto a los archivos del usuario que está ejecutando el SCADA en el sistema, incluyendo los archivos de configuración y *logs* del SCADA (*Tampering*). Un agresor pudiera aprovechar también esta vulnerabilidad para crear una puerta trasera (*backdoor*) en el ordenador atacado. Estos ataques mencionados con anterioridad pertenecen al grupo conocido como “Explotación de errores”.

A partir de la situación problémica expuesta con anterioridad se define como **problema a resolver**:

¿Cómo proveer de un entorno controlado al procesamiento de puntos calculados del SCADA UX?

Teniendo en cuenta el problema a resolver, se plantea como **objeto de estudio** la seguridad en la ejecución de programas en entornos controlados y dentro de esta área de conocimientos se propone como **campo de acción** la seguridad en la ejecución de programas en entornos controlados en el procesamiento de puntos calculados de un sistema SCADA.

Para dar solución al problema a resolver, se plantea el siguiente **objetivo**:

Desarrollar el procesamiento de puntos calculados del SCADA UX utilizando la tecnología *sandbox* para evitar los ataques de explotación de errores.

Para lograr los objetivos trazados se plantean las siguientes **tareas de investigación**:

1. Determinación de fortalezas y debilidades de manera general y atendiendo a la seguridad en el procesamiento de puntos calculados en otros SCADA utilizados en la actualidad.
2. Evaluación del procesamiento de puntos calculados en el SCADA UX identificando ventajas y deficiencias del mecanismo actual de procesamiento.
3. Selección de la tecnología adecuada al procesamiento de puntos calculados del SCADA UX a partir del análisis de bibliotecas para aislamiento de procesos o *sandboxing* así como los lenguajes en los que están implementadas.
4. Desarrollo de aplicaciones demostrativas con la tecnología seleccionada para comprobar la viabilidad de su uso.
5. Diseño y aplicación de casos de pruebas sobre las aplicaciones demostrativas desarrolladas para comprobar el comportamiento de los parámetros básicos de funcionamiento.
6. Integración en el módulo de procesamiento del SCADA UX de la solución propuesta y realización de pruebas a la misma para comprobar su correcto funcionamiento.
7. Valoración de los resultados obtenidos a partir de la investigación realizada y la integración de la solución.

Para la realización de la investigación se emplearon varios **métodos de investigación científicos**.

De los métodos científicos teóricos se emplearon el **Analítico-Sintético**, **Inductivo-Deductivo**, **Sistémico** y la **Modelación**.

De los métodos científicos empíricos se empleó el **Experimento**.

El método **Analítico-Sintético** se emplea para analizar la información obtenida en el proceso de investigación y a partir de la misma y su división, generar nuevos conceptos que sustenten la propuesta de solución presentada.

El método **Inductivo-Deductivo** se emplea en el estudio del estado del arte de los diversos sistemas SCADA, particularmente en sus componentes de procesamiento de puntos calculados.

El método **Modelación** se emplea en el diseño de las pruebas, aplicaciones y algoritmos que sustentan la evaluación tecnológica para dar solución al problema planteado.

El método **Sistémico** se emplea en el estudio de las aplicaciones de prueba desarrolladas y en la integración de la solución en el módulo de procesamiento del SCADA UX.

El método **Experimento** se utiliza en la realización de las pruebas y el análisis de sus resultados.

Con el objetivo de organizar y darle una estructura al trabajo se ha decidido dividir el mismo en cuatro capítulos. A continuación se muestra como se encuentran estructurados y un resumen de lo que se aborda en cada uno de ellos:

En el **Capítulo 1 Fundamentación teórica**, se describen los conceptos asociados al estudio realizado en el presente trabajo y se muestra un estado del arte referente al procesamiento de puntos calculados en varios sistemas SCADA.

En el **Capítulo 2 Selección tecnológica**, se investigan las tecnologías de aislamiento de procesos. Se definen y aplican casos de prueba para evaluar las tecnologías seleccionadas y se expone el ambiente utilizado para el desarrollo de la solución propuesta.

En el **Capítulo 3 Análisis y diseño de la solución**, se muestra el estado actual del negocio, así como los modelos asociados a la solución. Se describen los requisitos que debe cumplir la solución.

En el **Capítulo 4 Implementación y pruebas**, se valida la solución a través de las pruebas a los requisitos definidos. Se realiza la discusión de los resultados obtenidos.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

En el presente capítulo se introducen los basamentos teóricos relacionados con la seguridad y los ataques de explotación de errores de manera general, así como de la técnica de ejecución controlada de procesos *sandboxing*. Se analiza el procesamiento de puntos calculados en diversos sistemas SCADA incluyendo el SCADA UX, con el objetivo de estudiar el estado del arte actual de estos sistemas en cuanto al uso de tecnologías *sandbox* y comprender la naturaleza de lo que se quiere lograr con el desarrollo del presente trabajo.

1.2 Seguridad

Se entiende como seguridad una característica de cualquier sistema (informático o no) que indica que ese sistema está libre de todo peligro, daño o riesgo, y que es, en cierta manera, infalible. Como esta característica, particularizando para el caso de sistemas operativos o redes de computadores, es muy difícil de conseguir (según la mayoría de expertos, imposible), se suaviza la definición de seguridad y se pasa a hablar de fiabilidad (probabilidad de que un sistema se comporte tal y como se espera de él) más que de seguridad; por tanto, se habla de sistemas fiables en lugar de hacerlo de sistemas seguros (6).

Más concretamente se puede definir la seguridad informática como el área de la informática que se encarga de la protección de la información, así como de la estructura computacional.

Cualquier parte de un sistema de computación puede ser el destino de un delito. Se entiende por sistema de computación al conjunto de hardware, software, medios de almacenamiento, datos, y personal que una organización utiliza para realizar tareas de computación. En esencia cualquier sistema es tan vulnerable como su punto más débil (7).

1.2.1 Clasificación de los ataques de seguridad

Se pueden definir como ataques, todas aquellas acciones que suponen una violación de la seguridad de un sistema.

Estas acciones se pueden clasificar de modo genérico según los efectos causados, como (8) (9)**:**

- Interrupción: cuando un recurso del sistema es destruido o se vuelve no disponible.
- Intercepción: una entidad no autorizada consigue acceso a un recurso.

- Modificación: alguien no autorizado consigue acceso a una información y es capaz de manipularla.
- Fabricación (o Generación): cuando se insertan objetos falsificados en el sistema.

También se pueden ordenar por modalidades de ataque según la forma de actuar (8)**:**

- Escaneo de puertos: esta técnica consiste en buscar puertos abiertos, y fijarse en los que puedan ser receptivos o de utilidad.
- Ataques de autentificación: cuando un atacante suplanta a una persona con autorización.
- Explotación de errores: suceden en el momento que se encuentran agujeros de seguridad en los sistemas operativos, protocolos de red o aplicaciones.
- Ataques de denegación de servicio (DoS por sus siglas en inglés, *Denial of Service*): consiste en saturar un servidor con pedidos falsos hasta dejarlo fuera de servicio.

1.2.2 Tipos de ataques

A través de los años se han desarrollado formas cada vez más sofisticadas de ataques para explotar agujeros (o brechas) en el diseño, configuración y operación de los sistemas (10). Una brecha de seguridad es un fallo en un programa el cual permite mediante su explotación violar la seguridad de un sistema informático. Existen diversos y variados tipos de ataques para cada una de las modalidades presentadas en el epígrafe anterior, y cada uno de estos puede incurrir en mayor o menor medida en los efectos también mencionados con anterioridad. Algunos de esos ataques están dirigidos a descubrir y explotar los agujeros de seguridad presentes en un sistema. En el presente trabajo no se pretende abarcar todos los ataques de seguridad existentes, pero sí dejar claro algunos conceptos referentes a los tipos de ataques que se mencionan a lo largo de este estudio.

Backdoor: Mediante esta técnica se puede permitir que los procesos lanzados por un usuario sin privilegios de administrador ejecuten algunas funcionalidades reservadas únicamente a usuarios con estos privilegios (11). Esta técnica permite también crear otras brechas por las cuales el atacante puede acceder al sistema de forma más sencilla. También permite crear agujeros por los que un atacante puede penetrar incluso cuando la brecha de seguridad original ha sido eliminada.

Tampering o Data Diddling: Esta categoría se refiere a la modificación desautorizada de los datos o el software instalado en el sistema víctima (incluyendo borrado de archivos). Son particularmente serios cuando el que lo realiza ha obtenido derechos de Administrador o Supervisor, con la capacidad de

disparar cualquier comando y por ende alterar o borrar cualquier información que puede incluso terminar en la baja total del sistema (12).

Rootkit: Es un conjunto de técnicas usadas frecuentemente por los intrusos informáticos o crackers que consiguen acceder ilícitamente a un sistema informático. Estas técnicas sirven para esconder los procesos y archivos que permiten al intruso mantener el acceso al sistema, a menudo con fines maliciosos. Todo tipo de herramientas útiles para obtener información de forma ilícita pueden ser ocultadas mediante *rootkits* (11).

Los que se han expuesto anteriormente pertenecen a la modalidad de ataques de "Explotación de errores".

Jamming o Flooding: Este tipo de ataques desactivan o saturan los recursos del sistema. Por ejemplo, un atacante puede consumir toda la memoria o espacio en disco disponible, así como enviar tanto tráfico a la red que nadie más pueda utilizarla (13).

Este ataque pertenece a la modalidad de "Ataques de denegación de servicio".

1.3 Aislamiento de procesos

El aislamiento de procesos o entorno limitado es un mecanismo para la ejecución aislada de procesos o software de dudosa confiabilidad. Los *sandboxing* son técnicas utilizadas en el aislamiento de programas con el objetivo de prevenir fallos en la ejecución de los mismos que puedan poner en peligro la seguridad de un proceso. Esta técnica limita o reduce el nivel de acceso a las aplicaciones que se ejecutan dentro del *sandbox* (1).

Limitaciones del sandboxing:

Las implementaciones de los *sandboxing* actualmente sufren 5 grandes tipos de limitaciones (14):

- Demasiado aislamiento: significa que las aplicaciones importantes a veces no se pueden ejecutar en un determinado entorno limitado debido a que el recinto de seguridad interfiere con su acceso a algún recurso crítico o función del sistema.
- Aislamiento insuficiente: significa que con la tecnología *sandboxing* a veces no se puede aislar un recurso que pudiera vulnerar la seguridad de un proceso.
- Escasa portabilidad: es una limitación de muchos *sandboxing*, los cuales no poseen una portabilidad u estrecha integración con la interfaz de usuario del sistema principal.

- Escasa usabilidad: se refiere a la incapacidad ocasional, contraria a los usuarios interesados, para beneficiarse de la tecnología *sandboxing*.
- Diseño y errores de implementación: se refiere a la dificultad de escribir la lógica del correcto aislamiento alrededor de miles de conceptos de otras personas.

Beneficios del sandboxing:

- Ofrece un espacio seguro para la ejecución de procesos.
- Permite que el proceso en que se utilice sea más difícil de vulnerar que un proceso en que no se utilice.
- Aumenta la privacidad de los datos.

Funcionamiento del sandboxing:

Existen dos mecanismos básicos para la aplicación del aislamiento: los espacios de nombres y los controles de acceso. Los espacios de nombres funcionan según el principio de que “los agentes son incapaces de manipular los recursos que no pueden nombrar”. Los controles de acceso son los intentos de ejecutar obligaciones de prueba de forma que “el agente A puede manipular los recursos de R mediante el método M”. Las tecnologías actuales de aislamiento utilizan ambos mecanismos para controlar la capacidad de los agentes y para manipular recursos como: páginas de memoria, archivos, interfaces de red, procesos, entre otros (15).

1.4 Procesamiento

Se entiende por procesamiento a la aplicación sistemática de una serie de operaciones sobre un conjunto de datos, generalmente por medio de máquinas, para explotar la información que estos datos representan (16).

En el caso de los sistemas SCADA el procesamiento se lleva a cabo sobre puntos, alarmas, comandos y estados de las comunicaciones. Dichos elementos constituyen unidades básicas y centrales de información de este tipo de sistemas. El procesamiento consiste en analizar los puntos y estados de las comunicaciones y a partir de esta información, detectar situaciones excepcionales que pueden convertirse en avisos de alarma para los manejadores o usuarios del sistema.

El módulo de procesamiento es también conocido como Base de Datos en Tiempo Real (BDTR), ya que se encarga de procesar y almacenar información en tiempo real sobre actividades que se estén ejecutando.

1.5 Punto

Un punto es una variable o valor simple de entrada y (o) salida que es supervisado y (o) controlado por el sistema SCADA (5).

Una variable de entrada se refiere a un valor que se obtiene de los dispositivos de medición (o dispositivos de campo) y se utiliza solamente para la lectura de datos. Por su parte una variable de salida se refiere a aquellas que son usadas para escribir sobre un dispositivo de campo con el objetivo de controlar parámetros de un proceso que necesite ser modificado en un momento dado. Mientras que una variable de entrada y salida se refiere a un valor que puede ser tanto leído como escrito por el sistema.

