Universidad de las Ciencias Informáticas

Facultad 5

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título: Infraestructura para la configuración de trazas y excepciones en el SISCP- GALBA.

Autores: Carlos Michel Montano Cabrera

Alejandro Navarro Fernández

Tutores: Ing. José Antonio Aragón Cáceres

Ing. Amado Espinosa Hidalgo

La Habana, junio de 2014

"Año 56 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaramos ser autores del presente trabajo de diploma y reconocemos a la Universidad de las Ciencias Informáticas los derechos del mismo, con carácter exclusivo.

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

Carlos Michel Montano Cabrera

_____

Firma del Autor

Alejandro Navarro Fernández

_____

Firma del Autor

Ing. José Antonio Aragón Cáceres

_____

Firma del Tutor

Ing. Amado Espinosa Hidalgo

_____

Firma del Tutor

Datos de contacto

Nombre y apellidos del tutor: José Antonio Aragón Cáceres.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

Correo electrónico: jaaragon@uci.cu

Especialista graduado en la Universidad de las Ciencias Informáticas (UCI), profesor instructor con 5 años de experiencia docente y 7 años de experiencia en la producción de software, específicamente en el desarrollo de sistemas SCADA. Arquitecto de software del SISCP- GALBA.

Nombre y apellidos del tutor: Amado Espinosa Hidalgo

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

Correo electrónico: aespinosa@uci.cu

Ingeniero Informático y profesor asistente del Departamento de Ingeniería y Gestión de Software de la Facultad 5. Posee 11 años de experiencia en la actividad docente y productiva.


La inteligencia no es la facultad de imponerse;
es el deber de ser útil a los demás.
José Martí

Agradecimientos

Es realmente difícil organizar pocas palabras, para dejar plasmado el agradecimiento más completo y sincero. Qué puedo decir. A mis padres, les debo la vida, en máxima expresión de la palabra. Solo puedo asegurar que, dedicaré cada minuto de mi existencia, a intentar encontrar en cada acción, cada pensamiento; la sensación de haberles agradecido enteramente por lo que he sido, soy, y estoy seguro que con su apoyo, para bien seré. Quiero dar gracias a mi hermana, por su apoyo y consejos, desde su experiencia; y al resto de mi familia, mis tíos y primos (Richard); por su confianza.

A Nayrobis, por no abandonar la lucha y permanecer a mi lado. Gracias por la paciencia. Por servirme de refugio, en momentos difíciles por los que se pasa. A veces por necesidad, y otras, por caprichos del destino. Gracias. Y gracias también a tu familia; a tu mamá, siempre al tanto de todo.

A mi compañero de equipo, Alejandro Navarro Fernández, por confiar en mí para zarpar junto al él, en esta difícil travesía. Gracias también, por ser excelente compañero y amigo.

A nuestros tutores José Tony y Amado, por guiarnos siempre por el camino correcto.

Al profesor Olaf S., por sus rápidas y certeras respuestas ante los problemas que se presentaron durante esta batalla. Y a los profes Osmany Jorge y Omar D, por su contribución certera con la realización de este trabajo.

De forma general, a todos los que contribuyeron, de una forma u otra. A Henry Marcelo, Luis A Valido y al profe Pimienta; por dedicar, de buena voluntad, más de un minuto de su tiempo.

Agradecimientos

A mis padres, por sus eternas enseñanzas, guía, ejemplo y sacrificio durante todos estos años.

A mi abuela, aunque hoy su enfermedad no la deje disfrutar de este título, se sentiría muy orgullosa de su nieto.

A mi familia en general, a Marielena, Barbarita, mis primos Camilo y Mayito, en fin a toda la familia del Canal.

A mi hermano mayor David Lago, su apoyo siempre ha estado presente para que siguiera adelante, y a Javier Domínguez por sus consejos, enseñanzas y su ejemplo, recuerda flaco, las personas inteligentes y buenas de corazón, siempre tendrán un lugar privilegiado en esta vida.

A mi compañero de tesis y amigo, Carlos Michel Montano Cabrera, y a su novia Nayrobis, gracias por aguantarme durante estos 5 años, por brindarme tu ayuda cuando la necesite, te digo lo mismo que al flaco.

A mi niña linda, Blanca, mi novia, amiga, consejera, gracias de todo corazón por aguantarme durante más de un año, aunque no lo creas has sido una pieza vital en este fin de carrera.

A nuestros tutores por su guía indispensable, al profe Tony, Amado, Osmany Jorge y Omar D.

Al Dr. Olaf; sin su ayuda este sueño no hubiese sido posible.

A los muchachos del laboratorio 23 Henry, Yosvany, Valido y al profe Pimienta.

A mis amigos Ernesto y Luis Manuel my president, por estar siempre dispuestos a escucharme y ayudarme.

A las personas que me han acompañado estos 5 años como grupo, a los que hoy también culminan y a los que por una causa u otra no lo lograron.

Dedicatoria

Los resultados obtenidos hasta hoy, ya sea con este trabajo, o en otro contexto; y los que pudiera merecer de hoy en lo adelante, estarán siempre dedicados, con el más sincero agradecimiento, a los que han sido capaz, con sacrificio y amor, de enseñarme lo que es luchar por lo que verdaderamente se quiere. Mis padres.

Importante, no debo dejar de mencionar, a aquellas personas que también se alegran de verdad por tus logros, y sienten contigo cada fracaso.

Carlos.

En breves palabras, a las personas que lo darían todo porque yo pudiese alcanzar mis sueños, mis padres. A mi abuelo Ricardo y tía Isabelita que, donde quiera que estén, siempre aportaron para poder llegar hasta aquí.

A todas las personas que de buena fe, se regocijan junto a mí de este éxito.

Alejandro.

RESUMEN

El Sistema Integral de Supervisión y Control de Procesos Guardián del ALBA (SISCP-GALBA), necesita de nuevos conceptos que permitan resolver problemas reflejados a través de la dispersión del código. Resultado de una incorrecta asignación de responsabilidades a los distintos elementos que integran el sistema, término conocido como incumbencia transversal. Las trazas, seguridad, manejo de excepciones, conexiones a base de datos, son conceptos que se reflejan de forma transversal en una aplicación. La Programación Orientada a Aspectos (POA), es un paradigma que promete la separación de requisitos secundarios, encapsulándolos fuera de la implementación básica en un nuevo concepto denominado aspecto. Al mismo tiempo se imponen nuevos criterios que flexibilicen la gestión de las excepciones, permitiendo un manejo declarativo de las mismas. La infraestructura que se propone permitirá una generación de trazas y excepciones teniendo en consideración todos los elementos antes planteados. El módulo HMI (Interfaz Hombre Máquina) del SISCP-GALBA es la plataforma de validación, al valorarse como crítico en la gestión de trazas y excepciones. Alcanzar este propósito posibilita un mayor entendimiento del código, centrando al desarrollador en la funcionalidad básica de la aplicación y, entre otros aspectos, disminuirá la dispersión del código provocada por la utilización de diferentes llamadas a bibliotecas. Se pretende además, de dotar al módulo HMI de un mecanismo para la generación de trazas, de forma tal, que posibilite la realización de auditorías al sistema.

Palabras claves*:*

Incumbencias transversales, manejo declarativo, Programación Orientada a Aspectos (POA), SCADA.

ÍNDICE DE CONTENIDO

INTRODUCCIÓN

En la actualidad se imponen nuevos criterios que requieren del aumento de los niveles de eficiencia, disminución de costo y optimización de los procesos industriales. Por esta misma línea, surgen nuevos *softwares* que permiten llevar un control constante y automático de los eventos que ocurren en un entorno industrial. Los Sistemas de Supervisión, Control y Adquisición de Datos (SCADA), fueron diseñados para funcionar sobre ordenadores en el control de procesos de producción. Estos permiten la obtención de información a través de la comunicación con sus dispositivos de campo, la cual es utilizada para un análisis sobre el comportamiento de los procesos que ocurren en una planta industrial. (1)

Existe un gran número de estos sistemas automatizados, entre los que se encuentra, el Sistema Integral de Supervisión y Control de Procesos Guardián del ALBA (SISCP-GALBA), desarrollado específicamente para la industria petrolera de Venezuela. Como todos los sistemas de este tipo, precisa de una correcta modularidad de sus componentes, donde se garantice una adecuada independencia entre cada una de las entidades. Actualmente, en el sistema se pasan por alto ciertas cuestiones en lo que a diseño respecta. Por ejemplo, al introducir en la funcionalidad principal, responsabilidades que no le corresponden; entiéndase, el registro de trazas y manejo de excepciones, que si bien son de vital importancia para el control de los procesos del sistema, su uso indiscriminado conlleva a una dispersión de este código por toda la aplicación (ver Figura 1).

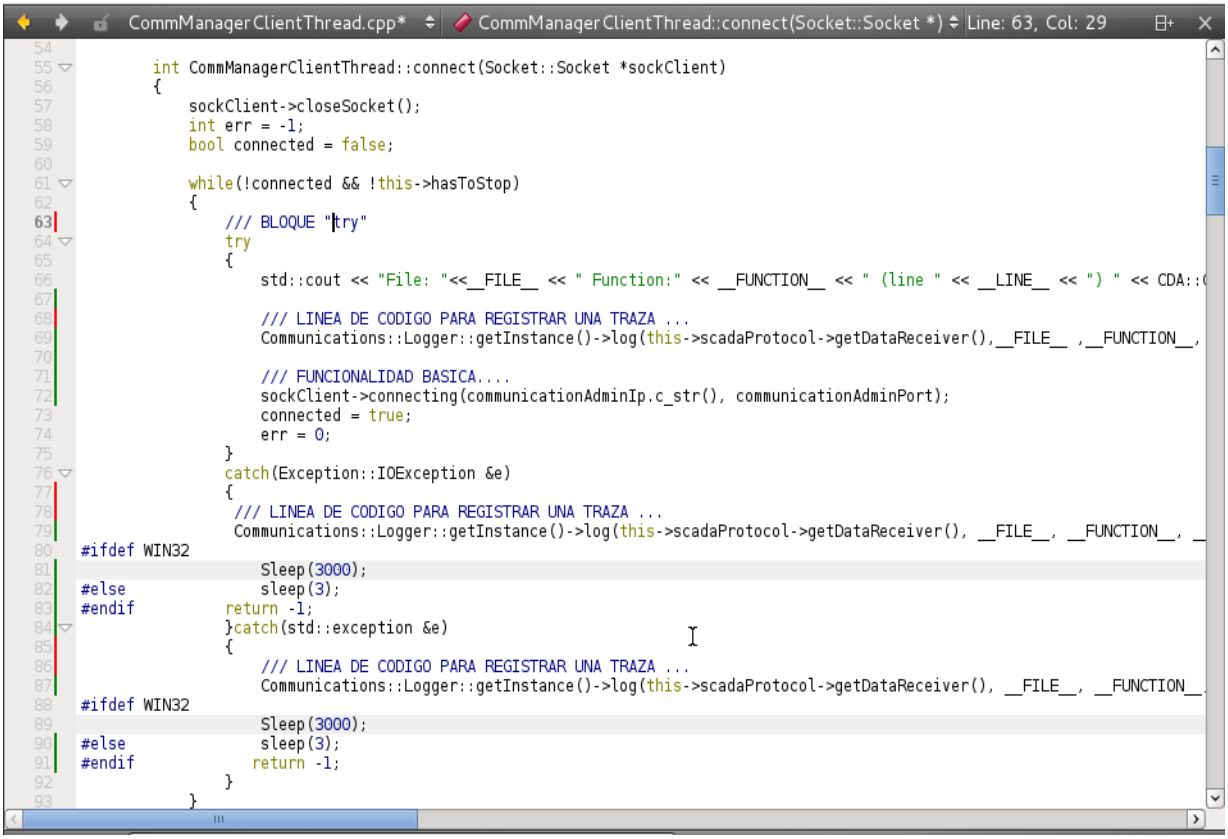


Figura 1. Ejemplo de código disperso (generación de trazas y captura de excepciones) en la clase CommManagerClientThread módulo Comunicación en el SISCP- GALBA.