Un punto puede ser tratado como un valor analógico o digital.

Los puntos digitales toman valores dentro de un conjunto discreto, por ejemplo un dispositivo encendido o apagado (17).

Otro ejemplo de punto digital sería si se quisiera abrir o cerrar una llave para permitir (o restringir) el paso de agua por un conducto perteneciente a una estación de bombeo.

Los puntos analógicos pueden tomar infinitos valores entre dos valores cualesquiera de su dominio, por ejemplo la presión (17).

Otro ejemplo de punto analógico sería el valor de temperatura que puede alcanzar una caldera de fundición en una instalación metalúrgica.

Los puntos, tanto analógicos como digitales, pueden ser de: entrada, salida, entrada/salida o calculados. Los puntos calculados obtienen su valor a partir de fórmulas matemáticas que pueden tomar como operandos valores de otros puntos calculados o no.

Existe otra clasificación en la que los puntos pueden ser duros o suaves. Mientras que los puntos duros representan entradas o salidas reales conectadas al SCADA, un punto se clasifica como suave si representa el resultado a partir de una fórmula matemática o de operaciones lógicas que emplean valores de otros puntos duros o suaves. O sea los puntos calculados son también llamados como puntos suaves.

Los sistemas SCADA almacenan los valores de los puntos en combinación con la estampa de tiempo que indica el instante exacto en que fue recogido o calculado el dato y la calidad del mismo (18).

La calidad de un dato suele tomar los siguientes estados:

- Buena: cuando el dato fue leído sin problemas del dispositivo.
- Incierta: cuando el dato está tomando valores fuera de rango.
- Mala: cuando el dato no ha podido ser leído del dispositivo.

Esta información de calidad del dato es usada para conocer la fiabilidad del valor.

1.6 Punto calculado

Un punto calculado es una variable que se obtiene mediante la evaluación de fórmulas matemáticas. Puesto que no posee salida de campo es considerado simplemente una variable de entrada. En el caso del SCADA UX puede observarse como una entidad de software que reside en el módulo de procesamiento, la cual permite hacer operaciones aritméticas/lógicas sobre valores de otros puntos ya sean calculados o no.

Estos puntos suelen tener dos expresiones. Una expresión de activación que indica si el punto debe calcularse o no y una expresión de cálculo que es la que define su valor.

Los puntos calculados pueden ser de tipo analógicos o digitales, en dependencia de los valores que son utilizados para realizar el cálculo. Aunque existen estas dos clasificaciones para dichos puntos, en el SCADA UX solamente se manejan los analógicos.

Un ejemplo de punto calculado analógico sería el promedio de consumo de electricidad de tres edificios docentes de la UCI. En este caso cada edificio cuenta con su metro-contador. El problema de determinar el promedio de consumo de los edificios docentes de la UCI se resolvería con un punto calculado, el cual tomaría su valor mediante una fórmula matemática de promedio que emplee el valor de consumo de cada edificio. Su fórmula de cálculo quedaría de la siguiente forma: (C1+C2+C3) / 3. Siendo Ci el consumo eléctrico del edificio i.

Por su parte, un ejemplo de punto calculado digital sería el estado (abierto/cerrado) de una cantidad x de tuberías de agua conectadas a una tubería principal asociada con una estación de bombeo de un acueducto. Si se quisiera evitar el derroche de agua por abastecimiento excesivo a la tubería principal, se

obtendría el estado de las tuberías conectadas y mediante una fórmula matemática se establecería una restricción sobre la cantidad de tuberías que necesitan abastecer a la principal.

Un punto calculado digital solo puede ser calculado utilizando variables digitales u otros puntos calculados digitales. De la misma forma sucede con los puntos analógicos. Por lo tanto, no se puede calcular un punto digital utilizando un punto analógico o viceversa.

Existen dos modos de operación para llevar a cabo el cálculo de puntos: por excepción o periódicamente. El cálculo por excepción se realiza cuando se cumple la expresión de activación establecida previamente para determinar si se realiza o no el cálculo. Por su parte el modo de operación periódico se realiza atendiendo a los parámetros de tiempo definidos para ejecutar la operación cada x milisegundos (19).

Algunos operadores aritméticos no son válidos a la hora de realizar el cálculo de puntos. Los puntos calculados digitales solamente admiten operadores aritméticos lógicos para realizar sus operaciones. Mientras que los puntos calculados analógicos admiten operadores aritméticos y algunas funciones de cálculo como seno (SIN(x)), coseno (COS(x)), valor absoluto (ABS(x)), entre otras.

Una información importante a tener en cuenta a la hora del manejo de los puntos calculados es la calidad del dato calculado. Existen varios estándares para determinar la calidad de un punto calculado a partir de que su calidad es asignada en dependencia de la calidad de sus operandos.

La especificación empleada en los proyectos SCADA del CEDIN es la siguiente:

Tabla 1. Especificación de calidad

Calidad de Punto Calculado	Origen
Buena	La calidad de todos los operandos involucrados en el cálculo es buena
Incierta	Existe al menos un operando con calidad incierta y los restantes presentan calidad buena
Mala	Al menos uno de los operandos presenta mala calidad

1.7 Procesamiento de puntos calculados en sistemas SCADA

A continuación se estudia el procesamiento de puntos calculados en algunos sistemas SCADA con remarcado impacto industrial, incluyendo el SCADA UX. Se realiza esta investigación para conocer acerca de la implementación de *sandbox* en el procesamiento de puntos calculados en otros sistemas de este tipo y para evaluar el estado del arte actual de dicho proceso en sistemas SCADA.

1.7.1 SCADA DDS

Desarrollado por GE Consumer & Industrial. Es utilizado en la automatización de subestaciones y centrado en tareas de protección y control. Implementa una amplia gama de “Funciones” predeterminadas como: funciones de tensión, funciones de frecuencia, funciones de protección, entre otras. Dichas funciones están a disposición de los manejadores y usuarios, ahorrándoles con esto gran parte del trabajo (20).

Este sistema permite al usuario ajustar y modificar los valores de las funciones que ofrece, pero es inflexible en cuanto a la declaración de nuevas funciones. Es importante destacar que basan la mayor parte de su mecanismo en el uso de las mismas.

1.7.2 SCADA Eros

Eros es un SCADA desarrollado en Cuba por SERCONI. Está soportado en ambiente Windows NT/2000/XP/Vista/W7. Ha sido implantado en la Planta de Lixiviación y Lavado de la Fábrica Niquelera Comandante Ernesto Che Guevara de Moa y en la Fábrica de níquel Comandante René Ramos Latour. La Unión Eléctrica labora también junto a SERCONI para automatizar todos los Grupos Electrógenos mediante este SCADA.

Eros define a los puntos calculados como variables internas. Se denominan variables internas a aquellas que se crean a partir de cálculos que se realizan con las variables y registros del sistema, y con los operadores y funciones. El sistema permite definir el cálculo para variables analógicas y discretas (o digitales) (21).

Eros brinda una línea de edición para detallar las fórmulas que se quieren evaluar y constituyen expresiones que se pueden definir en una sola línea.

El sistema cuenta con un botón, que al oprimirse, brinda una lista de variables y sus estadísticas para seleccionar con precisión las que irán en la fórmula (21).

Este SCADA brinda un número de 96 funciones que pueden ser utilizadas en las fórmulas y cuenta con 23 operadores aritméticos que pueden usarse para el cálculo. Las funciones y cálculos que se programan se escriben en el lenguaje Object Pascal.

Dicho sistema compilará la fórmula que se editó una vez que el operador lo requiera, e indicará los errores si los tiene. De no contar con errores se informará ¡Compilación Exitosa!, esta compilación revisa la

sintaxis de la fórmula, pero el operador deberá responsabilizarse por su integridad, para que dé el resultado deseado (21).

Ejemplo de interfaz gráfica para variables internas en el SCADA Eros:

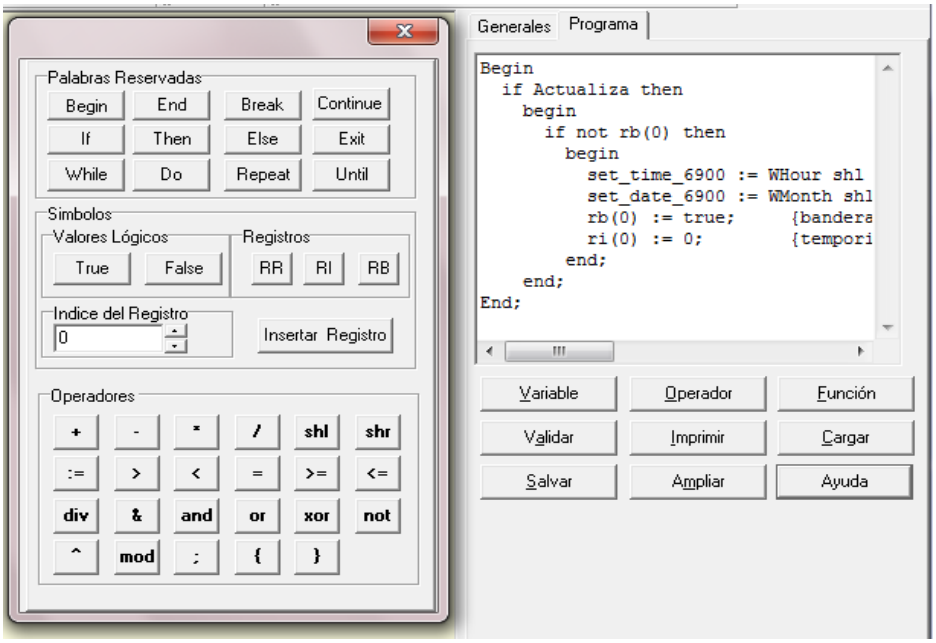

Figura 1. Interfaz gráfica del SCADA Eros.

1.7.3 SCADA GALBA

El procesamiento de puntos calculados en el SCADA GALBA se realiza al habilitar la opción “Tipo Calculado”. El sistema ofrece al usuario una interfaz para configurar parámetros de información general del punto, como:

- Nombre del Punto
- Descripción
- Grupo Operacional de Privilegios (GOP)
- Tipo de Dato
- Límite Inferior (Este parámetro no se aplica para Puntos Calculados Tipo Digital)
- Límite Superior (Este parámetro no se aplica para Puntos Calculados Tipo Digital)
- Valor Inicial
- Banda Muerta (Este parámetro no se aplica para Puntos Calculados Tipo Digital)

Este sistema ofrece al usuario una interfaz gráfica para seleccionar los puntos, las funciones lógicas y matemáticas; así como un espacio para mostrar la construcción del punto calculado y las alarmas configurables a este punto (19).

Este sistema no permite al operador escribir código al programar un punto calculado, sino que ofrece un número limitado de funciones para realizar estas operaciones.

Ejemplos de interfaz gráfica para los puntos calculados en el SCADA GALBA:

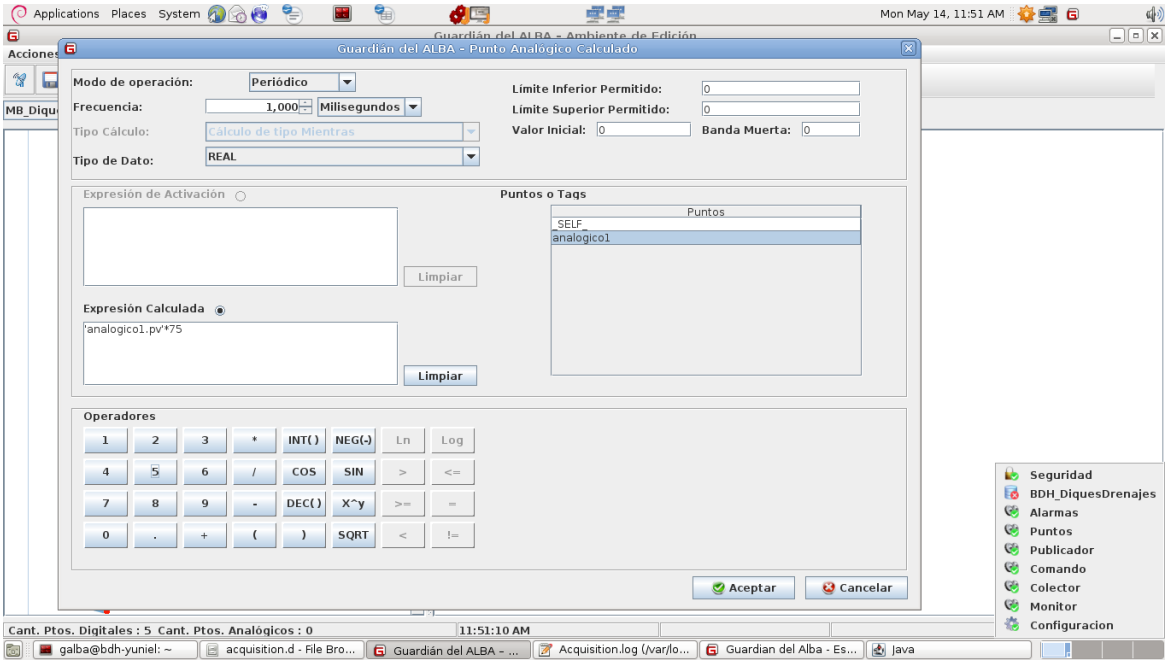

Figura 2. Interfaz gráfica en Qt para el cálculo de puntos analógicos, modo de operación "Periódico".