Estos problemas que afectan a varios sectores de forma transversal, provocan, que al querer reutilizar sus componentes, exista pérdida de tiempo, debido a que la actividad se hace más compleja. En la mayoría de las ocasiones, es necesario comprender dichos componentes para futuras modificaciones. Una alternativa válida podría ser, optar por un código menos enredado (ver Figura 2).

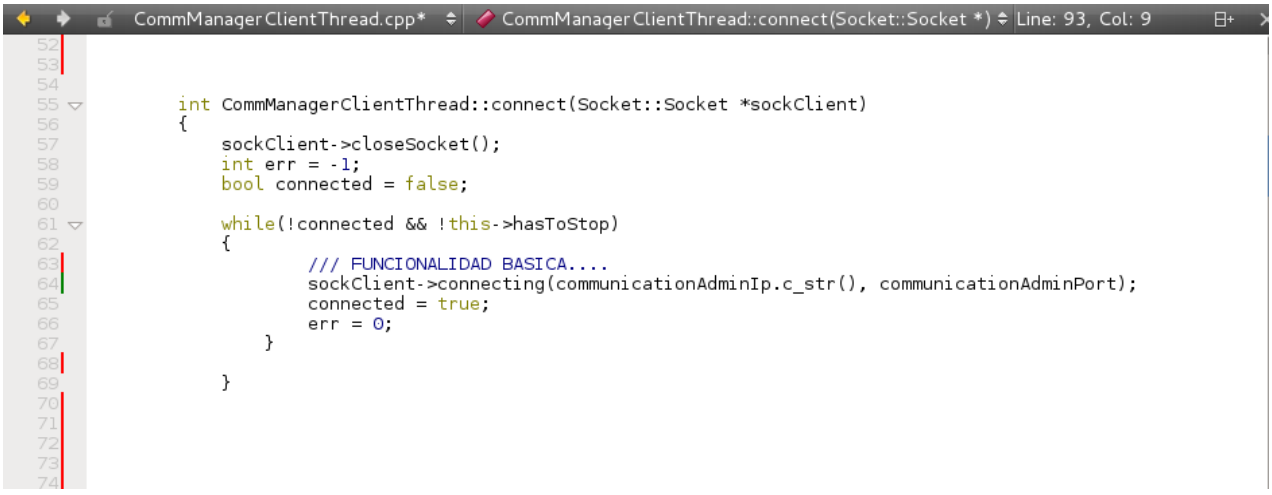


Figura 2. Ejemplo de un código limpio, eliminando el código disperso (generación de trazas y captura de excepciones) en la clase CommManagerClientThread del módulo Comunicación en el SISCP- GALBA.

No solo es importante el hecho de separar dichas funcionalidades de la lógica principal del sistema. Se requiere además, que estas se lleven a cabo satisfactoriamente; situación aún por resolver en el SISCP-GALBA. Se evidencia que de forma general, las trazas y la gestión de excepciones, no son definidas de manera homogénea en sus diferentes módulos y, en casos como el HMI (Interfaz Hombre-Máquina), no se efectúa. Estos son puntos claves en los procesos de registro y auditoría de todas las acciones ejecutadas por los usuarios sobre los recursos del sistema. El hecho de no realizar debidamente dicho proceso de auditoría, introduce el riesgo de no identificar violaciones, debilidades y amenazas para la aplicación.

Para el caso de las excepciones, es la biblioteca *IOException* la encargada de proveer mecanismos que permitan llevar a cabo la gestión antes mencionada. Para este punto, podría pensarse en nuevos mecanismos, dirigidos a mejorar el manejo de estos errores que puedan surgir durante la ejecución de la aplicación. Lo anterior, hace referencia a la idea de introducir nuevos criterios que posibiliten mayor flexibilización de este proceso; dígase a la hora de definir, lanzar y capturar las excepciones.

Atendiendo a la problemática planteada anteriormente, se define como **problema investigativo** a resolver**:** ¿Cómo estandarizar las trazas y excepciones en el SISCP-GALBA?

A partir del problema investigativo planteado se especifica como **objeto de estudio:** configuración de trazas y excepciones en sistemas distribuidos; teniendo como **campo de acción:** configuración de trazas y excepciones en el SISCP-GALBA.

Se define como **objetivo general** de la investigación: desarrollar una infraestructura para la configuración de trazas y excepciones en el SISCP- GALBA.

Para lograr el cumplimiento del objetivo general de esta investigación es necesario realizar las siguientes **tareas investigativas:**

- Elaboración del marco teórico-conceptual de los temas relativos a los sistemas SCADA y los mecanismos existentes para la generación de trazas y excepciones mediante un estudio del estado del arte de estas materias.

- Evaluación del estado actual de las bibliotecas que permitan gestionar la configuración de las trazas y excepciones en el desarrollo de *software* para el SISCP- GALBA.
- Definición de los requisitos para las funcionalidades implicadas en el desarrollo de la infraestructura para la configuración de trazas y excepciones.
- Definición e implementación de la infraestructura para la configuración de trazas y excepciones definida para el SISCP- GALBA.
- Validación del mecanismo desarrollado a partir de las pruebas de liberación establecidas en el módulo HMI del SISCP-GALBA.

Para el desarrollo de las tareas de investigación se combinan diferentes métodos en la búsqueda y procesamiento de la información, como son:

Métodos Teóricos:

Análisis histórico-lógico: Permitió conocer con mayor profundidad los antecedentes y tendencias actuales para la generación de trazas y un manejo de las excepciones. Así como los principales elementos que tratan estos conceptos que se reflejan transversalmente en la aplicación.

Analítico-Sintético: Utilizado para realizar un análisis de la documentación consultada que posibilite la extracción de la información requerida para una mejor generación de trazas y excepciones en el SISCP-GALBA.

Modelación: Para definir y representar gráficamente las funcionalidades y elementos representativos de la propuesta realizada, haciendo uso del Lenguaje Unificado de Modelado (UML).

Método Empírico:

Experimentos: Empleado en la elaboración de diferentes prototipos funcionales, con el propósito de comprobar la efectividad de la implementación realizada y un mayor entendimiento de la misma.

El presente trabajo de diploma, consta de tres capítulos estructurados de la siguiente forma:

Capítulo 1- Fundamentación teórica.

Se realiza un estudio del estado del arte, sobre las características principales que debe contener un SCADA, indagándose en mecanismos establecidos para la configuración de trazas y excepciones en el SISCP-GALBA. Se analizan otros conceptos que permitan una mayor abstracción de estos aspectos fuera de la funcionalidad básica de la aplicación. Conjuntamente se detallarán las herramientas, lenguajes y metodología seleccionada para la conformación del objetivo planteado.

Capítulo 2- Planificación y diseño.

Se realiza la propuesta inicial de la infraestructura para la configuración de trazas y excepciones. Igualmente se especifican los requisitos que debe cumplir, a través de las diferentes historias de usuarios. Se define la relación entre cada uno de los elementos de la aplicación, y su futura integración con el SISCP-GALBA. Además, se plasma la descripción de cada una de las principales clases empleadas en la creación de dicha infraestructura.

Capítulo 3- Codificación y prueba.

En la fase de codificación, se brinda una descripción de la propuesta desarrollada, especificando el estándar utilizado para patentizar las buenas prácticas de programación. Se realizan las pruebas de validación correspondientes, según propone la metodología de desarrollo seleccionada.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Introducción

La existencia de elementos que afectan transversalmente a la aplicación, pone en riesgo la correcta reusabilidad de la misma o de componentes que la integran. Los diferentes paradigmas de programación establecen mecanismos para tratar de eliminar esta situación. Aunque no ofrecen buenos resultados, al no poder encapsular estos intereses en una única entidad, debido a que atraviesan diferentes partes de la aplicación. En el presente capítulo, se indagará en el surgimiento y principales características de un paradigma de programación, que enfatiza su existencia para la eliminación de este tipo de problemas. Igualmente, se describirán los principales lenguajes que soportan en la actualidad estos conceptos y, se realizará la selección de la metodología para el desarrollo del *software*, principales *frameworks* y el entorno integrado de desarrollo.

1.1 Sistemas SCADA

Los sistemas SCADA utilizan las computadoras y tecnologías de comunicación para automatizar el monitoreo y control de procesos industriales. Estos sistemas son partes integrales de la mayoría de los ambientes industriales complejos o geográficamente dispersos, pues recopilan toda la información de una variada cantidad de fuentes, y la presentan a un operador en una forma amigable. La información obtenida de dichos procesos es utilizada para realizar una serie de análisis o estudios con los que se pueden obtener valiosos indicadores. Esto provee una retroalimentación sobre el trabajo de los usuarios del sistema o sobre el propio proceso, para poder tomar decisiones operacionales apropiadas. (1)

Este *software* debe ser capaz de complementar entre sus funcionalidades básicas la de adquisición de datos, para posibilitar la recolección, procesamiento y almacenamiento de la información. Lo anterior permite la realización de futuros análisis vitales para determinar el comportamiento de los procesos que se desarrollan en la planta industrial. La supervisión es otro de los elementos que debe estar establecido en estos sistemas automáticos. Dicha supervisión, permite observar desde un monitor, la evolución de las variables del proceso, lo que ayuda a la toma de decisiones ante una determinada fluctuación de los índices productivos. El último elemento es el control, muy significativo, pues permitirá modificar el desarrollo del proceso, actuando sobre los diferentes componentes básicos como alarmas y dispositivos. (1)

Deben ser sistemas de arquitectura abierta, capaces de crecer o adaptarse según las necesidades cambiantes de la empresa. También debe existir una fácil y transparente comunicación entre el usuario con el equipo de planta y el resto de la empresa.

En la mayoría de los sistemas, existen determinados aspectos que son de vital interés para la correcta ejecución de la aplicación. Las trazas y excepciones son un ejemplo de ello, ya que pueden definir la fiabilidad y robustez con que debe contar el producto a desarrollar.

1.2 Trazas y excepciones

En gran parte de las aplicaciones, existen aspectos claves a la hora de definir criterios de confiabilidad de la misma, permitiendo que el sistema ofrezca sus funcionalidades dentro de un entorno de calidad.

Excepciones

Con la evolución del *software*, los desarrolladores se han encontrado con algunos errores que ocurren mientras se están ejecutando los programas, como: dispositivos que fallan, datos que se introducen incorrectamente y valores fuera del rango esperado. Incorporar todas las posibilidades de fallo y casos extraños en la lógica de un algoritmo, puede entorpecer su legibilidad y eficiencia; no tenerlas en cuenta produce que los programas aborten de forma incontrolada y puedan generarse daños en la información manejada. Los mecanismos de control de excepciones permiten contemplar este tipo de problemas separándolos de lo que es el funcionamiento normal del algoritmo. (2)

Diseñar una estrategia efectiva de gestión de excepciones, es fundamental para la estabilidad e incluso la seguridad de la aplicación. Si no se realiza una gestión de excepciones correcta, la aplicación puede ser vulnerable ante ataques o revelar información confidencial. (3)

El originar excepciones de negocio y la propia gestión de excepciones, son operaciones con un coste de proceso relativamente caro en el tiempo, por lo que es importante que el diseño tenga en cuenta el impacto en el rendimiento. (3)

Trazas

En los últimos años, la auditoría de información ha aumentado en importancia como resultado de su impacto en la prevención o detección de violaciones que afecten la confidencialidad, integridad, disponibilidad y trazabilidad de los recursos de una organización. La auditoría de información es un componente importante de la auditoría

informática y depende en gran medida de la expresividad de las trazas para garantizar la calidad de los resultados. (4)

Una traza es un registro de actividad de un sistema, que generalmente se almacena en un fichero de texto, al que se le van añadiendo líneas a medida que se realizan acciones sobre el sistema. Son un registro oficial de eventos durante un período de tiempo en particular. Para los profesionales en seguridad informática, una traza es usada para registrar datos o información sobre quién, qué, cuándo, dónde y por qué (*who, what, when, where y why*) un evento ocurre para un dispositivo o aplicación. (5)

Si no se diseña e implementa correctamente, la aplicación puede ser vulnerable a acciones de repudio cuando ciertos usuarios nieguen sus acciones. Los ficheros de trazas pueden ser requeridos para probar acciones incorrectas en procedimientos legales.

La mayoría de las aplicaciones contienen estas funcionalidades en común. Normalmente se le denomina **aspectos transversales**, porque afectan a la aplicación entera, y deben estar por lo tanto centralizados en una localización específica si es posible, favoreciendo la reutilización de componentes. (3)

En la actualidad existen varias propuestas para modularizar estas incumbencias, los criterios más robustos van dirigidos hacia *Java* y *.NET*. Para el caso específico de C++, los intentos de encapsular estos conceptos se reflejan principalmente a través del uso de bibliotecas.

1.3 Intentos de encapsular las incumbencias transversales

Para mejor entendimiento, en el presente documento, el término ¨Incumbencias transversales¨ se aborda como: aquellas funcionalidades que resultan comunes para las distintas partes que integran un sistema. Ejemplo de estas son: el registro de trazas y manejo de excepciones. Dichas funcionalidades resultan difíciles de encapsular, y más, si se utilizan los métodos tradicionales; entiéndase, hasta el enfoque Orientado a Objeto.

1.3.1 Bibliotecas

Existen diferentes mecanismos que permiten alcanzar una mayor modularidad en las aplicaciones. El uso de bibliotecas ofrece la posibilidad de estructurar y simplemente invocar la funcionalidad brindada por la misma cuando sea requerida.

Log4j

La generación de trazas se ha realizado generalmente por medio de bibliotecas, o unidades dedicadas a ese efecto, tal como *Log4j*. En ocasiones, estas unidades deben

combinarse con el código del programa o aplicación a monitorizar y, en otras puede hacerse por medio de una conexión al canal de comunicación o al sistema hardware. Tanto de una forma como de otra, la traza deberá mantener la correspondiente coherencia y homogeneidad en su creación. (6)

La biblioteca *Log4j* fue desarrollada en Java por la *Apache Software Foundation.* Es de código abierto y posibilita a los desarrolladores de *software* elegir la salida y el nivel de granularidad de los mensajes o *logs* (*logging*) en tiempo de ejecución, y no en tiempo de compilación como es comúnmente realizado. Contiene tres componentes principales: ***loggers, appenders and layouts***. La configuración de salida y granularidad de los mensajes, es realizada mediante un archivo de configuración externo. Además de *Java, Log4j* ha sido implementado en otros lenguajes: *Log4cxx para C++ y Lognet* para *.Net*. (7)

Loggers

La principal ventaja del *Log4j* comparado con el tradicional *System.out.println(),* es su capacidad para habilitar o deshabilitar un registro de mensajes sin afectar a los demás. Son entidades autónomas con configuración independiente, de tal manera que uno tenga cierta funcionalidad que otro no. (7)

Appenders

Log4j permite registrar los eventos en varios destinos, donde una salida (pantalla, archivo) es un *appender*. Actualmente existen para consola, archivos, componentes visuales, servidores de *sockets* y demonios de registro *Unix*. (7)

Layouts

El *layout* es el responsable de formatear los mensajes de acuerdo a criterio del programador. Existen dos tipos de *Layout*: *SimpleLayout*, que básicamente muestra el nivel del mensaje y el mensaje en sí, y el *PatternLayout* que consiste en formatear la salida. (7)

1.3.2 Servidores de aplicaciones y componentes

Tanto *Java* como *.NET,* ofrecen como parte de sus arquitecturas de referencia, componentes denominados servidores de aplicaciones. Estos brindan un entorno de ejecución y una serie de servicios para las aplicaciones que son desplegadas en ellos. Al mismo tiempo permiten que los componentes desplegados hagan uso de los

mencionados servicios en forma declarativa, disminuyendo así el esfuerzo de desarrollo y la cantidad de código relacionado a la infraestructura. (8)

Si bien con el uso de servidores de componentes se facilita la implementación de algunas incumbencias transversales, existen algunas limitaciones. Una de las desventajas que tiene el uso de los mismos es que suelen imponer un modelo de programación bastante intrusivo. Entorpeciendo la prueba de los componentes y aumenta de manera considerable el esfuerzo de despliegue de la aplicación. (8)

1.4 Bibliotecas para la generación de trazas y excepciones en el SISCP-GALBA

La *Boost* constituye un conjunto de bibliotecas de alta eficacia para los desarrolladores que utilizan el lenguaje C++. Son gratuitas, de código fuente abierto, proporcionando a los programadores bibliotecas útiles y bien diseñadas, que funcionan con los estándares de C++ existentes. Se caracteriza por su gran portabilidad y calidad, aglutinando en cada una, las diferentes funcionalidades que pueden ofrecer. Su uso va desde almacenar punteros a funciones, hasta la generación de trazas y excepciones, tal es el caso de *Boost Log 2.0 y Boost Exception.* De igual forma la biblioteca estándar de C++ incluye una jerarquía de clases que posibilitan su uso para el manejo de diferentes situaciones de error. Aunque se generalizan otras que permitan una mayor portabilidad, al modularizar sus funcionalidades.

En el caso específico del SISCP-GALBA se hace uso de la biblioteca *Log4cxx* para la persistencia de las trazas .Está presentado bajo la licencia *Apache*, la cual es de código abierto certificada por la *Open Source Initiative.*

Para las excepciones se hace uso de la biblioteca *IOException*, la cual se centra en crear mensajes de error personalizados. Si bien las funcionalidades de las bibliotecas se encuentran modularizadas, las llamadas a las mismas pueden dispersar su código por toda la aplicación. Tratando de solventar esta situación, en donde el código se ve esparcido por la funcionalidad básica, surge el paradigma de Programación Orientada a Aspectos (POA).

1.5 Programación Orientada a Aspectos

La historia del *software* ha ido evolucionando a través de sus diferentes paradigmas de programación. Constantemente se buscan nuevas metodologías y tecnologías que reduzcan la complejidad del *software*, promuevan su reusabilidad y evolución. La Programación Orientada a Objetos (POO) busca alcanzar un *software* de calidad a través

de la descomposición, la abstracción y encapsulación de la información. A pesar de esto, siguen existiendo ciertas incumbencias que ni la POO, ni las técnicas precedentes de programación, han logrado modularizar de manera efectiva. La generación de registros de auditoría, accesos a bases de datos, seguridad y la concurrencia del acceso a cierta información, se ejemplos de dichas incumbencias, conocidas también como servicios comunes. Los mismos, esparcen su código por toda la aplicación, afectando a la misma de forma transversal.

El concepto de POA fue introducido por *Gregor Kiczales a* comienzos de la década del 90 con un propósito principal, separar conceptos y minimizar las dependencias entre estos. Con el primer objetivo se consigue que cada cosa esté en su sitio, es decir, que cada decisión se tome en un lugar concreto; con el segundo se tiene una pérdida del acoplamiento entre los distintos elementos. (9)

Aspecto, es una unidad que se define en términos de información parcial de otras unidades. La definición de aspecto ha evolucionado a lo largo del tiempo, pero la más difundida es la siguiente:

“Un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa o de código, es una unidad modular del programa que aparece en otras unidades modulares del programa.” (9)

De manera más informal, los aspectos son la unidad básica de la POA y pueden definirse como las partes de una aplicación que describen las cuestiones claves relacionadas con la semántica esencial o el rendimiento. También pueden verse como los elementos que se diseminan por todo el código y que son difíciles de describir localmente con respecto a otros componentes. (10)

La POA permite capturar las competencias que atraviesan el sistema (competencias transversales) en entidades bien definidas llamadas aspectos, consiguiendo una clara separación de intereses, eliminando el código disperso y enredado**,** además de los efectos negativos que este supone.(11)

- **código disperso** (*scatteredcode*): el código que satisface una competencia transversal, está esparcido por distintas partes del sistema. Se pueden distinguir dos tipos de código esparcido:
    1. bloques de código duplicados: cuando los mismos bloques de código aparecen en distintas partes del sistema.
    2. bloques de código complementarios: cuando las distintas partes de una incumbencia son implementadas por módulos diferentes.
- **código enredado** (*tangledcode*): una clase o módulo, además de implementar su funcionalidad principal, debe ocuparse de otras competencias. (11)

El código disperso y enredado, es un código difícil de reutilizar, mantener, evolucionar y de pobre trazabilidad. Estos efectos hacen que disminuya la calidad del *software* diseñado y se reduzca la productividad. De ahí la necesidad de nuevas técnicas que ayuden a conseguir una separación de intereses clara. Para así reducir la complejidad del sistema a implementar y mejorar aspectos de calidad como la adaptabilidad, extensibilidad, mantenibilidad y reutilización. (11)

Los aspectos no suelen ser unidades de descomposición funcional del sistema, sino, propiedades que afectan al rendimiento o la semántica de los componentes. Entre los principales beneficios que brindan se pueden mencionar:

- Un código menos enredado, más natural y más reducido.
- Una mayor facilidad para razonar sobre las materias, ya que están separadas y tienen una dependencia mínima.
- Más facilidad para depurar y hacer modificaciones en el código.
- Se consigue que un conjunto grande de modificaciones en la definición de una materia tenga un impacto mínimo en las otras.
- Se tiene un código más reusable, acoplándose y desacoplándose cuando sea necesario. (11)

1.5.1 Fundamentos de la Programación Orientada a Aspectos

Los lenguajes orientados a aspectos (LOA) definen una nueva unidad de programación de *software* para encapsular las funcionalidades que cruzan todo el código (**competencias transversales**). Estos deben soportar la separación de aspectos como la sincronización, la distribución, el manejo de errores, la optimización de memoria, la gestión de seguridad y la persistencia. (12)

- **competencias transversales** (*crosscutting-concern*): las que afectan a varias partes del sistema, relacionadas con requisitos no funcionales del sistema.