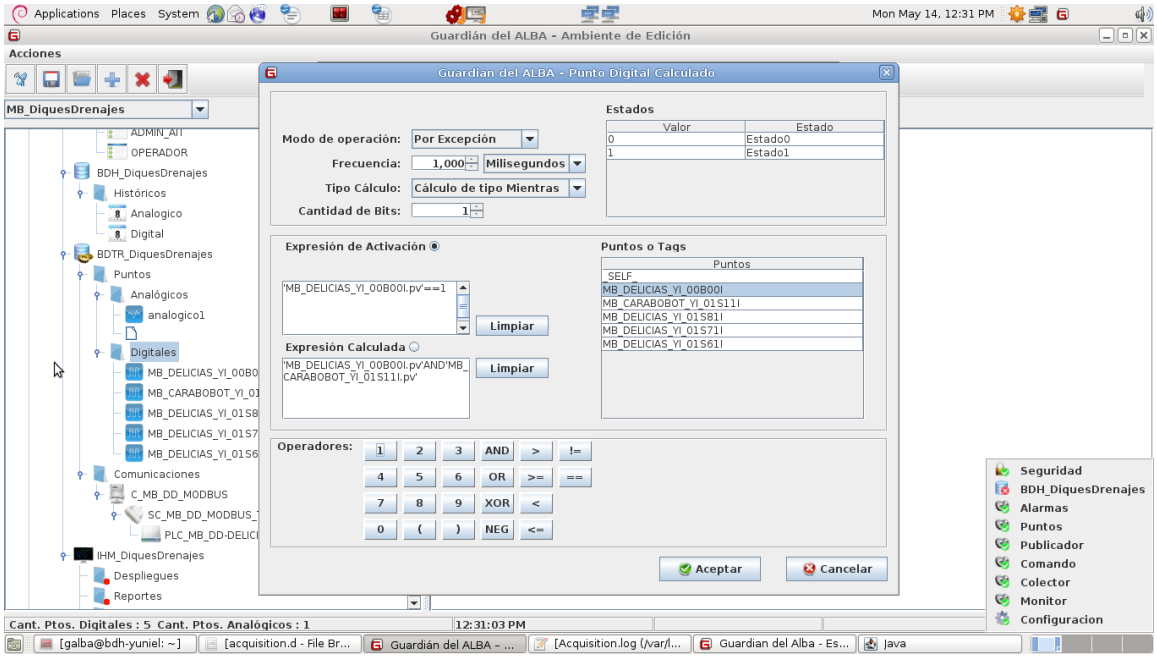

Figura 3. Interfaz gráfica en Qt para el cálculo de puntos digitales, modo de operación "por Excepción".

1.7.4 SCADA UX

El SCADA UX basa su procesamiento de puntos calculados en bases similares a las del SCADA GALBA, aunque emplean un mecanismo interno diferente.

El procesamiento de puntos calculados en este sistema se realiza al habilitar en la opción "Tipo de entrada" el parámetro "Calculada". El sistema ofrece al usuario una interfaz para configurar parámetros de información general del punto, como:

- Nombre del Punto a TAG
- Descripción
- Grupo Operacional de Privilegios
- Tipo de Dato
- Valor Inicial
- Límite Superior (Este parámetro no se aplica para Puntos Calculados Tipo Digital)
- Límite Inferior (Este parámetro no se aplica para Puntos Calculados Tipo Digital)
- Banda Muerta (Este parámetro no se aplica para Puntos Calculados Tipo Digital)
- Fuera de barrido

El SCADA UX ofrece al usuario una interfaz gráfica para seleccionar los puntos. Además proporciona un espacio para mostrar la construcción del punto calculado, las alarmas configurables a este punto y los datos históricos referentes al mismo.

Actualmente las funciones utilizadas para los puntos calculados en el SCADA UX son especiales y se escriben de la forma “tipo_retorno @nombre_función”, donde tipo_retorno es el valor que debe devolver la función y nombre_función es el nombre que se le asigna a la misma. Es importante tener en cuenta que a estas funciones no se les añade paréntesis “( )” al final como a las funciones comunes y que se diferencian del resto por el símbolo de arroba “@” delante del nombre de la función principal. Dichas funciones se programan en lenguaje C/C++.

Ejemplo de interfaz gráfica para los puntos calculados en el SCADA UX:

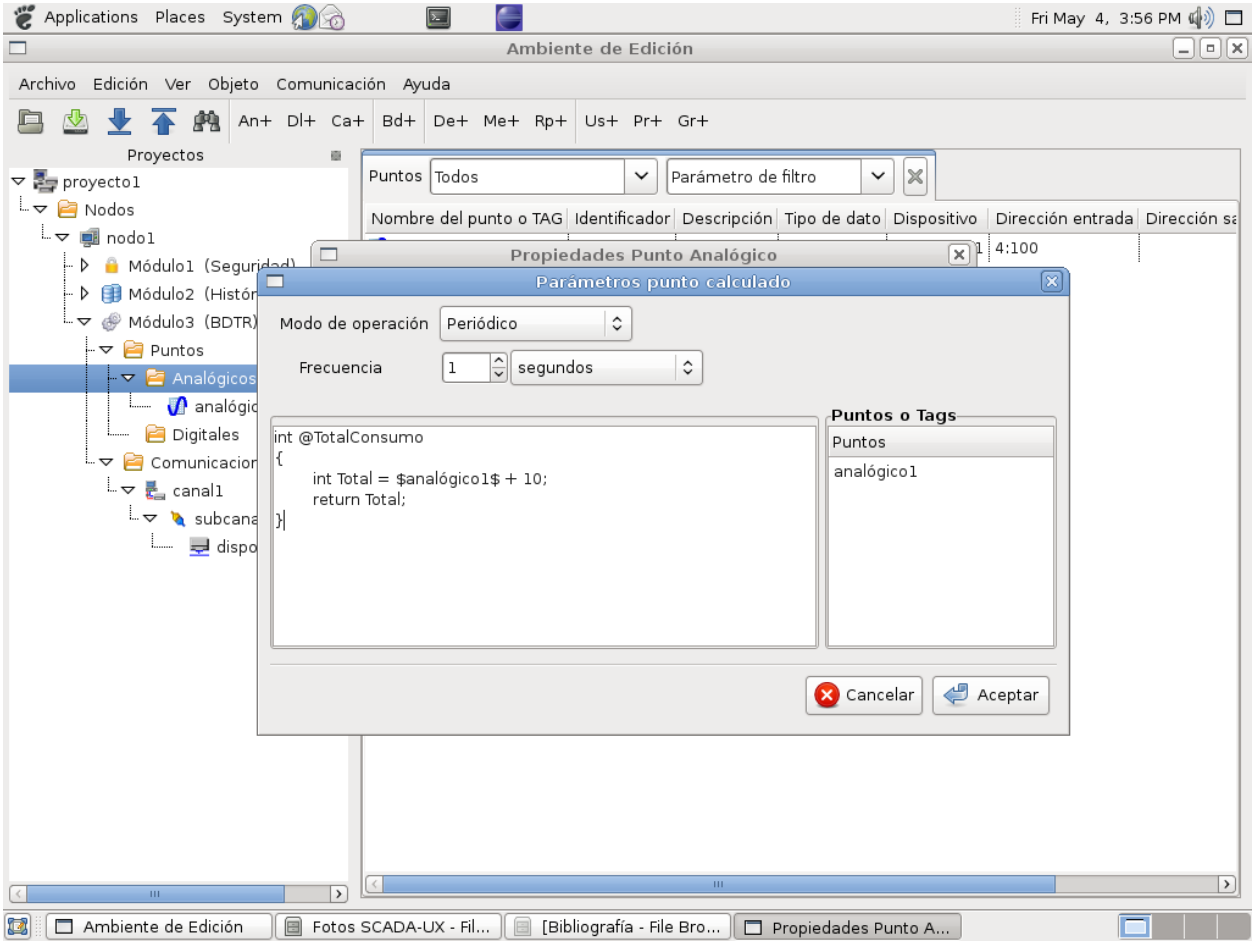

Figura 4. Ejemplo de interfaz en la que se muestra la función de un punto calculado.

1.8 Conclusión del estudio de sistemas SCADA

Como se ha visto anteriormente en el estudio de los sistemas SCADA analizados, ninguno emplea el aislamiento de procesos en su mecanismo para evitar las brechas de seguridad debido a que implementan sus propios lenguajes o mecanismos que por definición impiden realizar acciones malintencionadas sobre el sistema y los procesos supervisados. Por lo tanto, el área del *sandbox* es nueva en cuanto a su implementación en el mecanismo de procesamiento de puntos calculados.

Es importante observar que los SCADA DDS y GALBA disminuyen la brecha de seguridad, restringiendo las opciones que los usuarios y manejadores pueden ejecutar en su sistema. Esto por supuesto aumenta la seguridad del procesamiento de puntos calculados pero al costo de disminuir la flexibilidad. Entiéndase flexibilidad como la posibilidad de realizar un gran conjunto de operaciones complejas en las fórmulas de cálculo.

Los SCADA Eros y UX son sistemas flexibles, pero por otra parte mantienen la brecha de seguridad. En el caso del SCADA UX se tiene la ventaja de la velocidad de procesamiento que permite realizar cálculos de manera eficiente al ser usado código compilado en C/C++ para definir los puntos calculados; pero actualmente estos códigos se ejecutan sin restricciones, representando una peligrosa brecha de seguridad.

1.9 Conclusiones parciales

Como se ha podido apreciar, el procesamiento de puntos calculados de una manera u otra es una funcionalidad básica en todos los SCADA analizados en el presente trabajo. Es por ello que resulta fundamental contar con un mecanismo seguro, eficiente, robusto e intuitivo para realizar este proceso en el módulo de procesamiento del SCADA UX. Mediante las tecnologías *sandbox* se pueden procesar los puntos calculados de forma segura al limitar el nivel de permisos sobre los recursos del ordenador a los ficheros de cálculo, los cuales se deben limitar a ejecutar correctamente su función e impedir cualquier intento de acceso más allá de su propósito.

Capítulo 2: Selección tecnológica

2.1 Introducción

En este capítulo se realiza un estudio de varias tecnologías y bibliotecas de *sandboxing* evaluando sus ventajas y desventajas, teniendo en cuenta los niveles de restricción que permiten establecer a los programas que ejecutan. Se selecciona una tecnología de aislamiento de procesos y se realiza un estudio acerca del lenguaje en el cual y para el cual está desarrollada la misma, así como su facilidad de integración en el sistema SCADA UX. Se realizan pruebas de rendimiento y seguridad para evaluar la selección del *sandbox* y se define el ambiente utilizado para el desarrollo del estudio.

2.2 Estudio de aplicaciones sandbox

Existen varias aplicaciones *sandbox* para diversos lenguajes de programación. Debido a la complejidad de implementar este tipo de aplicación para lenguajes compilados casi no existen *sandbox* para los mismos. Un ejemplo de lenguaje compilado lo constituye el C/C++, el cual es usado en el desarrollo del SCADA UX.

Actualmente las bibliotecas para *sandboxing* en C/C++ están en desarrollo y las pruebas que se han realizado con las mismas han sido inestables. Un ejemplo de esto se puede encontrar en el desarrollo del Boost Sandbox, el cual no está completamente finalizado (22).

En el abanico de aplicaciones para *sandboxing* priman los que se han desarrollado sobre lenguajes script. Estos brindan una sintaxis sencilla para la construcción de dichas técnicas al ser lenguajes interpretados.

En el presente epígrafe se estudian diversas tecnologías que implementan técnicas de *sandboxing* con el fin de conocer más acerca de sus aplicaciones y seleccionar la que mejor se ajuste a las características del procesamiento de puntos calculados del SCADA UX, permitiendo proponer su introducción en el dicho sistema.

2.2.1 Rainbow

Es un *sandboxing* desarrollado por Michael Stone y colaboradores en nombre del proyecto “Una laptop por niño” (“One Laptop Per Child”). En la actualidad, este *sandbox* se compone de scripts de Python, una aplicación y un módulo para la inyección de información de la cuenta. Es utilizado también como vehículo de investigación para experimentos de aislamiento de red basados en los espacios de nombres (23).