En la figura 3 se observa la estructura de una aplicación implementada mediante un lenguaje de procedimiento generalizado (LPG), y la estructura de la misma aplicación implementada mediante POA. Para la construcción de dicha aplicación se necesitan tener en cuenta cuestiones de sincronización y gestión de memoria. En esta figura se distinguen mediante trazas de distintas formas y color, el código relativo a la funcionalidad básica de la aplicación, el código de sincronización y el código para gestionar la memoria.

Figura 3. Lenguaje de procedimiento generalizado y POA (13)

Los lenguajes orientados a aspectos definen una nueva unidad de programación de *software* para encapsular las funcionalidades diseminadas por todo el código(los aspectos). Sin embargo, **componentes** y **aspectos** deben interactuar. En la ejecución del programa, finalmente, los aspectos deben estar insertados dentro de los componentes. Para ello es necesario definir claramente cómo será ésta estrategia de inserción. (13)

- **un componente:** puede encapsularse claramente dentro de un procedimiento generalizado. Los componentes son unidades de descomposición funcional del sistema.
- **un aspecto:** no puede encapsularse claramente en un procedimiento generalizado. Suelen ser propiedades que afectan al rendimiento o a la semántica de los componentes.

En la terminología de POA, este proceso se denomina entretejido, ya que puede pensarse que aspectos y componentes deben entretejerse, para formar finalmente un código ejecutable. (13)

Para poder realizar este entretejido entre aspectos y componentes, es necesario definir o declarar ciertos puntos de enlace (*joinpoint*). Estos son una clase especial de interfaz entre los aspectos y los módulos del lenguaje de componentes. Son los lugares del código en los que éste se debe modificar, incorporando los comportamientos adicionales especificados en los aspectos. (13)

El encargado de realizar la inserción de los aspectos en los puntos de enlace, es conocido como tejedor (*weaver*). El tejedor se encarga de entremezclar los diferentes mecanismos de abstracción y composición que aparecen en los lenguajes de aspectos y componentes, en los puntos de enlace (ver Figura 4). (13)

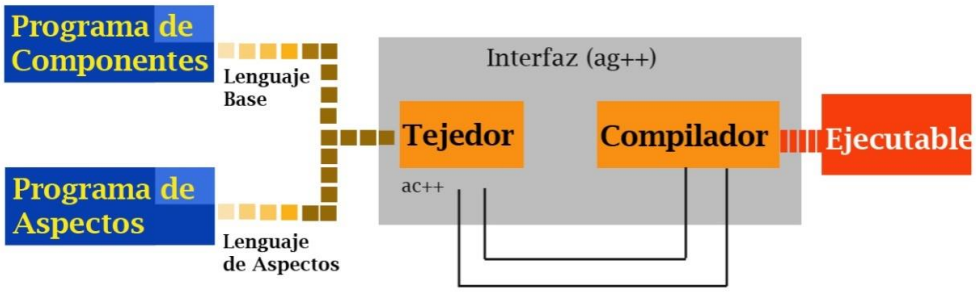


Figura 4. Ejemplo para la construcción de un proyecto Qt, con AspectC++.

Para tener un programa orientado a aspectos es necesario definir los siguientes elementos:

- Un lenguaje para definir la funcionalidad básica. se conoce como lenguaje base. Suele ser un lenguaje de propósito general, tal como C++, *Java*.
- Uno o varios lenguajes de aspectos. define la forma y comportamiento de los mismos (*AspectJ, AspectC++*).
- Un tejedor de aspectos. se encargará de combinar ambos lenguajes. El proceso de mezcla se puede retrasar para hacerse en tiempo de compilación, o bien en tiempo de ejecución. (12)

Los componentes y los aspectos se pueden tejer en forma estática (en tiempo de compilación), o en forma dinámica (en tiempo de ejecución).

El entretejido estático consiste en modificar el código fuente de los componentes, en los puntos de enlace, insertando el código definido en los aspectos. La principal ventaja de este tipo de entretejido, es evitar que el nivel de abstracción que se introduce con la POA,

derive en una degradación del rendimiento de la aplicación. Otro criterio de peso, es que varios controles de errores, pueden ser realizados al compilar. (13)

El entretejido dinámico requiere que los aspectos existan, y estén presentes tanto en tiempo de compilación como en tiempo de ejecución. Una manera de conseguir esto, es lograr que tanto los aspectos como los componentes se modelen como objetos y se mantengan en el ejecutable. Un tejedor dinámico será capaz de añadir, adaptar y remover aspectos de forma dinámica durante la ejecución. Pueden utilizarse mecanismos de herencia en forma dinámica para agregar el código de los aspectos a los componentes (o clases). La ventaja del entretejido dinámico radica en su potencialidad, ya que podrían tomarse decisiones en base a criterios dinámicos en la configuración de los aspectos. Las desventajas radican en afectar directamente el rendimiento de la aplicación y en la complejidad de la depuración de errores, ya que varios de éstos podrían ser detectados únicamente en tiempo de ejecución. (13)

1.5.2 Principales elementos de los lenguajes orientados a aspectos

Existen ciertos elementos fundamentales a la hora de hacer uso de un LOA:

Puntos de enlace (*joinpoint*): son puntos bien definidos en la ejecución de un programa. Ejemplo de ello, la llamada a un método, la lectura de un atributo. Representan los lugares donde los aspectos añaden su comportamiento. Mientras más tipos de *joinpoints* se puedan especificar con una herramienta POA, más casos se podrán representar en un aspecto y por ende más potente será dicha herramienta. (13)

Puntos de corte (*pointcut*): agrupan puntos de enlace. Mediante un punto de corte podemos agrupar las llamadas a todos los métodos de cierta clase y exponer su contexto (argumentos, objeto invocador, objeto receptor). (13)

Advice: es un fragmento de código equivalente al contenido de un método, en donde se especifica el comportamiento transversal que contendrá el aspecto. Este comportamiento va acompañado de la indicación del *pointcut* donde debe ser insertado, y la forma de hacerlo (antes, después o en reemplazo del *pointcut*). (13)

Matemáticamente, los aspectos constituyen una extensión de segundo orden en el paradigma de programación. Mientras estos utilizan simples funciones, mensajes o equivalentes, la POA permite razonar en términos de conjuntos de dichas entidades a través del uso de los puntos de corte. De esta forma se puede ver a la POA como una poderosa extensión lógica más que como un nuevo paradigma. (13)

1.5.3 Lenguajes orientados a aspectos (LOA)

En la actualidad existen diferentes LOA que posibilitan mayor facilidad a la hora de tratar con este paradigma, a continuación se realiza un breve análisis de los más representativos en estos últimos tiempos:

COOL

COOL (*COOrdination Language*) es un lenguaje de dominio específico, desarrollado por *Xerox*, cuya finalidad es tratar los aspectos de sincronismo entre hilos concurrentes. El lenguaje base que utiliza es Java. (12)

En COOL, la sincronización de los hilos se especifica de forma declarativa y, por lo tanto, más abstracta que la correspondiente codificación en Java.

RIDL

RIDL (*Remote Interaction and Datatransfersaspect Language*) es un LOA de dominio específico que maneja la transferencia de datos entre diferentes espacios de ejecución.

Un programa *RIDL* consiste de un conjunto de módulos de portales. Un portal es el encargado de manejar la interacción remota y la transferencia de datos de la clase asociada a él, y puede asociarse como máximo a una clase. (12)

MALAJ

MALAJ (*MultiAspect LAnguage for Java*) es un LOA de dominio específico, focalizado en la sincronización y reubicación. Sigue la misma filosofía que *COOL* y *RIDL*, indicando que la flexibilidad ganada con los LOA de propósito general, puede potencialmente llevar a conflictos con los principios básicos de la POO. Por esta razón, *MALAJ* propone un LOA de dominio específico, donde varios aspectos puedan ser resueltos, cada uno especializado en su propia incumbencia. (12)

El propósito final de los creadores de *MALAJ* es cubrir un gran conjunto de aspectos específicos, más allá de las dos mencionadas anteriormente. (12)

AspectJ

AspectJ es un LOA de propósito general, que extiende el lenguaje *Java,* donde un aspecto es una clase, exactamente igual que las definidas para este lenguaje. Aunque

presenta una particularidad, porque pueden contener unos constructores de corte, que no existen en *Java*. Los cortes de *AspectJ* capturan colecciones de eventos en la ejecución de un programa. Estos eventos pueden ser invocaciones de métodos, de constructores y excepciones de señales y gestión. Los cortes no definen acciones, sino que describen eventos. En forma similar a *AspectC*++, en *AspectJ* se definen cortes, avisos y puntos de enlace. (12)

AspectC++

AspectC++ es un LOA de propósito general que extiende el lenguaje C++ para soportar el manejo de aspectos. Sintácticamente, un aspecto en este lenguaje es muy similar a una clase en C++. Sin embargo, además de funciones, puede definir avisos (*advice*). Luego de la palabra clave *advice*, una expresión de corte (*pointcut*) define el punto donde el aspecto modificará al programa (los puntos de enlace o *joinpoints*). (14)

De manera general, existen otros LOA que no son presentados en esta investigación, los cuales extienden principalmente hacia *Java* y *.Net*. Entre los lenguajes analizados, *AspectJ* es considerado como uno de los más robustos, convirtiéndose con el paso del tiempo, en el principal lenguaje del enfoque orientado a aspectos. Con relación a otros, específicamente basados en C++, su existencia es muy limitada. En la bibliografía consultada, se hace referencia a *AspectC*++ como el idóneo para el desarrollo de aplicaciones en C++, utilizando este enfoque (POA). La idea anterior descansa en el criterio: AspectC++ es un LOA definido pensando en AspectJ, potente y de fácil interacción con su lenguaje base.

1.6 Metodología de desarrollo

1.6.1 *XP* (*eXtreme Programming*)

Actualmente, los negocios operan en un entorno global que cambia rápidamente, donde es muy difícil obtener un conjunto completo de requisitos del *software* estables. La metodología *XP* esta definida especialmente para proyectos con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico. Está enfocada hacia pequeños y medianos equipos, centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de *software*. Se basa en una retroalimentación continua entre el

cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes y simplicidad en las soluciones implementadas. (15)

Igualmente *XP* refleja ciertos criterios que lo hacen compatibles con la POA destacándose:

- La simplicidad con relación al código de la aplicación, ya que las funciones que se entrecruzan están definidas dentro de los aspectos.
- Mejora la comunicación y el entendimiento del código entre los desarrolladores.
- Se conduce a un diseño simple incorporando la separación de incumbencias.
- La creciente modularidad del código hace más fácil mejorar el *software* y hacerle modificaciones. (17)

Según *Pressman* la metodología *XP* define 4 fases fundamentales (ver Figura 5), las cuales son expuestas a continuación:

- **Planeación:** la actividad de planeación comienza creando una serie de historias también llamadas (historias de usuarios) que describen las características y las funcionalidades requeridas para el *software* que se construirá. (15)
- **Diseño:** se sigue de manera rigurosa el principio MS (mantenerlo simple) ofreciendo una guía de implementación para una historia como está escrita, ni más ni menos. Apoya el uso de tarjetas CRC (colaborador- responsabilidad- clase) para identificar y organizar las clases que son relevantes para el incremento del *software* actual. (15)
- **Codificación:** después de diseñar las historias y realizar el trabajo de diseño preliminar el equipo no debe moverse hacia la codificación. Desarrollando una serie de pruebas de unidad que ejerciten cada una de las historias que vayan a incluirse en el lanzamiento actual (incremento del *software*).Una vez creada la prueba de unidad, el desarrollador es capaz de centrarse en lo que debe implementarse para pasar dicha prueba. Cuando el código está completo, la unidad puede probarse de inmediato, y así proporcionar una retroalimentación instantánea a los desarrolladores. (15)
- **Pruebas:** la creación de una prueba de unidad antes de comenzar la codificación es un elemento clave para el enfoque de la *XP*. Cuando las unidades individuales de prueba se organizan en un conjunto universal de pruebas, las pruebas de integración y validación del sistema pueden realizarse a diario. Esto proporciona al

equipo de desarrollo una indicación continua del progreso y también puede encender luces de emergencia previas si las cosas salen mal. (15)

"Arreglar problemas pequeños cada pocas horas toma menos tiempo que arreglar problemas enormes justo antes de la fecha límite" (15)

Las pruebas de aceptación son especificadas por el cliente, enfocándose en las características generales, la funcionalidad del sistema, elementos visibles y revisables por el cliente. Las pruebas de aceptación se derivan de las historias de usuario que se han implementado como parte de un lanzamiento de *software*. (15)

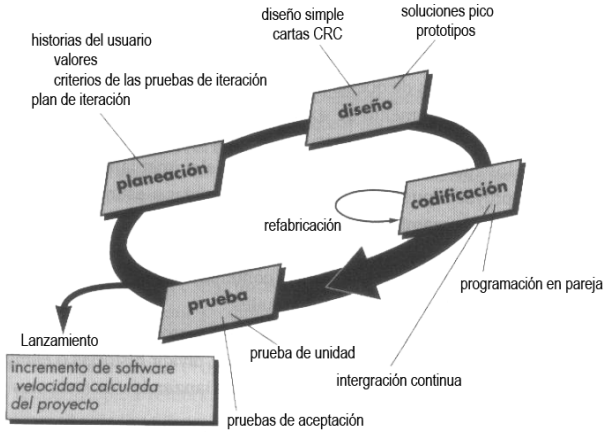


Figura 5 Fases de la metodología XP (15)

1.7 Herramientas de desarrollo

En la construcción de aplicaciones se involucran varios programas de software, lenguajes y tecnologías diferentes. Este proceso puede ser complejo y de larga duración por lo que las herramientas disponibles pueden reducir tiempo de desarrollo y aumentar el desempeño de los programadores y demás miembros del equipo.

1.7.1 *LOA AspectC++*

Para definir el lenguaje a utilizar, se establecen características esenciales que deberán fundamentar este objetivo. En la actualidad, *AspectJ* se expone como el LOA más difundido, pero presenta características que no hacen factible su utilización, ya que el lenguaje base que utiliza es *Java* y no C++, el cual es el definido para la implementación del SISCP-GALBA. Se debe destacar que otros proyectos, como *AspectC++,* son un enfoque de *AspectJ* pero orientado hacia C/C++. Criterio de selección con peso significativo para la selección del LOA a utilizar.

1.7.2 *Framework AspectC++ 1.2*

Esta herramienta es desarrollada para la construcción de aplicaciones orientadas a aspectos, basadas en el lenguaje AspectC++. Sus desarrolladores brindan un amplio soporte en caso de ser necesario y, mantienen una documentación actualizada. Este compilador presenta su código fuente de forma gratuita, cubierto por la GPL (Licencia Pública General). Este se integra en las distribuciones de *Debian* y *Ubuntu,* ya que la mayor parte de su desarrollo continúa haciéndose en *Linux*, aunque *Windows* se ha convertido en su segunda plataforma. Su última versión es la 1.2, lanzada en octubre de 2013 y se trabaja en su versión 2.0, con soporte preliminar para C++ 11 y mejoras en su rendimiento.

En la práctica, ac++ es un compilador para el lenguaje de programación *AspectC* ++, se implementa como un preprocesador que transforma (a través de la interfaz ag++, en caso de usar el compilador g++) código *AspectC*++ en código ordinario C++. Las definiciones de aspecto tienen que ser aplicadas en archivos de cabeceras de aspectos, que normalmente tienen la extensión de nombre de archivo .ah. Después de la transformación del código, la salida de ac++ puede servir como entrada a compiladores del lenguaje C++, como *GNU* g++, o *Microsoft VisualC++,* para finalmente obtener un código ejecutable. (18)

1.7.3 *Framework Qt*

Entre las tecnologías en auge que proporcionan un juego de herramientas y elementos gráficos para la creación de interfaces así como aplicaciones multiplataforma, se encuentra *Qt*. Es distribuido bajo los términos de *GNU Lesser General Public License*. El *API* (*Application Programming Interface*) de la biblioteca cuenta con métodos para acceder a bases de datos mediante *SQL*, uso de *XML* (*eXtensible Markup Language*), gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y otras herramientas para el manejo de ficheros, además de estructuras de datos tradicionales. Extiende el lenguaje de programación C++, a través de macros y meta-información. (19)

Otro de los elementos significativos de este *framework* es que permite una fácil integración con *AspectC++,* para la construcción de aplicaciones orientada a aspectos. Todo es a través de la herramienta *qmake,* cuyo comportamiento puede ser modificado a

través de una llamada al archivo de configuración acxx.prf, el cual le dirá que reemplace el compilador de C++ por ag++.

Qt cuenta con un robusto IDE (Entorno Integrado de Desarrollo), *QtCreator;* de código abierto y multiplataforma, creado por la empresa *Trolltech* para el desarrollo de aplicaciones utilizando las bibliotecas de *Qt*. Este ofrece múltiples herramientas entre las que se encuentran: editor de código fuente para los lenguajes de programación C++ y *JavaScript*, diseñador de interfaces gráficas de usuario, ayuda sensible al contexto y depurador de código fuente.

1.7.4 *eXtensible Markup Language* (*XML*)

Poder representar y hacer uso de las propiedades definidas en determinada estructura, es una de las características principales de *XML*. Posibilitando jerarquizar y estructurar la información, al describir los contenidos dentro del propio documento, así como la reutilización de partes del mismo.

El lenguaje *XML* se basa en el lenguaje *Unicode*[1], consiste de una serie de reglas, pautas o convenciones para planificar formatos de texto con tales datos. De manera que produzcan archivos que sean fácilmente generados y leídos por el desarrollador. Evitando los problemas más comunes como la falta de extensibilidad, de interoperabilidad entre plataformas o de soporte para universalizar su tratamiento. (20)

1.7.5 *Autotools*

Existen herramientas que se encuentran disponibles en la mayoría de los proyectos *open source,* que ayudan a la portabilidad de las aplicaciones a nivel de código fuente. Estas se abstraen, en la medida de lo posible, de versiones tradicionales disponibles en cada sistema operativo tipo *Unix*. En términos generales, es conveniente que las aplicaciones sean portables a muchas plataformas, debido que esto permite que sean empleadas por más personas sin requerirse ninguna adaptación. Una aplicación diseñada desde el inicio

[1] **Unicode** proporciona un número único para cada carácter, sin importar la plataforma, sin importar el programa, sin importar el idioma. Unicode representa cada carácter como un número de 2 bytes, de 0 a 65535. Cada número de 2 bytes representa un único carácter utilizado en al menos un idioma del mundo (los caracteres que se usan en más de un idioma tienen el mismo código numérico). Hay exactamente un número por carácter, y exactamente un carácter por número. Los datos de Unicode nunca son ambiguos.

para ser portable, tiene más posibilidades de adaptarse y sobrevivir en plataformas del futuro. (21)

Autotools son un conjunto de herramientas que facilitan la compilación de proyectos *software* en plataformas tipo *Unix, MacOS* e incluso *Windows*. Su empleo en el desarrollo de software libre cada día adquiere mayor relevancia, debido a que las aplicaciones cuentan con características como las antes mencionadas. Se basa en la construcción de programas de una manera más portable (*autoconf*), de tal forma que su código fuente pueda adaptarse a los diferentes sistemas. Del mismo modo, posibilita la generación automática de archivos m*akefile,* y provee funciones adicionales que permiten mejor control sobre los programas a través de una herramienta denominada *automake*. Facilita además, la creación de bibliotecas de software portables, simplificando el trabajo de los desarrolladores mediante el encapsulado de las dependencias específicas de la plataforma; y la interfaz del usuario en un único script. Lo anterior, se logra gracias una herramienta conocida como *libtool*.

Todas estas características sirven como un impulso para que proyectos como el SISCP-GALBA, dirija su mira al uso de esta forma de construir las aplicaciones para el control de sus procesos. Resaltan también, las ventajas que incluye para la generación de las bibliotecas, proveyéndole gran portabilidad, al poder ser utilizada en otras plataformas.

1.8 Lenguaje Unificado de Modelado (UML)

El modelado es una parte central de todas las actividades ayuda a visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de *software*. El Lenguaje Unificado de Modelado (*Unified Modeling Language*) proporciona una forma estándar de escribir los planos de un sistema, procesos del negocio, funciones del sistema, clases escritas en un lenguaje de programación específico, esquemas de bases de datos y componentes *software* reutilizables. (22)

1.8.1 Herramientas de modelado

Las herramientas CASE (*Computer Aided Software Engineering*), en español, Ingeniería de S*oftware* Asistida por Computadora, son aplicaciones informáticas utilizadas en el proceso de desarrollo de *software.* Ayudan en la realización de tareas como: diseñar un proyecto, cálculo de costos, implementación de parte del código automáticamente a partir del diseño, compilación automática y documentación o detección de errores. (23)

1.8.1.1 Visual Paradigm

Para el modelado será utilizado *Visual Paradigm* en su versión 8.0. Es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño, construcción, pruebas y despliegue. Incluye como lenguaje de modelado UML. Permite diseñar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Esta herramienta se caracteriza fundamentalmente por su agilidad en el proceso de modelado, así como por ser muy flexible con una organización lógica y deducible de sus componentes.