2.2.2 Immunity

Es un *sandboxing* desarrollado sobre scripts de Python por Torsten Werner del proyecto Debian. Es notable por su minimalismo y por el uso de las capacidades del sistema de archivos (24).

2.2.3 NaCl (Google Native Client)

Este es un proyecto anunciado en diciembre de 2008 para definir una API y ABI. Con NaCl se puede compilar código nativo de forma segura por diversos clientes, que van desde los navegadores hasta las máquinas virtuales de Java del lado del servidor. Es notable en la comunidad de *sandboxing* por su apoyo corporativo, su competencia de seguridad, su activo desarrollo y sus resultados prometedores (25).

2.2.4 Plash

Es un *sandboxing* desarrollado sobre Python por Mark Seaborn y contribuyentes en diciembre de 2004. Su desarrollo es notable en que hace uso extensivo de las historias de usuario para guiar la creación de nuevas características. Es un *sandboxing* que proporciona un aislamiento a través de “chroot”, “setuid” y “ptrace” (26).

2.2.5 Selinux

Es un *sandbox* desarrollado sobre scripts de Python por Dan Walsh y colaboradores. Proporciona un aislamiento por la generación dinámica de la política de SELinux, la cual carga y aplica. Debido a su dependencia de SELinux no es tan ampliamente disponible como otras tecnologías *sandboxing*, sin embargo, por esta misma razón, puede ser capaz de lograr un mejor aislamiento (27).

2.2.6 Seccomp-nurse

Seccomp-nurse está basado en Seccomp y ha sido escrito por Nicolás Bareil en el año 2010. Su idea central es utilizar “prctl (PR_SET_SECCOMP)” para forzar un hilo cerrado para realizar llamadas al sistema a través de un hilo supervisor (28).

Seccomp es un *sandbox* ideado por Andrea Arcangeli en enero de 2005 y pensado originalmente para que los propietarios sistemas Linux pudieran alquilar sus computadoras a personas que realizaran trabajos de procesamiento graves. Seccomp coloca un *sandbox* alrededor del proceso que está siendo ejecutado por otro. Esto permite que un proceso realice una transición de un solo sentido dentro de un estado seguro en el que no se pueden realizar llamadas excepto: exit (), sigreturn (), read () y write (). En

el caso de que se intentara hacer alguna otra llamada, el núcleo terminaría inmediatamente el proceso (29).

2.2.7 Shikashi

Es un *sandboxing* de Ruby desarrollado por Dario Seminara bajo una licencia GPL. Maneja todos los métodos de Ruby que se llaman y ejecutan en el intérprete. Permite o deniega las llamadas en función del objeto receptor, el nombre del método, el archivo de origen desde donde se originó la llamada y el archivo de origen donde el método llamado es implementado. Los permisos para cada recinto de seguridad son totalmente configurables y la aplicación de los métodos llamados desde el interior del *sandbox* pueden ser sustituidos de forma transparente (30).

2.2.8 Bibliotecas sandbox (libsandbox & pysandbox)

Las bibliotecas (libsandbox & pysandbox) son una colección de *time-tested* (pruebas de tiempo). Son bibliotecas de código abierto para analizar las pruebas y perfiles de programas simples en un entorno restringido. Mediante estas bibliotecas, los comportamientos de tiempo y ejecución de los programas ejecutables binarios pueden ser capturados y bloqueados de acuerdo a las políticas configurables o programables. Estas bibliotecas fueron originalmente diseñadas y utilizadas como módulo de seguridad básico para el entrenamiento en un jurado en línea (HIT Online Judge) de la ACM / ICPC. Desde ese momento se convirtió en una herramienta de propósito general para la prueba de programas binarios, perfiles y restricciones de seguridad (31).

2.2.9 Sandboxie

Sandboxie es un *sandbox* desarrollado para Microsoft Windows y funciona como un sistema de virtualización por aislamiento. Este software aísla dentro de su contenedor los archivos de navegación y los de funcionamiento de las aplicaciones que controla. Este *sandbox* realiza una copia de una parte del registro de Windows para proteger el original contra las inserciones malévolas de claves y valores, esto permite que los programas ejecutados por medio de Sandboxie y los archivos descargados se encuentren encerrados en unas réplicas de los directorios normalmente utilizados. De esta forma cuando un programa se está ejecutando dentro de Sandboxie, las modificaciones efectuadas no afectan los archivos auténticos del sistema operativo. Sandboxie evita desde la zona que controla toda inyección en el núcleo de Windows (32).

2.3 Selección del sandbox

En el estudio realizado acerca del aislamiento de procesos se puede apreciar que existen una cantidad extensa de tecnologías de este tipo. Una gran parte de los *sandbox* están dirigidos al área de restringir procesos relacionados con archivos de sistema, acceso a disco, acceso a memoria, acceso a redes y acceso a registros; ejemplos de estos son el Immunity y el Plash. Algunos como NaCl y Selinux están completamente dirigidos al aislamiento de entornos web y máquinas virtuales. Existen casos también como el de Rainbow, el cual es orientado hacia el aislamiento de sesiones y cuentas de usuario. Por su parte el *sandbox* Seccomp-nurse presenta una estructura poco flexible y está dirigido al área del aislamiento de aplicaciones. Otros como las bibliotecas libsandbox & pysandbox están orientados al área de la programación en línea (de inglés *online*) y existen casos como el de Sandboxie que son desarrollados para sistemas Windows.

Las técnicas de *sandboxing* se dividen en diversas áreas. Las que pueden observarse en el estudio realizado son:

- Aislamiento de procesos en archivos de sistema, acceso a disco, acceso a redes, acceso a registros y acceso a memoria.
- Aislamiento de procesos en entornos web.
- Aislamiento de procesos en máquinas virtuales.
- Aislamiento de procesos en cuentas de usuario.
- Aislamiento de procesos en aplicaciones.
- Aislamiento de procesos en programación *online*.
- Aislamiento de procesos en ejecución de programas.

El desarrollo de un *sandbox* desde cero para el SCADA UX no debe considerarse una solución fiable por la complejidad que lleva implícito tal proceso. Existen muchos casos de intentos fallidos de desarrollar *sandbox* completamente fiable. Un ejemplo es el *sandbox* de la biblioteca estándar de Python, el cual tuvo que ser extraído de la misma; pues los desarrolladores no consiguieron estabilizar su proceso debido al extremo aislamiento que se quería alcanzar.

Por lo estudiado hasta el momento, la mayor parte de los *sandbox* investigados se pueden descartar como una solución aplicable al procesamiento de puntos calculados del SCADA UX. Por tanto, se decide evaluar el *sandbox* Shikashi, el cual está orientado al aislamiento de procesos en la ejecución de

programas, lo cual representa el área que interesa en este trabajo. Este mecanismo está desarrollado en lenguaje Ruby y está orientado al aislamiento de dicho lenguaje.

Para aplicar esta solución, primero se debe realizar un estudio acerca de la forma de embeber el lenguaje Ruby en el lenguaje C/C++, buscando mantener un rendimiento acorde al procesamiento de puntos calculados del SCADA UX.

2.4 Embebido de lenguaje

Embeber un lenguaje dentro de otro es el hecho de incrustar código de un lenguaje dentro de otro. Usualmente se emplea para usar las potencialidades de un lenguaje scripting dentro de un lenguaje compilado o viceversa. Esto permite usar bibliotecas de otros lenguajes diferentes al que se está empleando. Para realizar esto casi todos los lenguajes ofrecen diversas API que permiten integrarlos fácilmente con otros lenguajes.

A continuación se ofrece una breve reseña del lenguaje Ruby, del cual se estudiará su embebido en C/C++ en el presente epígrafe.

2.4.1 Lenguaje de programación Ruby

Ruby es un lenguaje que incorporara tanto la programación funcional como la programación imperativa. Es un lenguaje de programación interpretado, reflexivo, flexible y orientado a objetos. Este lenguaje tiene un conjunto de funcionalidades importantes entre las cuales se encuentra la de incluir llamadas para embeber Ruby en otros programas, y así usarlo como lenguaje de scripting. También posee manejo de hilos (del inglés *threading*) independiente del sistema operativo. Ruby tiene derechos de autor de software libre por Yukihiro Matsumoto. Puede redistribuirse y/o modificarse bajo cualquiera de los términos de la licencia GPL. Ruby es un lenguaje multiparadigma y soporta tipado dinámico (33).

2.4.2 Embeber Ruby en C/C++

La versión de Ruby que fue utilizada para realizar el empotramiento del lenguaje en C/C++ fue la 1.8. Se comprobó el correcto funcionamiento de este lenguaje mediante aplicaciones demostrativas, las cuales serán descritas a continuación. Para la realización de estos ejemplos se incluyó la biblioteca “ruby.h” en la cabecera del proyecto C/C++.

Ejemplo 1:

En este ejemplo se leyó y ejecutó un fichero .rb desde una aplicación de C/C++. Para esto se utilizaron las funciones que brinda la biblioteca de Ruby:

ruby_init (): Permite inicializar el intérprete.

ruby_init_loadpath (): Inicializa una variable que contiene el camino que se consulta para cargar ficheros externos a la aplicación.

rb_load_protect (): Permite realizar llamadas a un fichero externo y atrapar los errores de Ruby para asegurarse de que no se bloquee el proceso si se encuentra algún error en el script Ruby.

ruby_finalize (): Finaliza el intérprete de Ruby.

Ejemplo 2:

En este ejemplo se creó y escribió un fichero .rb desde una aplicación de C/C++ de forma estática con la función “fprintf ()”, el cual posteriormente fue leído y ejecutado desde la propia aplicación. Se utilizaron las funciones descritas en el Ejemplo 1 que brinda la biblioteca de Ruby.

Ejemplo 3:

En el ejemplo anterior se dificultaba escribir el fichero .rb estáticamente con la función “fprintf ()”, esto significaba un problema y se requería que el trabajo fuera más natural y sencillo. Tampoco se podía leer de la consola con el operador “cin” de C++ ya que este ignoraba todos los espacios al inicio del fichero y se requería que estos ficheros mantuvieran la sintaxis de su lenguaje. Por lo tanto la solución fue adoptar las funciones “read” y “write”, para leer de la consola y escribir en el fichero respectivamente. Luego de realizar dichas operaciones correctamente se procedió a ejecutar el fichero .rb y se utilizaron las funciones mencionadas en los ejemplos anteriores.

Ejemplo 4:

Se requería que el resultado de una función ejecutada en el fichero .py se pudiera leer desde la aplicación en C++ para su manejo posterior en otras acciones. Para esto se desarrolló un ejemplo en el que se pasaron valores desde C++ a una función escrita en un fichero .py. Luego, el valor que devolvió dicha función fue recogido y almacenado en una variable de C++ para su posterior uso. Se utilizaron las funciones mencionadas en los ejemplos anteriores y se agregaron algunas más que brinda la biblioteca de Python:

PyImport_AddModule (): Permite devolver el objeto del módulo correspondiente a un nombre de módulo.

PyModule_GetDict (): Permite devolver el objeto diccionario que implementa el espacio de nombres del módulo.

PyDict_New (): Devuelve un diccionario vacío en caso de ocurrir algún fallo.

PyDict_SetItemString (): Introduce un valor en el diccionario de Python utilizando la clave como una clave.

PyInt_AsLong (): Convierte el objeto a un objeto de Python si este no está convertido y luego devuelve su valor.

PyDict_GetItemString (): Devuelve un objeto del diccionario de Python que tiene una clave fundamental.

PyRun_File (): Se trata de otra interfaz simplificada de PyRun_SimpleFileExFlags (), la cual permite ejecutar código fuente desde un fichero definido.

En la Figura 10 de los anexos se muestra un ejemplo del código utilizado para embeber un script de Ruby en C/C++.

2.5 Pruebas de rendimiento sobre el sandbox

Para sistemas de tiempo real y sistemas empotrados, es inaceptable el software que proporciona las funciones requeridas pero no se ajusta a los requisitos de rendimiento. La prueba de rendimiento está diseñada para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado. Incluso a nivel de unidad, se debe asegurar el rendimiento de los módulos individuales (34).

Luego de realizar las aplicaciones demostrativas para empotrar código del lenguaje Ruby en C/C++, se procedió a integrar el *sandbox* Shikashi a un *script* de Ruby para realizar pruebas de rendimiento sobre el mismo. Para realizar estas pruebas se utilizó la función “gettimeofday ()” de C++ para medir los tiempos de ejecución y el comando “ps” de Linux para capturar el porciento de uso de memoria y procesador. Dichas pruebas parten desde una menor complejidad y aumentan poco a poco hasta alcanzar una complejidad superior. Cada prueba se realizó un número de 10 veces para determinar con mejor exactitud los valores mínimos y máximos críticos que arrojaba el sistema. A continuación se describen las mismas y sus resultados.