La portabilidad de la herramienta posibilita que pueda estar disponible en varias plataformas como Microsoft Windows (98, 2000, *XP*, Vista, 7), Mac OS y Linux, por solo mencionar algunas. (24)

Entre las ventajas que proporciona *Visual Paradigm* para UML se pueden mencionar: (24)

- Dibujo: facilita el modelado de UML, ya que proporciona herramientas específicas para ello permite la estandarización de la documentación, ya que la misma se ajusta al estándar soportado por la herramienta.
- Corrección sintáctica: controla que el modelado con UML sea correcto.
- Coherencia entre diagramas: al disponer de un repositorio común, es posible visualizar el mismo elemento en varios diagramas, evitando duplicidades.
- Integración con otras aplicaciones: permite integrarse con otras aplicaciones, como herramientas ofimáticas, lo cual aumenta la productividad.
- Reutilización: facilita la reutilización, ya que dispone de una herramienta centralizada donde se encuentran los modelos utilizados para otros proyectos.
- Generación de código: permite generar código de forma automática reduciendo los tiempos de desarrollo y evitando errores en la codificación del *software*.

1.9 Conclusiones parciales

En el capítulo, luego de abordarse los conceptos fundamentales, como mecanismos para encapsular las incumbencias transversales y la forma en que se reflejan en el SISCP-GALBA, se identificaron algunos de los problemas que este presenta a la hora de generar las trazas y gestionar excepciones. Se profundiza en el estudio de conceptos que brinda la POA, con el propósito de encapsular estos elementos fuera de la funcionalidad básica. Los mismos, le permitirán al desarrollador centrarse en los que son fundamentales para el

cumplimiento del objetivo principal de la aplicación. Igualmente se seleccionan las principales herramientas, tanto de modelado como de desarrollo, que serán utilizadas en la construcción de este mecanismo.

CAPÍTULO 2. PLANEACIÓN Y DISEÑO

Las etapas iniciales dentro de cualquier metodología de desarrollo son de gran importancia para el futuro de la aplicación. Es aquí donde se enmarcan los primeros elementos representativos y se identifican las pautas con que debe cumplir el *software* que se desea construir. En este capítulo son aplicadas las dos primeras etapas de la metodología *XP*, basándose en sus criterios fundamentales y denotando todos los artefactos que se van generando en cada una de estas.

2.1 Propuesta

Definir una infraestructura para la configuración de trazas y excepciones, proporciona al sistema y a los desarrolladores una forma flexible y novedosa de manejar estos elementos. Para la generación de trazas se utilizó la biblioteca *Log4cxx,* definida por el SISCP-GALBA. Son incorporados además, conceptos de la POA, lo cual posibilitará abstraer el concepto de trazas fuera de la funcionalidad básica. El nuevo mecanismo le permitirá al desarrollador definir los puntos específicos en la estructura (ejecución) de la aplicación donde desea que se generen las mismas. Dichos puntos serán registrados en un archivo *XML* (ver Figura 6) donde se describe con mayor precisión la acción que se realiza; descripción que será emitida junto al resto de los datos de la traza.

```
<?xml version="1.0"?>
<repository>
 <unit file="main.cpp">
  <JP id="6" signature="int main(int,char **)">
   <description value="default description..."/>
  </JP>
 </unit>

 <unit file="mainwindow.cpp">
  <JP id="25" signature="void MainWindow::on_pushButton_clicked()">
   <description value="Estableciendo comunicación con el resto de los módulos
  </JP>

  <JP id="24" signature="void MainWindow::on_pushButton_2_clicked()">
   <description value="Preparando el sistema para la recolección..."/>
  </JP>
```

Figura 6. Estructura para la representación de los puntos de enlaces (JP), en el XML

Para la generación de excepciones se propone el fortalecimiento de la biblioteca definida (*IOException*), al permitir una definición y manejo declarativos de las mismas, que brinda la posibilidad al desarrollador de crear sus propias situaciones de error. Se definen también varias opciones a la hora de lanzar una excepción, al realizarse de la manera tradicional o haciendo uso de las que se agregan a la biblioteca. Las nuevas opciones se

centran en utilizar las características de las excepciones descritas en un archivo *XML* (ver Figura 7), siguiendo una estructura conformada por los elementos: código, tipo de excepción y lenguaje. Un ejemplo concreto sería, lanzar una excepción indicando únicamente su código y, a partir de este, emitir su mensaje de error correspondiente.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--LEYENDA:
Codigos: <E0>, <E1>,..., <En>: codigos, donde 'n' es el numero del error
Idiomas: <es>: español   <en>: ingles   <ptg>: portugues ....
Tipos: <LP>: Logging-Presentation    <L>: Logging   <P>: Presentation
-->
<exceptions>
<E0 Lang="es">
<type value="LP"/>
<es message="Mensaje."/>
<en message="Message."/>
</E0>
</exceptions>
```

Figura 7. Estructura para la representación de las excepciones, en el XML

Una segunda situación dada al especificar, además del código, el idioma en el que se desea emitir su mensaje. Esta última, pone de manifiesto un concepto denominado, internacionalización de las excepciones; al definir un único mensaje de respuesta en diferentes idiomas, para una misma situación de error. Esto permitirá un mejor control, si se habla de integridad y versatilidad de la aplicación.

Definidas estas representaciones, solo quedará que ante una determinada situación, el sistema pueda actuar de acuerdo al tipo de la excepción emitida (*logging*- presentación, presentación, *logging*). La POA se encargará de que la aplicación responda según los términos antes mencionados, al realizar una captura y manejo de las excepciones en un único punto del código, mediante los mecanismos especificados en la definición del aspecto; abstrayendo esta operación de la funcionalidad básica. Es importante aclarar que este manejo no está determinado solamente por el tipo de la excepción, se debe tener en cuenta además, la situación específica que la generó.

Luego de haber realizado la propuesta inicial, se presentan las primeras fases (planeación y diseño) de la metodología de desarrollo seleccionada, las cuales son vitales para la correcta construcción del *software* en su ciclo de vida.

Requisitos del sistema

La identificación de los requisitos permite tener una correcta visión del producto que se quiere construir, son la descripción de los servicios que proporciona y sus restricciones operativas. Estos reflejan las necesidades de los clientes de un sistema, que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar una información. En algunos casos, un requisito es simplemente una declaración abstracta de alto nivel de un servicio que debe proporcionar el sistema, o una restricción del mismo. Se clasifican en requisitos funcionales y no funcionales, exponiendo a continuación los que se establecen para esta investigación.

Requisitos funcionales

Los requisitos funcionales describen lo que el sistema debe hacer, los servicios que debe proporcionar, como debe reaccionar a una entrada particular y ante situaciones específicas. (16)

RF 1: Almacenar trazas.

RF 2: Generar trazas.

RF 3: Definir excepción a través del *XML*.

RF 4: Modificar la excepción mediante *XML.*

RF 5: Eliminar excepción mediante *XML.*

RF 6: Lanzar excepciones.

RF 7: Capturar excepciones.

Requisitos no funcionales.

Los requisitos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de este cómo la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento, se denominan cómo requisitos no funcionales. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y las representaciones de datos que se utilizan en las interfaces del sistema. (16)

Requisitos de estándares:

- Para la generación de las trazas, se deberán mantener los estándares definidos en la biblioteca *Log4cxx* para el SISCP-GALBA. Las mismas deben ser legibles y auditables.

Requisitos de fiabilidad:

- Presenta un registro de las principales acciones que se realicen en el módulo HMI.

Requisitos de *software.*

- Sistema Operativo GNU/ Linux Debian 6 o 7 y compilador AspectC++ versión 1.2.

2.2 Planeación

La planeación es la etapa inicial de todo proyecto en *XP*. En este punto se comienza a interactuar con el cliente y el resto del grupo de desarrollo para descubrir los requisitos del sistema. En este instante se identifica el número y tamaño de las iteraciones al igual que se plantean ajustes necesarios a la metodología según las características del proyecto. El sistema es desarrollado para el cliente, por lo tanto, el usuario es quien decide que tareas realizará la aplicación. Este planteamiento se desarrolla a lo largo del proyecto: el cliente es quien decide qué hacer. Como primer paso, se debe proporcionar una idea clara de lo que será el proyecto en sí. (15)

2.2.1 Historias de usuario

Las Historias de Usuario (HU) son utilizadas como herramienta para dar a conocer los requisitos del sistema al equipo de desarrollo. Son pequeños textos en los que el cliente describe una actividad que realizará el sistema; la redacción de los mismos no se realiza bajo la terminología del desarrollador, de forma que sea clara y sencilla, sin profundizar en detalles. (15)

A continuación se muestran las principales HU extraídas para complementar esta investigación.

Historia de Usuario	
Número: 1	**Usuario:** Aplicación
Nombre historia: Generar trazas.	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta
Tiempo de Estimación: 3 semanas	**Iteración asignada:** 1
Descripción: El sistema deberá ser capaz de generar trazas de las acciones principales que realiza la aplicación: al cargar los servicios, en la ocurrencia de una alarma, al cambiar alguno de los estados de los servicios que brinda el sistema, al establecerse la conexión. Cuando se identifique alguna situación de error durante la	

ejecución de la aplicación, se debe mostrar el mensaje asociado a dicho error.

Tabla 1. HU Generar trazas

Historia de Usuario	
Número: 2	**Usuario:** Aplicación
Nombre historia: Almacenar trazas.	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Media
Tiempo de Estimación: 1 semana	**Iteración asignada: 1**
Descripción: Al ser generadas las trazas, estas deberán ser almacenadas en un archivo *.log* bajo las normas establecidas por el SISCP-GALBA y estandarizadas con la biblioteca *Log4cxx*.	

Tabla 2. HU Almacenar trazas.

Historia de Usuario	
Número: 3	**Usuario:** Desarrollador
Nombre historia: Definir excepción a través del *XML.*	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta
Tiempo de Estimación: 2 semanas	**Iteración asignada: 2**
Descripción: Se definirán las excepciones bajo la estructura especificada en el *XML* (código de la excepción, tipo de excepción, lenguaje). De esta forma, se podrán construir todas las que el desarrollador desee.	

Tabla 3. HU Definir excepción a través de XML.

Historia de Usuario	
Número: 4	**Usuario:** Desarrollador
Nombre historia: Modificar la excepción mediante *XML.*	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta
Tiempo de Estimación: 1 semana	**Iteración asignada: 2**
Descripción: Al ser definidas las excepciones en el archivo *XML*, se deberá permitir al desarrollador agregar o cambiar ciertas características dentro de la propia estructura. Al definir el mensaje de una excepción en un determinado idioma, se debe poder agregar para dicha excepción, el mismo mensaje, pero en un idioma diferente. También permitirá modificar el tipo de excepción definida en caso de haberse cometido	

algún error.

Tabla 4. HU Modificar la excepción mediante XML.

Historia de Usuario	
Número: 5	**Usuario:** Desarrollador
Nombre historia: Eliminar excepción mediante *XML.*	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta
Tiempo de Estimación: 1 semana	**Iteración asignada: 2**
Descripción: Debe permitir al desarrollador la eliminación de las excepciones que se definan en el *XML*.	

Tabla 5. HU Eliminar excepción mediante XML.

Historia de Usuario	
Número: 6	**Usuario:** Desarrollador
Nombre historia: Lanzar excepción.	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta
Tiempo de Estimación: 2 semanas	**Iteración asignada: 2**
Descripción: Realizará el lanzamiento de excepciones a partir de las características definidas y agregadas a la biblioteca *IOException*. Se introducen dos nuevas vías para hacerlo: al indicar solo el código de la excepción, definida en el *XML* o, junto al código, el idioma. Todo lo anterior, sin obviar las opciones que se encontraban en dicha biblioteca anteriormente.	

Tabla 6. HU Lanzar excepción.

Historia de Usuario	
Número: 7	**Usuario:** Aplicación
Nombre historia: Capturar excepción.	
Prioridad en el negocio: Alta	**Nivel de complejidad:** Alta
Tiempo de Estimación: 1 semana	**Iteración asignada: 2**
Descripción: Deberá ser capaz de capturar las excepciones lanzadas durante la ejecución de la aplicación (ej., módulo HMI). Luego de capturadas, serán tratadas bajo las normas que define el SISCP-GALBA.	

Tabla 7. HU Capturar excepción.

Las estimaciones de esfuerzo asociado a la implementación de las historias, se establecen utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos.

Historia de Usuario	Estimación por puntos
Generar trazas.	3
Almacenar trazas.	1
Definir excepción a través del *XML.*	2
Modificar la excepción mediante *XML.*	1
Eliminar excepción mediante *XML.*	1
Lanzar excepciones.	2
Capturar excepciones.	1

Tabla 8. Estimación de esfuerzos por Historias de Usuario.

Historia de Usuario	Tiempo estimado	Iteración	Tiempo real
Generar trazas. Almacenar trazas.	**4**	**1**	**4**
Definir excepción a través de *XML.* Modificar la excepción a través de *XML.* Eliminar excepción mediante *XML.* Lanzar excepciones. Capturar excepciones.	**7**	**2**	**7**

Tabla 9. Plan de duración de las iteraciones.

En la metodología *XP*, la creación del sistema se divide en etapas para facilitar su realización. Por lo general, los proyectos constan de más de tres etapas, las cuales toman el nombre de iteraciones, de allí se obtiene el concepto de metodología iterativa. La duración ideal de una iteración es de una a tres semanas. Según define esta metodología, se presenta seguidamente el plan de iteraciones propuesto.

2.2.2 Plan de iteraciones

Después de realizar un estimado de esfuerzo para cada historia de usuario y especificar un plan de entrega, se comienza a realizar la planificación para la etapa de implementación del sistema. En esta se detallan las historias de usuarios que serán implementadas por cada iteración, de acuerdo al orden definido. A continuación se realiza un análisis del sistema en dos iteraciones:

Iteración 1

En la primera iteración se define como objetivo, implementar las tareas esenciales para el desarrollo de las trazas, que son reflejadas mediante las siguientes historias de usuario: Generar trazas y Almacenar trazas. Finalizando la iteración, se obtendrá la primera parte del sistema, el cual debe ser mostrado al cliente con el objetivo de buscar una aceptación de los elementos desarrollados.

Iteración 2

En la segunda iteración se implementan todos los elementos correspondientes al desarrollo de las excepciones. Los mismos son reflejados a través de las historias de usuario: Definir excepciones a través de *XML*, modificar excepciones mediante *XML*, eliminar excepciones mediante *XML*, lanzar y capturar excepciones. Luego de culminado, será sometido a la fase de prueba, donde debe ser validado para la culminación de la aplicación desarrollada.

Iteración	Historias de Usuario	Estimación por puntos
1	Generar trazas.	4
	Almacenar trazas.	
2	Definir excepción a través de *XML.*	7
	Modificar la excepción a través de *XML.*	
	Eliminar excepción mediante *XML.*	
	Lanzar excepciones.	
	Capturar excepciones.	
Total		11

Tabla 10. Plan de duración de las iteraciones.

Historias de usuarios divididas por tareas

Dentro de cada una de las HU, se hace necesario identificar un número de tareas que permitirán tener una mejor visión para el desarrollo de la aplicación. Las mismas son especificadas por la iteración a la cual corresponde su HU. Seguidamente se muestra la relación de las tareas por cada iteración.

Historias de Usuario	Tareas por HU
Generar trazas.	Definir los aspectos para generar trazas.
Almacenar trazas.	Definir el archivo de configuración *logger.conf.*

Tabla 11. Tareas abordadas para la iteración 1.

Historias de Usuario	Tareas por HU
Definir excepción a través de *XML*.	Definir la estructura que deben tener las excepciones en el *XML*. Parsear el archivo.*XML* y obtener información de sus nodos. Generar la instancia de la clase *IOException* con los datos obtenidos en el *XML*.
Modificar la excepción a través de *XML*.	Realizar modificación en el archivo.*XML*, a la excepción deseada.
Eliminar excepción mediante *XML*.	Borrar del *XML* la excepción deseada.
Lanzar excepciones.	Definir una interfaz para acceder a los datos previamente declarados en el *XML*.
Capturar excepciones.	Definir los aspectos para capturar excepciones. Realizar el manejo de las excepciones según su tipo: LP (*logging- presentation*), L (*logging*), P (*presentation*).

Tabla 12. Tareas abordadas para la iteración 1.

El plan de entregas especifica, qué historias de usuario serán implementadas en cada entrega del sistema y sus prioridades, de modo que también permita conocer con exactitud qué historias de usuario serán implementadas en la próxima liberación. Debe ser negociado y elaborado en forma conjunta entre el cliente y el equipo desarrollador.

Historia de Usuario	01/04/2014	15/04/2014	20/05/2014
Generar trazas.	1.0	1.1	
Almacenar trazas.	1.0	1.1	
Definir excepción a través de *XML.*	Ya iniciado	2.0	2.1
Modificar la excepción a través de *XML.*	Ya iniciado	2.0	2.1
Eliminar excepción mediante *XML.*	Ya iniciado	2.0	2.1
Lanzar excepciones.	Ya iniciado	2.0	2.1