Prueba #1:

Se implementó un ciclo “for” desde 1 hasta 1000 iteraciones. Dentro del ciclo se calculaba el valor de la raíz de una variable “x”, multiplicada por el seno de una variable “y” dividida por el coseno de una variable “z”, todo esto dividido finalmente por una constante con un valor real. La forma de la ecuación utilizada era la siguiente $(\text{sqrt } x * \frac{\sin y}{\cos z}) / constante$. En la siguiente tabla se muestra el comportamiento del sistema para esta prueba:

Tabla 2. Resultado de prueba de rendimiento #1

Valores	Tiempo de ejecución (segundos)	Uso de memoria (%)	Uso de CPU (%)
Mínimo	0.3	0.7	0.0
Máximo	0.4	0.7	0.0

Prueba #2:

En esta prueba se realizó el mismo procedimiento de la prueba #1, pero el ciclo “for” se ejecutó desde 1 hasta 10000 iteraciones. En la siguiente tabla se muestran las respuestas del sistema para esta prueba:

Tabla 3. Resultado de prueba de rendimiento #2

Valores	Tiempo de ejecución (segundos)	Uso de memoria (%)	Uso de CPU (%)
Mínimo	2.0	0.7	92.0
Máximo	2.1	0.7	89.0

Prueba #3:

En esta prueba se realizó el mismo procedimiento de la prueba #1 y #2, pero el ciclo “for” se ejecutó desde 1 hasta 100000 iteraciones. En la siguiente tabla se muestran las respuestas del sistema para esta prueba:

Tabla 4. Resultado de prueba de rendimiento #3

Valores	Tiempo de ejecución	Uso de memoria (%)	Uso de CPU (%)

	(segundos)		
Mínimo	14.8	0.7	99.8
Máximo	18.5	0.7	99.5

Prueba #4:

En esta prueba se realizó el mismo procedimiento de las pruebas anteriores, pero el ciclo "for" se ejecutó desde 1 hasta 1000000 iteraciones. En la siguiente tabla se muestran las respuestas del sistema para esta prueba:

Tabla 5. Resultado de prueba de rendimiento #4

Valores	Tiempo de ejecución (segundos)	Uso de memoria (%)	Uso de CPU (%)
Mínimo	179.1	0.7	101.0
Máximo	186.1	0.7	99.8

Las pruebas de rendimiento aplicadas muestran que el *sandbox* embebido en lenguaje C/C++ mediante un script de Ruby presenta tiempos de ejecución acordes con los requeridos por el módulo de procesamiento del SCADA UX.

2.6 Pruebas de seguridad sobre el sandbox

"La prueba de seguridad comprueba que los mecanismos de protección integrados en el sistema realmente lo protejan de irrupciones inapropiadas.

"Por supuesto que debe probarse la seguridad del sistema para asegurad que es invulnerable a los ataques frontales, pero también a los perpetrados por los flancos o la retaguardia." [Beizer]

Durante la prueba de seguridad, quien la aplica desempeña el papel del individuo que desea entrar en el sistema. ¡Todo se vale! Debe tratar de obtener contraseñas por cualquier medio externo; podría atacar el sistema con software personalizado, diseñado para burlar cualquier defensa que se haya construido; podría saturar el sistema, negando así el servicio a otros; podría producir errores intencionales en el sistema para tratar de tener acceso durante la recuperación; podría revisar datos sin protección, con la idea de encontrar la clave de acceso al sistema.

Si se dan el tiempo y los recursos suficientes, una buena prueba de seguridad terminará por irrumpir en el sistema. El papel del diseñador del sistema es que el costo de la irrupción sea mayor que el valor de la información que habrá de obtenerse.” (34)

A continuación se describen las pruebas de seguridad aplicadas sobre el *sandbox* Shikashi. Se probaron vulnerabilidades del sistema tanto fuera como dentro del *sandboxing*, el cual se encontraba incluido en un script de Ruby embebido en C/C++ de la forma en que se demostró en el sub-epígrafe 2.4.2.

Para realizar estas pruebas se escribieron varias instrucciones (system) que agreden directamente al sistema. Las mismas se escribieron primeramente dentro y luego fuera del *sandboxing* para observar, estudiar y comparar los resultados. En la siguiente tabla se muestran los resultados de estas pruebas:

Tabla 6. Resultados de pruebas de seguridad con el sandbox Shikashi

Instrucción	Descripción	Dentro del sandboxing	Fuera del sandboxing
system('ls -l')	Permite que se listen todos los archivos referentes al directorio en el que se ejecuta la aplicación.	La instrucción no se ejecuta.	La instrucción se ejecuta y devuelve el resultado: total 44 drwx------ 3 gaara gaara 4096 Jun 4 15:34 Debug -rw-r--r-- 1 gaara gaara 1045 Jun 1 15:58 declarations.rb -rw-r--r-- 1 gaara gaara 436 May 15 23:41 eficiencia.rb -rw-r--r-- 1 gaara gaara 435 May 15 23:31 eficiencia.rb~ -rw-r--r-- 1 gaara gaara 97 Apr 28 17:57 nuevo.rb -rw-r--r-- 1 gaara gaara 65 May 26

			18:03 ruby.rb~ -rw-r--r-- 1 gaara gaara 423 Jun 4 15:35 sandbox.rb -rw-r--r-- 1 gaara gaara 314 May 29 22:33 sandbox.rb~ drwx------ 2 gaara gaara 4096 Apr 23 22:03 src -rw-r--r-- 1 gaara gaara 0 Apr 25 22:33 test~ -rw-r--r-- 1 gaara gaara 103 May 7 11:04 test.rb -rw-r--r-- 1 gaara gaara 72 May 7 10:56 test.rb~
system('mkdir /home/gaara/Desktop/prueba')	Permite crear una carpeta en el directorio especificado.	La instrucción no se ejecuta.	Se crea en el escritorio (Desktop) una carpeta de nombre “prueba”.
system('rm -R /home/gaara/Desktop/prueba')	Permite eliminar una carpeta del directorio especificado.	La instrucción no se ejecuta.	Se elimina del escritorio (Desktop) la carpeta de nombre “prueba”.
system('ls -l /root/')	Permite que se listen todos los archivos referentes al directorio root.	La instrucción no se ejecuta.	La instrucción se deniega e informa: ls: cannot open directory /root/: Permission denied
system('sudo ls -l /root/')	Permite que se listen todos los archivos	La instrucción no se ejecuta.	Al ejecutar la instrucción se informa:

	referentes al directorio root ejecutando la directiva a través de permisos de administración.		[sudo] password for gaara: Al introducir la contraseña (password) para sudo, el cual pudiera forzarse por medio de cualquier técnica de hacking aparece la información correspondiente al comando: total 4 drwxr-xr-x 2 root root 4096 Jun 1 12:18 Desktop
system('sudo shutdown -r now')		La instrucción no se ejecuta.	Al ejecutar la instrucción se informa: [sudo] password for gaara: Como el caso anterior, al introducir la contraseña se reinicia el sistema operativo.

Las pruebas de seguridad demuestran que el *sandbox* Shikashi presenta una estructura que permite aislar de manera segura los procesos que se deseen ejecutar sobre una aplicación que lo integre. Las pruebas de seguridad expuestas en este epígrafe, así como las pruebas de rendimiento mostradas en el epígrafe 2.5, se realizaron en un ordenador con las siguientes características:

Procesador: Intel(R) Core(TM)2 Duo CPU E4500 @ 2.20GHz.

Memoria RAM: 1Gb.

Sistema Operativo: Debian 6.0 Squeeze.

2.7 Ambiente de desarrollo

A continuación se describen las herramientas y tecnologías utilizadas en el presente trabajo. Dichas tecnologías pertenecen a la categoría de software libre y con el uso de las mismas se está siendo consecuente con las políticas trazadas por el Centro de Informática Industrial.

2.7.1 Sistema Operativo GNU/Linux

GNU/Linux es un sistema operativo que ha sido desarrollado bajo los principios del software libre. Se selecciona el sistema operativo GNU/Linux, en su distribución Debian 6.0 Squeeze por ser un sistema operativo robusto, estable y rápido, que por su característica de ser software libre tiene un costo cero o muy bajo y disponibilidad del código fuente.

2.7.2 Lenguajes de programación

Se utilizan los lenguajes de programación C/C++ y Ruby. C/C++ es el lenguaje utilizado para el desarrollo de los proyectos SCADA del CEDIN. Por otra parte el lenguaje Ruby es el que permite integrar el *sandbox* Shikashi al módulo de procesamiento del SCADA UX.

2.7.3 Metodología de desarrollo AUP

Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos. Imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente (35).

El Proceso Unificado Ágil (AUP, por sus siglas en inglés, *Agile Unified Process*) es una metodología de procesos de desarrollo simplificada del Proceso Unificado de Rational (RUP, por sus siglas en inglés, *Rational Unified Process*). Mediante AUP se describe un método sencillo, fácil de entender para el desarrollo de software de aplicaciones empresariales utilizando técnicas ágiles y conceptos que permanecen fieles a la RUP.

Esta metodología se basa en los siguientes principios (36)**:**

- El personal que desarrolla sabe lo que está haciendo.
- Todo se describe de manera concisa generando solamente la documentación necesaria.
- La atención se centra en las actividades que realmente cuentan, no en todas las cosas posibles que pueden suceder en el proyecto.
- Se pueden utilizar cualquier conjunto de herramientas que se desee con AUP.
- AUP es fácilmente adaptable a las necesidades de cada proyecto.

A continuación se muestra el ciclo de vida que describe AUP:

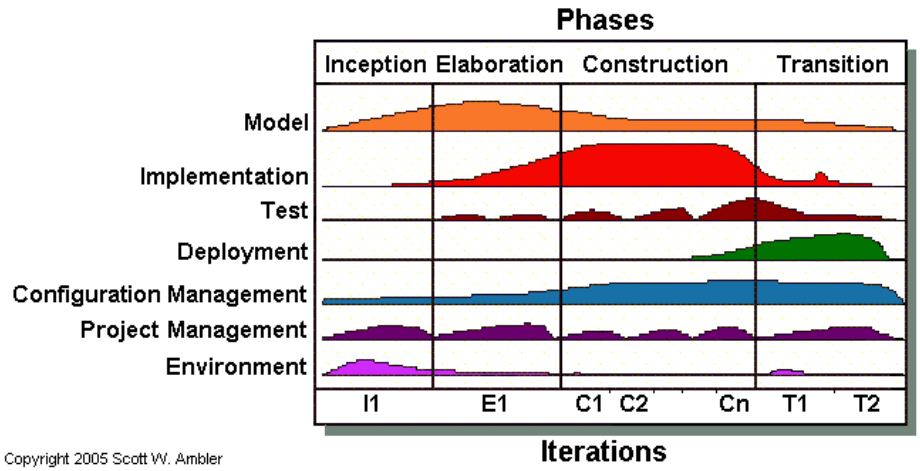

Figura 5. Ciclo de vida de AUP.

2.7.4 Herramienta CASE

CASE (Ingeniería de Software Asistida por Computadora o en inglés *Computer Aided Software Engineering*) es una filosofía que se orienta a la mejor comprensión de los modelos de empresa, sus actividades y el desarrollo de los sistemas de información. Esta filosofía involucra además el uso de programas que permiten (37):

- Construir los modelos que describe la empresa.
- Describir el medio en el que se realizan las actividades.
- Llevar a cabo la planificación.

Visual Paradigm:

Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas y generar documentación. Algunas características de esta herramienta son:

- Posibilita la creación de diagramas de flujo de datos.
- Posee distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.

2.7.5 Lenguaje de modelado UML

El lenguaje de modelado es la notación (principalmente gráfica) utilizada para expresar un diseño, el proceso indica los pasos que se deben seguir para llegar a este (38).

El Lenguaje de Modelado Unificado (UML por sus siglas en inglés, *Unified Modeling Language*) es un lenguaje para la especificación, visualización, construcción y documentación de los artefactos de un proceso. Este método surgió de la combinación de ideas de Grady Booch, James Rumbaugh, e Ivar Jacobson (34).

2.7.6 Entorno de desarrollo Eclipse

Un entorno de desarrollo integrado (IDE por sus siglas en inglés, *Integrated Development Environment*) es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de éstos.

Eclipse es un IDE de desarrollo de software de código fuente abierto, cuyo objetivo es la construcción de herramientas integradas para el desarrollo de aplicaciones. Es neutral y adaptable a cualquier tipo de lenguaje, por ejemplo C/C++, Cobol, C#, XML, entre otros (39).

2.8 Conclusiones parciales

El sistema operativo, metodología, herramientas, lenguajes y tecnología a utilizar para el desarrollo de la solución son:

- Debian 6.0 Squeeze
- AUP
- Visual Paradigm
- Eclipse
- UML
- C/C++
- Ruby
- Shikashi

Capítulo 3: Análisis y diseño de la solución

3.1 Introducción

En el presente capítulo se expone el estado actual del procesamiento de puntos calculados del SCADA UX. Se muestran los modelos asociados con la solución. Se describen y detallan los requisitos que debe cumplir finalmente la solución propuesta.

3.2 Modelo del dominio

Un modelo del dominio captura los tipos más importantes de objetos que existen o los eventos que suceden en el entorno donde estará el sistema. Un modelo del dominio es una representación de las clases conceptuales del mundo real, no de componentes de software. Este tipo de modelo tiene como objetivo principal ayudar a comprender los conceptos con los que debe trabajar nuestra aplicación.

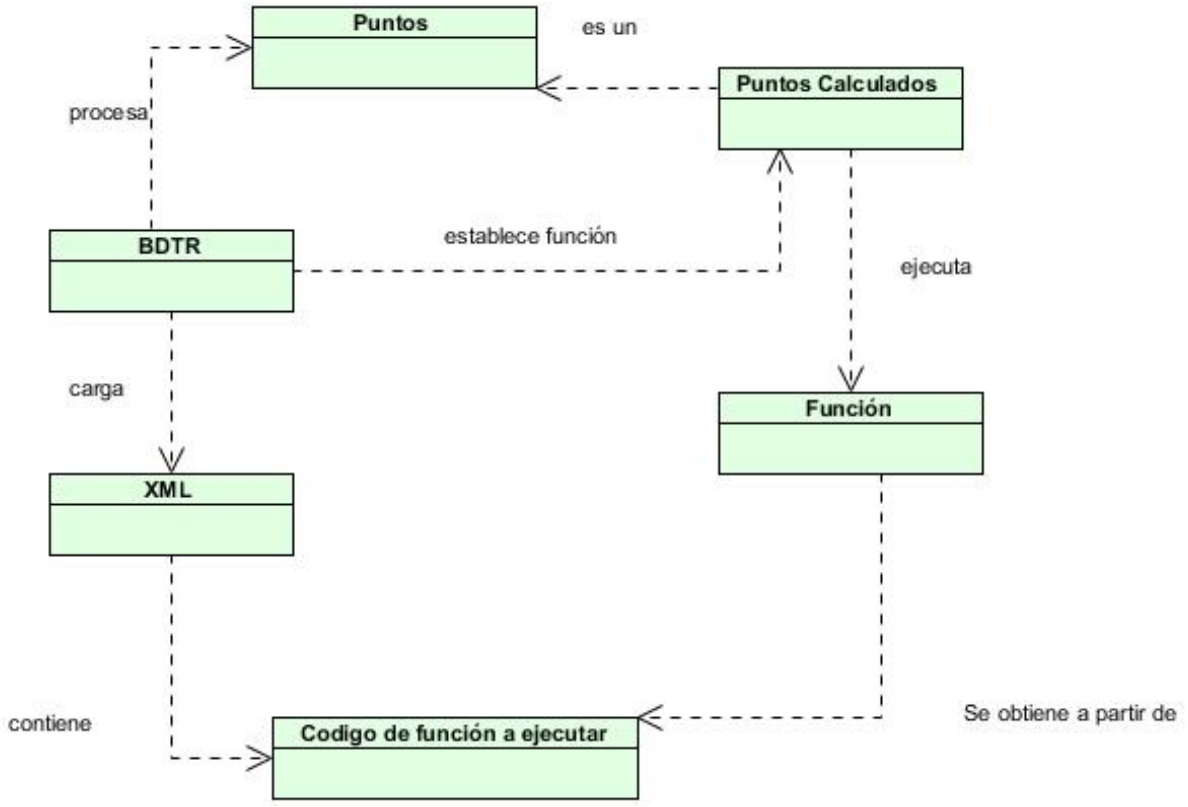

Figura 6. Modelo del dominio.

A continuación se expone una descripción de los elementos que integran el modelo del dominio.

BDTR: módulo de procesamiento del SCADA UX. En este se cargan las configuraciones establecidas por el archivo XML y se procesa la información obtenida de los medidores de campo. Una de las tareas que lleva a cabo el mismo es el procesamiento de puntos.

XML: archivo que se genera mediante el HMI cliente (o de edición) y que es cargado por el módulo de procesamiento, el cual lo utiliza para establecer las configuraciones pertinentes. Ejemplos de las configuraciones que brinda este archivo son: los identificadores de los puntos y las funciones de cálculo para los puntos calculados.

Código de función a ejecutar: segmento de código escrito en lenguaje C/C++ presente en el archivo XML el cual contiene la función de cálculo para un punto calculado.

Puntos: entidad que se procesa en la BDTR. Constituyen valores simples de entrada y (o) salida que son supervisados y (o) controlados por el sistema SCADA.

Puntos calculados: tipo de puntos, los cuales residen en el módulo de procesamiento. Son variables que se obtienen mediante la evaluación de fórmulas matemáticas.

Función: función de cálculo en C/C++ que se obtiene a partir del código establecido por el archivo XML. Un punto calculado ejecuta una función determinada para obtener su valor.

3.3 Definición de requisitos

Requisitos no funcionales:

Los requisitos no funcionales del módulo de procesamiento se encuentran presentes en el documento 0113_Especificación de Requisitos de Software de la línea de procesamiento. Además de los allí descritos se agrega como requisito no funcional de la solución la utilización del *sandbox* Shikashi y el lenguaje Ruby embebido en C/C++ (40).

Requisitos funcionales:

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera que este debe reaccionar a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos estos requisitos pueden declarar de forma explícita lo que el sistema no debe hacer (37).

El requisito funcional “Procesar punto calculado” se encuentra afectado como parte del desarrollo de la solución. El mismo se puede consultar en el documento 0113_Especificación de Requisitos de Software de la línea de procesamiento (40).

A partir de dicho requisito parten los requisitos que se describen a continuación. Los mismos cuentan con una complejidad de desarrollo media y una prioridad alta.

Tabla 7. Definición del requisito: Cargar configuración de punto calculado

Nº	Nombre	Descripción
RF1	Cargar configuración de punto calculado	La función de cálculo se escribe en lenguaje Ruby con algunas modificaciones como se describe a continuación: <código de Ruby> def @nombre de la función() <operación> end Antes de definir la función principal que será invocada para la obtención del punto calculado se pueden declarar otras funciones para apoyar dicho cálculo. En el ejemplo anterior se evidencia en el segmento representado como “<código de Ruby>”. La función principal contiene el signo de “@” delante del “nombre de la función” y termina con el signo “()”. En el cuerpo de la función, representado por el identificador “<operación>” en el ejemplo expuesto, se escriben las operaciones a realizar por la función. Dentro del cuerpo de una función se pueden involucrar otros puntos simples o calculados siempre que no se establezca un ciclo recursivo sobre la misma función. Estos puntos se representan con el signo de “$” seguidos por su nombre y terminan igualmente con el signo de “$”.

Tabla 8. Definición del requisito: Obtener punto calculado

Nº	Nombre	Descripción
RF2	Obtener punto calculado	En el módulo de procesamiento se carga el fichero .rb y se ejecuta la función que contiene mediante la API de Ruby. En la Figura 10 de los Anexos se puede observar un ejemplo del código utilizado para realizar estas operaciones.

3.4 Diagrama de clases del diseño

El diagrama de clases captura la estructura lógica del sistema y las clases que constituyen el modelo. Es un modelo estático, describiendo lo que existe y qué atributos y comportamientos tiene. Los diagramas de clases son los más útiles para ilustrar las relaciones entre las clases e interfaces.

En el siguiente diagrama de clases se muestran las que tienen un impacto en el diseño de la solución final a partir de los requisitos definidos. Solamente fue necesario realizar variaciones en el diseño de la clase CalculusInformant, en la cual se modificaron dos atributos.

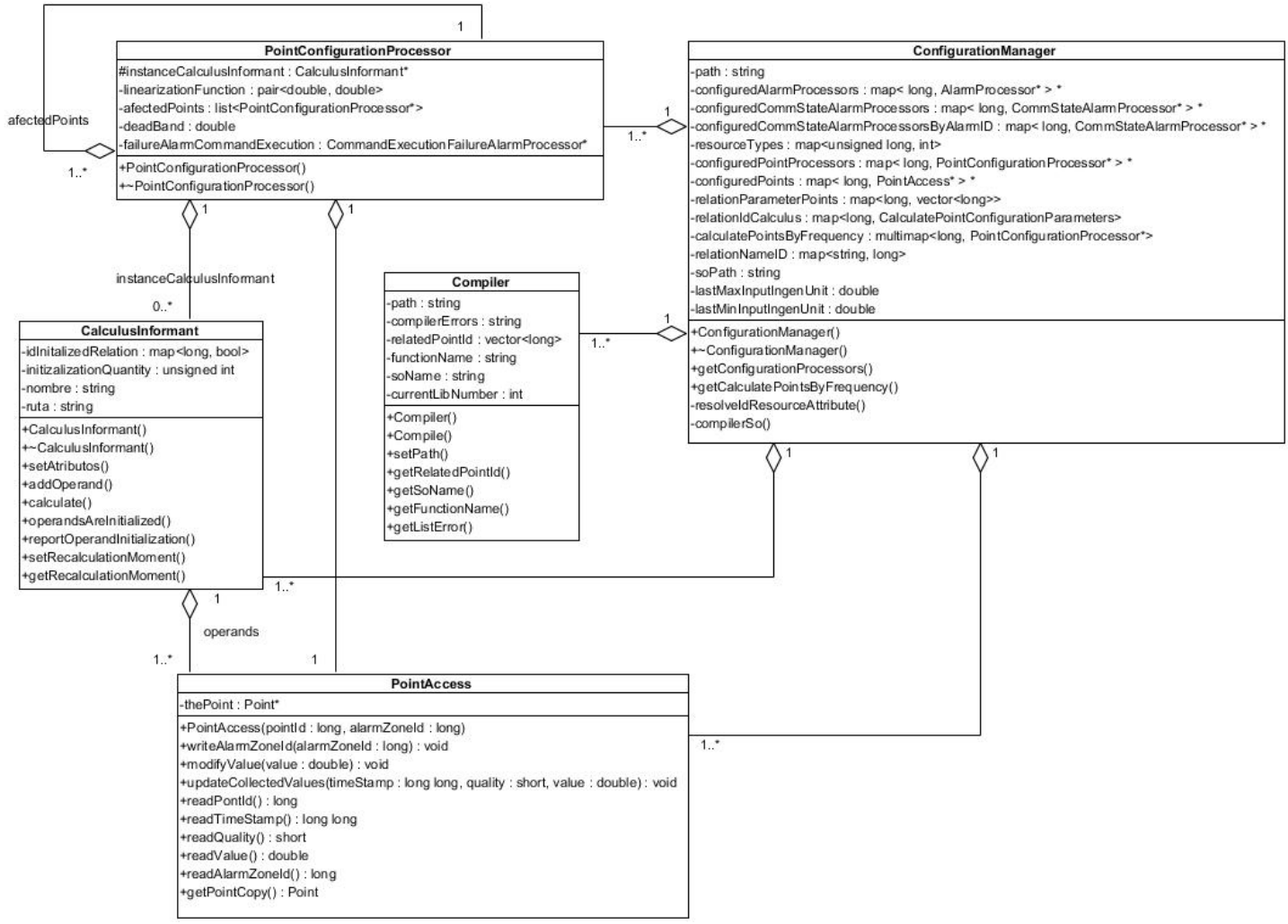

Figura 7. Diagrama de clases del diseño.

3.4.1 Descripción de las clases

ConfigurationManager tiene como propósito cargar las configuraciones definidas por el archivo XML. Es la encargada de crear las estructuras correspondientes al módulo BDTR, estableciéndoles los parámetros de configuración cargados desde el XML. En el presente trabajo es necesario el uso de las estructuras **Compiler** y **CalculusInformant**, con las que la clase **ConfigurationManager** establece una relación directa. Esta clase cuenta con el método “compileSo” el cual es el encargado de definir los valores correspondientes a los métodos “Compile” y “setAtributos”, los cuales son utilizados para la construcción y localización del archivo que contiene la función para la obtención de un punto calculado.

Compiler tiene como propósito construir el archivo de Ruby (.rb) con la función de cálculo asignada para un punto calculado. El archivo generado debe tener incluida la integración con el *sandbox* Shikashi. El método principal de esta clase es el método “Compile”, en el cual se llevan a cabo las acciones anteriormente descritas. El mismo es invocado a través de un objeto de la clase **Compiler** desde **ConfigurationManager**.

CalculusInformant tiene como propósito la implementación de los métodos que se utilizan para ejecutar el archivo generado por la clase **Compiler**. El método “setAtributos” se utiliza para establecer el nombre de la función a ejecutar y la ruta en la que se encuentra el fichero para su ejecución. Por su parte el método “calculate” es el encargado de establecer los valores de los puntos relacionados con el cálculo de una función y ejecutar el fichero para devolver el valor de la operación. El método “calculate” se invoca a través de un objeto de la clase **CalculusInformant** desde **PointConfigurationProcessor**.

PointConfigurationProcessor es la encargada de obtener el valor de un punto calculado y actualizar la clase **PointAccess**.

PointAccess contiene información relevante para un punto, como por ejemplo el identificador, el valor y la estampa de tiempo en que dicho valor fue actualizado.

3.4.2 Descripción de las relaciones entre las clases

Como se puede observar en el diagrama de clases expuesto, existe una fuerte relación entre las clases afectadas por la solución y las demás clases presentes en el procesamiento de un punto calculado. La clase **ConfigurationManager** es la encargada de establecer los valores de configuración a las demás estructuras relacionadas con el módulo BDTR. La clase **PointConfigurationProcessor** se encarga de invocar el método “calculate” de la clase **CalculusInformant** para obtener el valor de un punto calculado y es la encargada de actualizar la clase **PointAccess** la cual contiene la información de los puntos. La clase **ConfigurationManager** es la encargada de establecer los valores correspondientes a las clases **Compiler** y **CalculusInformant**, las cuales son las encargadas de construir y ejecutar el archivo que contiene la función para la obtención de un punto calculado. También la clase **ConfigurationManager** es la que invoca al método “Compile” de la clase **Compiler** para indicar la construcción del archivo anteriormente mencionado.

3.5 Conclusiones parciales

Los requisitos funcionales “Cargar configuración de punto calculado” y “Obtener punto calculado” guiarán el desarrollo de la solución. Su implementación tendrá un impacto directo en las clases CalculusInformant y Compiler del módulo de procesamiento.

Capítulo 4: Implementación y pruebas

4.1 Introducción

En este capítulo se presentan los diagramas referentes al diseño de la solución, así como el estilo de código utilizado para el desarrollo de la misma. Se realizan pruebas al requisito funcional obtenido en el capítulo anterior para validar el funcionamiento correcto del mismo. Se realiza una valoración del resultado obtenido, tomando en cuenta la investigación realizada y la integración de la solución.

4.2 Diagrama de componentes

Un diagrama de componentes ilustra las piezas del software, controladores embebidos, entre otros, que conformarán un sistema. Un diagrama de componentes tiene un nivel más alto de abstracción que un diagrama de clases. Usualmente un componente se implementa por una o más clases (u objetos) en tiempo de ejecución (41).

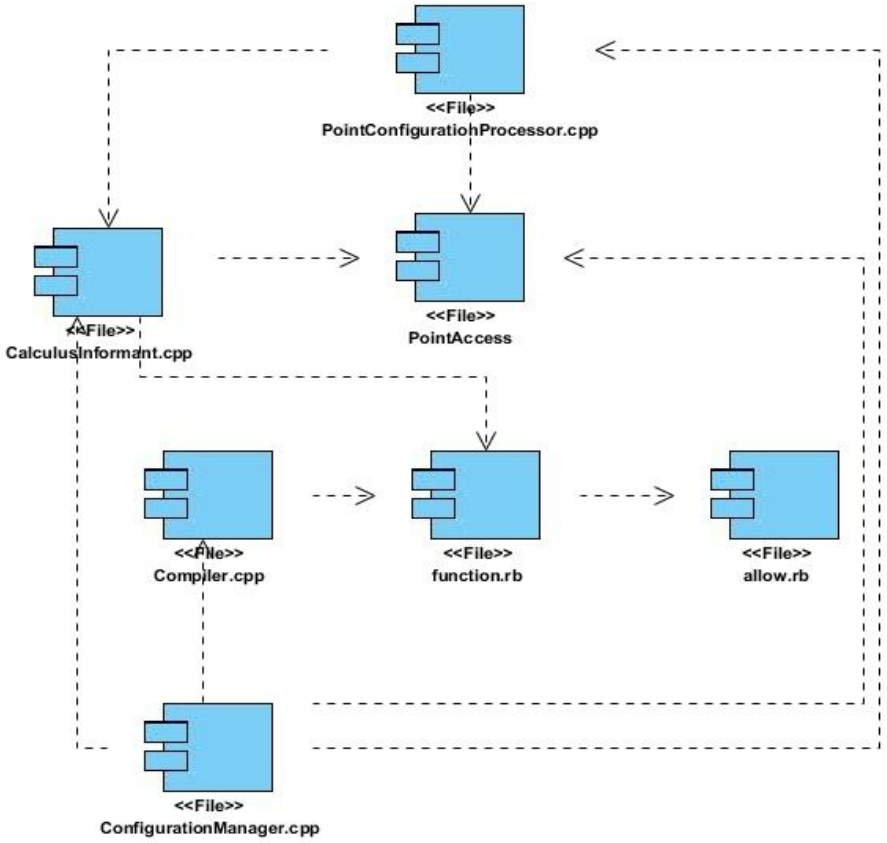

Figura 8. Diagrama de componentes.

En el diagrama de componentes expuesto se muestran los componentes utilizados en la solución y sus relaciones. Se actualizaron los componentes Compiler.cpp y CalculusInformant.cpp y se agregaron dos nuevos componentes identificados como function.rb y allow.rb. El componente ConfigurationManager.cpp utiliza el componente Compiler.cpp para generar al componente function.rb, el cual contiene el entorno controlado por el *sandbox* Shikashi y dentro de dicho entorno se encuentra la función para la obtención del valor de un punto calculado. El componente CalculusInformant.cpp es el encargado de ejecutar el componente function.rb generado por el componente Compiler.cpp y proporcionar el valor del cálculo al componente PointConfigurationProcessor.cpp. El componente function.rb utiliza al componente allow.rb para definir los métodos que se desean utilizar dentro del entorno controlado por el *sandbox* Shikashi, ya que el mismo deniega por defecto la ejecución de todo método.

En la Figura 11 y la Figura 12 de los Anexos se muestran ejemplos de los componentes function.rb y allow.rb respectivamente.

4.3 Estilo de código

El objetivo de un estilo de código es el de fomentar el uso de buenas prácticas de programación. Además de esto con él se pretende educar al equipo de desarrollo y definir estándares para mejorar la comunicación del equipo de trabajo y la gestión del conocimiento en general (42).

El estilo de código utilizado para integrar la solución es el que establece el Centro de Informática Industrial, registrado en el informe 0120_1 Estándares de codificación para C++ del Programa de Mejora del Proyecto SCADA Guardián del ALBA (43).

4.4 Pruebas

“La prueba de software es un elemento de un tema más amplio que suele denominarse verificación y validación. Verificación es el conjunto de actividades que aseguran que el software implemente correctamente una función específica. Validación es un conjunto de actividades que aseguran que el software construido corresponde con los requisitos del cliente.

En resumen:

Verificación: ¿Estamos construyendo el producto correctamente?

Validación: ¿Estamos construyendo el producto correcto?” (34)

Las pruebas de caja negra, también denominada prueba de comportamiento, se centran en los requisitos funcionales del software. O sea, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa (34).

4.4.1 Descripción de los casos de prueba

A continuación se describen los casos de prueba (CP) de caja negra elaborados para los requisitos funcionales RF1 y RF2, con los cuales se comprueba el correcto funcionamiento de la solución bajo distintas condiciones a las que puede someterse. Ambos requisitos se probaron en conjunto, ya que no cumple objetivo realizar pruebas por separado a los mismos.

La condición de ejecución para estos casos de prueba es común. Esta condición está dada por el momento del cálculo del punto, el cual se estableció en modo periódico a 5 segundos.

La entrada de las variables se realiza a través de la aplicación ModSim, la cual se utiliza para simular los valores que puede tomar un punto.

Tabla 9. Descripción de casos de prueba

Caso de prueba	Descripción	Entrada	Resultado esperado
CP1	Se verifica el cálculo: $analogico1$ + 10	Se estableció el valor de 20 al punto $analogico1$, luego 30 y así sucesivamente hasta 50, realizando un incremento de 10.	Se espera que el cálculo devuelva el valor del punto $analogico1$ incrementado a 10.
CP2	Se verifica el cálculo: sqrt ($analogico1$) + 4	Se estableció el valor de 16 al punto $analogico1$.	Se espera que el cálculo devuelva el valor de 8.
CP3	Se verifica el cálculo: $analogico1$ + $analogico2$	Se establecieron valores aleatorios a los puntos $analogico1$ y $analogico2$.	Se espera que el cálculo devuelva el valor de la suma de ambos puntos.

CP4	Se verifica el cálculo: (sqrt ($analogico1$) - $analogico2$) / 5	Se estableció el valor de 100 al punto $analogico1$ y el valor de 5 al punto $analogico2$.	Se espera que el cálculo devuelva el valor de 1.
CP5	Se verifica el cálculo: sqrt ($analogico1$ + $analogico2$) / sin ($analogico2$)	Se estableció el valor de 10 al punto $analogico1$ y el valor de 90 al punto $analogico2$.	Se espera que el cálculo devuelva el valor de 10.

4.4.2 Resultado de las pruebas

El resultado de las pruebas realizadas fue satisfactorio para todos los casos. Esto permite comprobar que el funcionamiento de la solución es correcto y que se pueden ejecutar cálculos con los niveles de complejidad requeridos para las funciones de puntos calculados.

4.5 Discusión de resultados

A partir del estudio realizado en el presente trabajo, la integración realizada al SCADA UX y las pruebas aplicadas sobre la solución obtenida, se realizan las siguientes valoraciones:

Es importante mantener tiempos bajos de ejecución en las funcionalidades que se realizan con una alta frecuencia en el módulo de procesamiento. Tal es el caso de la obtención de puntos calculados.

La integración del *sandbox* Shikashi a través del lenguaje Ruby embebido en C/C++ permite mantener tiempos de ejecución acordes a las necesidades del módulo. Esto quedó demostrado a través de las pruebas de rendimiento realizadas en el Capítulo 2 a esta tecnología. También dicha integración permite conservar la expresividad y flexibilidad necesarias en la definición de las funciones para la obtención de los puntos calculados.

Utilizando el *sandbox* Shikashi se limitan las instrucciones posibles a emplear en la programación de las funciones para la obtención de los puntos calculados y de esta forma se logra el aislamiento necesario para eliminar la brecha de seguridad relacionada con los ataques de explotación de errores en el módulo de procesamiento del SCADA UX.

4.6 Conclusiones Parciales

La solución obtenida satisface los requisitos funcionales “Cargar configuración de punto calculado” y “Obtener punto calculado”, definidos para el desarrollo del presente trabajo.

Conclusiones

Al término de la presente investigación es posible arribar a las siguientes conclusiones:

- La utilización de *sandbox* en el procesamiento de puntos calculados en sistemas SCADA constituye una técnica novedosa.
- El aislamiento de la ejecución de programas escritos en Ruby y embebidos en C/C++ utilizando el *sandbox* Shikashi proporciona un entorno controlado con opciones de seguridad y bajos tiempos de respuesta.
- La solución obtenida evita los ataques de explotación de errores en el procesamiento de puntos calculados del SCADA UX.

Recomendaciones

Con el objetivo de mejorar la solución propuesta en el presente trabajo se recomienda:

- Realizar los cambios pertinentes en el módulo HMI Edición para que sea posible establecer de forma visual los nuevos parámetros que necesita la solución.
- Diseñar el mecanismo visual de configuración del *sandbox* con el fin de definir el nivel de seguridad requerido para una implantación determinada.
- Mejorar la solución agregando un nuevo nivel de seguridad que permita evitar los ataques de “Jamming o Flooding”.

Referencias bibliográficas

1. **Special Interest Group (SIG).** Sandboxing.org. *Sandboxing.org.* [En línea] http://sandboxing.org/?page_id=103.

2. **Salazar Videaux, Leonel y Pérez-Alejo Neyra, Raúl.** *Módulo de Visualización de Gráficos Vectoriales para aplicaciones SCADA.* Habana : UCI, 2007.

3. **González Tamayo, Juan Carlos.** *DESARROLLO DE FUNCIONALIDADES AL MÓDULO DE PROCESAMIENTO DEL SCADA UX.* Habana : UCI, 2011.

4. **Peralta González, Yunior, y otros, y otros.** *MÓDULO DE ADQUISICIÓN PARA UN SISTEMA SCADA.* Habana : UCI.

5. **SCADA Systems.** SCADA Systems Concepts. *SCADA Systems Concepts.* [En línea] http://www.scadasystems.net/.

6. **Villalón Huerta, Antonio.** *SEGURIDAD EN UNIX Y REDES. Versión 2.1.* 2002.

7. **Pearson Education.** *Security in Computing, Fourth Edition.* s.l. : Prentice Hall, 2006. 0-13-239077-9.

8. **terra.** Diferentes tipos de intrusiones o ataques. *Diferentes tipos de intrusiones o ataques.* [En línea] 2012. http://www.terra.es/tecnologia/articulo/html/tec10590.htm.

9. **Ramio Aguirre, Jorge.** *LIBRO ELECTRÓNICO DE SEGURIDAD INFORMÁTICA Y CRIPTOGRAFÍA. Versión 4.1.* ESPAÑA : UNIVERSIDAD POLITÉCNICA DE MADRID, 2006.

10. **Segu-Info.** Amenazas Lógicas - Tipos de Ataques. *Amenazas Lógicas - Tipos de Ataques.* [En línea] 2009. http://www.segu-info.com.ar/ataques/ataques.htm.

11. **Info Spyware.** ¿Qué son los Rootkits? *¿Qué son los Rootkits?* [En línea] 2012. http://www.infospyware.com/articulos/que-son-los-rootkits/.

12. **Segu-Info.** Amenazas Lógicas - Tipos de Ataques - Ataques de Modificación (Daño). *Amenazas Lógicas - Tipos de Ataques - Ataques de Modificación (Daño).* [En línea] 2009. http://www.segu-info.com.ar/ataques/ataques_modificacion.htm.

13. **Scribd.** Software de Aplicación Administrativa. Tipos de Ataques Informáticos. *Software de Aplicación Administrativa. Tipos de Ataques Informáticos.* [En línea] 2009. http://www.scribd.com/doc/19397003/Tipos-de-Ataques-informaticos.

14. **Special Interest Group (SIG).** Sandboxing.org. *Sandboxing.org.* [En línea] http://sandboxing.org/?page_id=53.

15. —. Sandboxing.org. *Sandboxing.org.* [En línea] http://sandboxing.org/?page_id=77.

16. **Real Academia Española.** Buscador Drae. *Buscador Drae.* [En línea] 2012. http://buscon.rae.es/draeI/.

17. **Herrera Vázquez, Moisés, Hernández, Yaneisy y Fardales Pérez, Jaime.** *Conferencia 3 Base de Datos de Tiempo Real en el "GUARDIÁN DEL ALBA".* 2008.

18. **DPS Telecom.** SCADA Tutorial White Paper. *SCADA Tutorial White Paper.* [En línea] http://www.dpstele.com/white-papers/scada/offer.php?link=wp_pod.

19. **GALBA.** *Especificación de puntos calculados y scripts.* Venezuela : s.n.

20. **GE Multilin.** *DDS Sistema Integrado de Protección y Control Manual de Instrucciones.* 2005.

21. **Grupo Eros Serconi.** *SISTEMA DE SUPERVISIÓN Y CONTROL DE PROCESOS EROS - Manual del Usuario.*

22. **Boost.** Boost C++ libraries. *Boost C++ libraries.* [En línea] http://www.boost.org/community/sandbox.html.

23. **Special Interest Group (SIG).** Sandboxing.org. *Sandboxing.org.* [En línea] http://sandboxing.org/?page_id=5.

24. **Special Interest Gropu (SIG).** Sandboxing.org. *Sandboxing.org.* [En línea] http://sandboxing.org/?page_id=16.

25. **Special Interest Group (SIG).** Sandboxing.org. *Sandboxing.org.* [En línea] http://sandboxing.org/?page_id=19.

26. —. Sandboxing.org. *Sandboxing.org.* [En línea] http://sandboxing.org/?page_id=8.

27. —. Sandboxing.org. *Sandboxing.org.* [En línea] sandboxing.org/?page_id=11.

28. —. Sandboxing.org. *Sandboxing.org.* [En línea] http://sandboxing.org/?page_id=110.

29. **Corbet, Jonathan.** Seccomp and sandboxing. *Seccomp and sandboxing.* [En línea] 13 de Mayo de 2009. http://lwn.net/Articles/332974/.

30. **Seminara, Dario.** Shikashi - A flexible sandbox for ruby. *Shikashi - A flexible sandbox for ruby.* [En línea] 2011. http://rubydoc.info/gems/shikashi/frames.

31. **LIU Yu.** OpenJudge Alliance. *OpenJudge Alliance.* [En línea] 2012. http://openjudge.net/~liuyu/Project/LibSandbox.

32. **Vicente Vallejo, Javier.** Sandbox I. Sandboxie. Aislamiento de procesos mediante control de acceso a objetos en kernel. *Sandbox I. Sandboxie. Aislamiento de procesos mediante control de acceso a objetos en kernel.* [En línea] 23 de Mayo de 2011. http://blog.48bits.com/2011/05/23/sandbox-i-sandboxie-aislamiento-de-procesos-mediante-control-de-acceso-a-objetos-en-kernel/.

33. **Matsumoto, Yukihiro.** Ruby, A Programer's Best Friend. *Ruby, A Programer's Best Friend.* [En línea] http://www.ruby-lang.org/.

34. **S. Pressman, Roger.** *Ingenieria del Software: un enfoque práctico. V Edición.* s.l. : Mc Graw Hill, 2005.

35. **Universidad de Murcia.** *Metodología de desarrollo de software.* s.l. : Universidad de Murcia, 2011.

36. **Ambysoft.** The Agile Unified Process (AUP). *The Agile Unified Process (AUP).* [En línea] Scott W. Ambler, 2009. http://www.ambysoft.com/unifiedprocess/agileUP.html.

37. **Sommerville, Ian.** *Ingeniería del software. Séptima edición.* s.l. : Addison-Wesley, 2005.

38. **González Cornejo, J. E.** ¿Qué es UML? *¿Qué es UML?* [En línea] 16 de Octubre de 2009. http://www.docirs.cl/uml.htm.

39. **Mednieks, Zigurd, y otros, y otros.** *Programing Android.* s.l. : O'Reilly, 2011.

40. **Hechavarria R., Dayra I., Rodríguez Valdés, Orlando y Ruiz Rodríguez, Alian.** *Especificación de requisitos de software. Versión 1.3.* Habana : UCI, 2012.

41. **SPARX SYSTEMS.** Diagrama de Componentes UML 2. *Diagrama de Componentes UML 2.* [En línea] 2007. http://www.sparxsystems.com.ar/resources/tutorial/uml2_componentdiagram.html.

42. **Chávez Lorenzo, Ariel.** *Estándares de codificación para C++.*

43. —. *Estándares de codificación para C++. Versión 1.0.* 2010.

Bibliografía

1. **Salazar Videaux, Leonel y Pérez-Alejo Neyra, Raúl.** *Módulo de Visualización de Gráficos Vectoriales para aplicaciones SCADA.* Habana : UCI, 2007.

2. **González Tamayo, Juan Carlos.** *DESARROLLO DE FUNCIONALIDADES AL MÓDULO DE PROCESAMIENTO DEL SCADA UX.* Habana : UCI, 2011.

3. **Peralta González, Yunior, y otros, y otros.** *MÓDULO DE ADQUISICIÓN PARA UN SISTEMA SCADA.* Habana : UCI.

4. **Villalón Huerta, Antonio.** *SEGURIDAD EN UNIX Y REDES. Versión 2.1.* 2002.

5. **Pearson Education.** *Security in Computing, Fourth Edition.* s.l. : Prentice Hall, 2006. 0-13-239077-9.

6. **Ramio Aguirre, Jorge.** *LIBRO ELECTRÓNICO DE SEGURIDAD INFORMÁTICA Y CRIPTOGRAFÍA. Versión 4.1.* ESPAÑA : UNIVERSIDAD POLITÉCNICA DE MADRID, 2006.

7. **Herrera Vázquez, Moisés, Hernández, Yaneisy y Fardales Pérez, Jaime.** *Conferencia 3 Base de Datos de Tiempo Real en el "GUARDIÁN DEL ALBA".* 2008.

8. **GALBA.** *Especificación de puntos calculados y scripts.* Venezuela : s.n.

9. **GE Multilin.** *DDS Sistema Integrado de Protección y Control Manual de Instrucciones.* 2005.

10. **Grupo Eros Serconi.** *SISTEMA DE SUPERVISIÓN Y CONTROL DE PROCESOS EROS - Manual del Usuario.*

11. **S. Pressman, Roger.** *Ingenieria del Software: un enfoque práctico. V Edición.* s.l. : Mc Graw Hill, 2005.

12. **Universidad de Murcia.** *Metodología de desarrollo de software.* s.l. : Universidad de Murcia, 2011.

13. **Sommerville, Ian.** *Ingeniería del software. Séptima edición.* s.l. : Addison-Wesley, 2005.

14. **Mednieks, Zigurd, y otros, y otros.** *Programing Android.* s.l. : O'Reilly, 2011.

15. **Hechavarria R., Dayra I., Rodríguez Valdés, Orlando y Ruiz Rodríguez, Alian.** *Especificación de requisitos de software. Versión 1.3.* Habana : UCI, 2012.

16. **Chávez Lorenzo, Ariel.** *Estándares de codificación para C++.*

17. —. *Estándares de codificación para C++. Versión 1.0.* 2010.

18. **RedUSERS.** *El lenguaje Ruby.*

19. **García Hernández, Luis Enrique, y otros, y otros.** *DESARROLLO SISTEMA DE SUPERVISIÓN Y CONTROL DE PROCESOS SCADA-UX.* La Habana : s.n., 2011.

20. **Hernández León, Rolando Alfredo y Coello González, Sayda.** *EL PROCESO DE INVESTIGACIÓN CIENTÍFICA.* La Habana : Editorial Universitaria del Ministerio de Educación Superior, 2011.

21. **Kölling, Michael.** *The problem of teaching object-oriented programming.* Australia : School of Computer Science and Software Engineering. Monash University.

Anexos

```
int main(){
    VALUE result;

    int status;
    ruby_init();
    ruby_init_loadpath();
    rb_load_protect(rb_str_new2("fichero.rb"), 0, &status);
    rb_eval_string("$result = sum()");
    result = rb_gv_get("result");
    ruby_finalize();

    cout<<"El resultado de la suma es = "<< NUM2INT(result) << endl;

    return 0;
}
```

Figura 9. Ejemplo de embebido de Ruby en C/C++.

```
require "rubygems"
require "shikashi"
require "declarations"

include Math
include Shikashi

s = Sandbox.new
priv = Privileges.new

priv = Priv()

code = "
def sum()
    c = 16
    (c * sqrt(c))/2
end
"

s.run(code, priv, :no_base_namespace => true)
```

Figura 10. Ejemplo de fichero de Ruby con sandbox Shikashi.

```
=begin
Nota: Los operadores que no se incluyen es porque ya se encuentran incluidos...
Nota: Se pueden agregar operadores o sobrecargar funciones fuera del sandbox que agreguen otras operaciones...
=end

require "rubygems"
require "shikashi"

include Math
include Shikashi

def Priv()
priv = Privileges.new

#Including Method Print...
priv.allow_method :print

#Including Methods of Math...
priv.allow_method :sin
priv.allow_method :cos
priv.allow_method :sqrt
priv.allow_method :%

#Including Operators...
priv.allow_method :+
priv.allow_method :-
priv.allow_method :*
priv.allow_method :/

#Others and Others...
end
```

Figura 11. Ejemplo de fichero utilizado para configurar los permisos del sandbox Shikashi.

Glosario de términos

ABI: es una interfaz utilizada para describir la interfaz de bajo nivel entre una aplicación o cualquier tipo de programa y el sistema operativo u otra aplicación.

API: es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.

CPU: es el componente del computador y otros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.

Fiabilidad: probabilidad de que un sistema se comporte tal y como se espera de él.

Lenguaje interpretado: es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete. A ciertos lenguajes interpretados también se les conoce como lenguajes de script.

Lenguaje flexible: es el cual se adapta fácilmente a los cambios o las circunstancias.

Lenguaje multiparadigma: es el que puede soportar más de un paradigma de programación con el objetivo de que un programador utilice el más conveniente a la hora de resolver un problema.

Lenguaje script: lenguaje de programación diseñado para ser ejecutado por medio de un intérprete.

Log: es un registro oficial de eventos durante un rango de tiempo en particular. En seguridad informática es usado para registrar datos o información sobre quién, qué, cuándo, dónde y por qué un evento ocurre para un dispositivo en particular o aplicación. Es un registro de actividad de un sistema, que generalmente se guarda en un fichero de texto, al que se le van añadiendo líneas a medida que se realizan acciones sobre el sistema.

Portabilidad: propiedad de un programa o aplicación que le permite funcionar bajo diferentes sistemas.

Reflexivo: la reflexión es una actividad computacional que razona sobre su propia computación. Es la capacidad que tiene un programa para observar y opcionalmente modificar su estructura de alto nivel.

Software libre: software que respeta la libertad de los usuarios sobre su producto adquirido y, por tanto, una vez obtenido puede ser usado, copiado, estudiado, modificado y redistribuido libremente.

Tipado dinámico: se dice de un lenguaje de programación que usa un tipado dinámico cuando el chequeo de tipificación se realiza durante el tiempo de ejecución, opuesto al de compilación.

Tipificar: clasificar u organizar en tipos o clases una realidad o un conjunto de cosas.