Capturar excepciones.	Ya iniciado	2.0	2.1
1.0 Entrega de la primera versión correspondiente a la iteración 1. **1.1** Entrega de la segunda versión y representa el final de la iteración 1. **2.0** Entrega de la primera versión correspondiente a la iteración 2. **2.1** Entrega de la segunda versión y representa el final de la iteración 2.			

Tabla 13. Plan de entregas.

2.3 Diseño del sistema

La metodología *XP* sigue de manera rigurosa el principio MS (mantenerlo simple). Siempre se prefiere un diseño simple respecto de una presentación más compleja. Esto ofrece una guía de implementación, para una historia como está escrita. *XP* define el uso de tarjetas CRC (Contenido o Clase, Responsabilidad y Colaboración) como un mecanismo efectivo para identificar y organizar las clases que son relevantes para el incremento del *software* actual. (16)

Cada tarjeta representa una clase con su nombre en la parte superior, en la sección izquierda están descritas las responsabilidades y a la derecha las clases que le sirven de soporte. A continuación se muestran las CRC realizadas para esta etapa.

Tarjetas CRC (Contenido o Clase, Responsabilidad y Colaboración)

Tarjetas CRC	
Clase: *IOException*	
Responsabilidad	**Colaboración**
***what()* :** Método encargado de mostrar el mensaje de la excepción.	*IOE_Manager* *std::exception*

Tabla 14. Tarjeta CRC de la clase IOException.

Tarjetas CRC	
Clase: *IOE_Manager*	
Responsabilidad	**Colaboración**
openDoc(): Se define para cargar el archivo *XML* y dejarlo preparado para ser utilizado. ***parseDoc():*** Método que funciona como interfaz entre *IOException* y *IOE_Manager*. ***searchNode ()*:** Busca un nodo (*exception*) en el *XML* a partir de los parámetros especificados. ***parseNode():*** Una vez encontrado el nodo buscado, se procede a su parseo, para obtener la información requerida. ***registerError():*** De no ser encontrado el nodo buscado, se procede a registrar un nuevo nodo con la información correspondiente.	

getInstance(): Devuelve una instancia estática de la clase IOE_Manager.	

Tabla 15. Tarjeta CRC de la clase IOE_Manager

Tarjetas CRC	
Clase: *XMLApplicationContext*	
Responsabilidad	**Colaboración**
updateContext(): Encargado de re-cargar el archivo *XML* y dejarlo preparado para ser utilizado. ***getDescription():*** Método que funciona como interfaz entre *IOManagerTrace* y *XMLApplicationContext*. *searchUnitByFilePath():* Busca un nodo (unidad [.cpp]) en el *XML* a partir del parámetro especificado. ***searchJPById():*** Una vez encontrada la **unidad**, se procede a buscar el punto de enlace según el parámetro especificado. ***registerJoinPoint():*** De no ser encontrado el punto de enlace o su unidad correspondiente, se procede a su registro dejando plasmada la información tanto del punto de enlace como de la unidad a la que pertenece.	*ApplicationContext*

Tabla 16. Tarjeta CRC de la clase XMLApplicationContext.

Tarjetas CRC	
Clase: *applicationContext*	
Responsabilidad	**Colaboración**

updateContext()*:** Encargado de re-cargar el archivo *XML* y dejarlo preparado para ser utilizado. ***getDescription(): Método que funciona como interfaz entre *XMLApplicationContext* y *applicationContext.*	

Tabla 17. Tarjeta CRC de la clase applicationContext.

Tarjetas CRC	
Clase: *ErrorLogManager*	
Responsabilidad	**Colaboración**
set()*:** Método que procesa el error a partir de un código de error. ***setLogPath(): Método para establecer la ruta de donde se va a obtener el archivo de configuración de los *logs*. ***Instance ():*** Funcionalidad que devuelve un apuntador a la única instancia de la clase en el sistema.	

Tabla 18. Tarjeta CRC de la clase ErrorLogManager.

Tarjetas CRC	
Clase: *IOManagerTraceDerived*	
Responsabilidad	**Colaboración**
***pointcut** joinPointCollector():* Definición del punto de corte para la recolección de los puntos de enlaces correspondientes al mismo. La acción a realizar estará definida mediante la implementación del (de los*) advice(s)* necesarios. ***pointcut** ioManagerCahtException():*Definición del punto de corte para la captura de excepciones generadas en los puntos de enlaces correspondientes al mismo. La acción a realizar	*IOManagerTrace* *IOManagerException*

estará definida mediante la implementación del (de los) *advice(s)* necesarios.	

Tabla 19. Tarjeta CRC de la clase IOManagerTraceDerived.

2.3.1 Diagrama de paquetes

La metodología *XP* no especifica la creación del diagrama de paquetes. Se genera este artefacto con el objetivo de documentar la vinculación entre el HMI y los nuevos conceptos incluidos para trazas, captura de excepciones y las bibliotecas *Logger* e *IOException* (ver Figura 8).

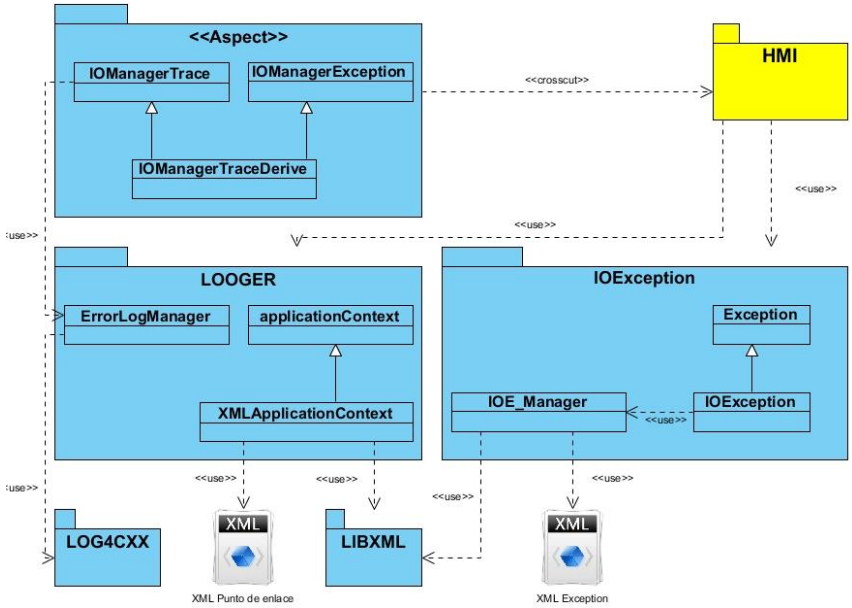


Figura 8. Diagrama de paquetes que vincula los paquetes Aspect, IOException y Logger con el módulo HMI

Aspect: este paquete se conforma por tres archivos cabeceras, los cuales contendrán todo el código de los aspectos que serán implementados. En *IOManagerTrace.ah* se definen los *advices* (consejos) necesarios para las acciones que se realizarán en determinados puntos de enlace (métodos), ordenando la lectura de los mismos, descritos en el archivo *XML.* Luego de obtener de este último, la descripción del punto de enlace, se genera una traza que incluye dicha descripción.

Se define además el archivo *IOManagerException.ah,* que especifica los *advices* (consejos) necesarios para las acciones que se realizarán en determinados puntos de enlace. En este caso, interesa capturar las excepciones lanzadas desde los mismos, por

lo que se precisan las acciones (capturas) a realizar para que el sistema responda de manera satisfactoria ante tales situaciones, esto incluye, el registro de las trazas que las describan, si su tipo (logging ó, logging-presentacion) lo requiere.

Por último, *IOManagerTraceDerive.ah:* en el que se establecen los puntos de corte para la captura de excepciones y generación de trazas. Se definen los sitios concretos sobre qué clase se desea que actúe el código de aspecto. Este paquete ***Aspect,*** opera de forma transversal sobre cualquier aplicación (C/C++), en este caso sobre el módulo HMI. Se impone destacar que, dicho código de aspecto no realiza cambios, ni agrega ningún elemento dentro de la estructura básica de dicho módulo.

Looger: este paquete representa la biblioteca ***Looger,*** y se estructura bajo tres clases fundamentales. *ErrorLogManager:* hace uso de la biblioteca *Log4cxx* para la generación de trazas. En esta clase se configuran los tipos de trazas y, bajo qué características se van a generar. La clase *XMLapplicationContext* es la encargada de efectuar un parseo por cada uno de los nodos donde se definieron los puntos de enlaces y, en caso de no encontrarse en el archivo, deberá generarlo.

IOException: en este paquete se representa la biblioteca ***IOException,*** está compuesta por tres clases fundamentales. *Exception:* es la clase establecida para las excepciones que genera el propio sistema. *IOException:* hereda de la anteriormente mencionada, y su propósito de mostrar el mensaje de la excepción. *IOE_Manager* se describe como la clase principal de esta estructura. La misma realiza todo lo correspondiente al parseo del archivo *XML* (*exceptionsDefines*) para obtener información que se requiere para una excepción en específico y, en caso de no contenerla en el archivo, deberá generarla.
XML **Punto de enlace:** archivo en el que se definen todos los puntos de enlaces con sus descripciones.

XML **Excepción:** archivo en el que se definen todas las excepciones que serán emitidas en la aplicación.

HMI: representa al módulo HMI del SISCP- GALBA el cual hace uso de las bibliotecas ***IOException*** y ***Looger.*** Con la primera, se permite la definición de las excepciones, teniendo un manejo declarativo de las mismas. En la segunda se especifican los posibles puntos de enlaces en los cuales serán generadas las trazas. El paquete ***Aspect*** se define de manera transversal, ejecutando las acciones definidas para la generación de trazas y capturar las excepciones que sean emitidas en dicho módulo.

2.3.2 Integración al sistema SISCP-GALBA

El diseño de la arquitectura de un sistema es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. En este se concretan los componentes que forman el sistema, la relación entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios de calidad indicados como seguridad, disponibilidad, eficiencia o usabilidad. (3)

El SISCP-GALBA presenta una arquitectura distribuida orientada a componentes, donde cada uno de los módulos se relaciona a través de la capa de comunicación. Esta arquitectura precisa de otros conceptos referentes a la modularidad y la reutilización del sistema, debido a la presencia de funcionalidades que normalmente abarcan a toda la aplicación. Específicamente, en esta investigación se abordan dos de estos elementos determinados por el registro de trazas y manejo de excepciones. Por lo anterior, se propone una infraestructura que centralice estos conceptos que la afectan de forma transversal.

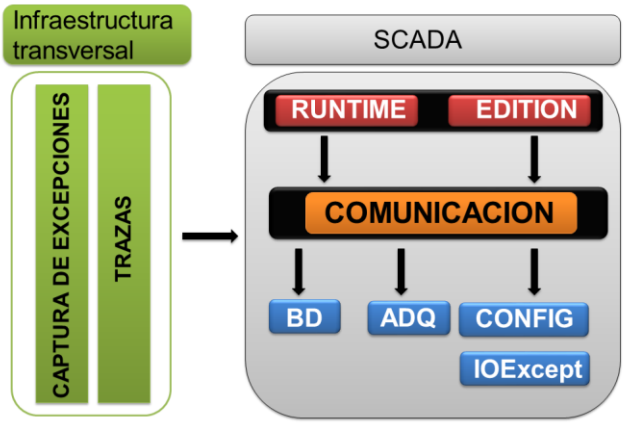


Figura 9 Integración al sistema

Dicha infraestructura (ver Figura 9) encapsula el código encargado de generar trazas y capturar excepciones. Al alcanzar este propósito permitirá que en un futuro, si se desea cambiar el comportamiento de dichos aspectos, solamente se centren en un área concreta. Además proporcionará una mejor organización debido a que se centralizan conceptos fuera de la funcionalidad básica, y no se dispersan por diferentes partes del sistema. De igual forma, a la hora de diseñar esta estrategia se considera como una pauta

esencial que, de no tenerla en cuenta, pudiera impactar negativamente en el rendimiento de la aplicación.

Patrones de diseño

En el uso de la POA y los patrones de diseños existen criterios que plantean que si se utilizan patrones, no es necesario utilizar POA. Estos ofrecen soluciones genéricas y la mayoría de los programadores que utilizan la POO se familiarizan con su uso pero, hay ocasiones en las que están presentes algunas desventajas:

- Pueden afectar a más de una clase
- Son intrusivos
- Su reutilización a nivel de código fuente es difícil

El uso de POA puede enriquecer la implementación de estos patrones ayudando a superar las desventajas planteadas, disminuyendo su complejidad, facilitando su reutilización y aumentando el grado de flexibilidad. En algunos casos la POA puede llegar a reemplazar la utilización de ciertos patrones (8). En el mecanismo propuesto, no se plantea ningún patrón de diseño vinculado a los conceptos de POA. En el caso de la biblioteca *IOException* se utiliza el patrón *Singleton o Instancia Única* para la clase *IOE_Manager.* Con el uso de este patrón se garantiza que solamente sea una instancia de la clase, y provee un punto de acceso global a la misma para realizar el manejo de las excepciones.

2.4 Conclusiones parciales

En el presente capítulo se generaron los disímiles artefactos que propone la metodología de desarrollo seleccionada. A partir de diferentes intercambios entre el cliente y los desarrolladores se determinaron los principales requisitos con que debe cumplir el mecanismo propuesto, plasmado a través de las diferentes historias de usuarios. Además se realiza una estimación de esfuerzo por historias de usuario, el plan de iteraciones y el plan de entregas por las diferentes iteraciones presentadas. Se definen nuevos conceptos que permitan la generación de trazas y el fortalecimiento de la biblioteca *IOException*. Tal es el caso del manejo declarativo de las excepciones, y la internacionalización de las mismas. Se define el tipo de la excepción, que será tomado en cuenta para el proceso a realizar luego de ser capturada. Igualmente se realiza la propuesta sobre los principales conceptos que formarán parte del mecanismo y como se podrá integrar este al SISCP-GALBA.

CAPÍTULO 3. CODIFICACIÓN Y PRUEBAS

En el presente capítulo se hará referencia a la etapa de codificación y pruebas que plantea la metodología de desarrollo seleccionada. Se expondrán las consideraciones que se realizaron para fomentar las buenas prácticas de programación. Además se efectúan las pruebas de validación mediante el uso de las pruebas de aceptación. Las mismas son uno de los métodos para determinar si los objetivos planteados a través de las diferentes historias de usuarios fueron alcanzados.

3.1 Estándar de código empleado

Los diferentes estándares de codificación, o estilos de programación como también se les llama, definen la estructura y apariencia física del código, facilitando su lectura, comprensión y mantenimiento. Para la implementación de los diferentes aspectos que intervienen en este mecanismo, se utilizó como guía el documento de referencia: Estándares de codificación para C++, Proyecto SCADA Guardián del ALBA (25). Al hacer uso de este, se pretende fomentar las buenas prácticas, y definir estándares para mejorar la comunicación del equipo de trabajo y la gestión del conocimiento en general.

3.2 Diagrama de despliegue

En todo proyecto se hace necesario mostrar la arquitectura del sistema desde el punto de vista del despliegue de los artefactos del software. Estos artefactos representan elementos concretos en el mundo físico que son el resultado de un proceso de desarrollo. Se pueden mencionar los archivos ejecutables, bibliotecas, esquemas de bases de datos y archivos de configuración.

Los diagramas de despliegue se utilizan para mostrar la vista estática de un sistema, mostrando las relaciones físicas entre los componentes de hardware y software en el sistema final.

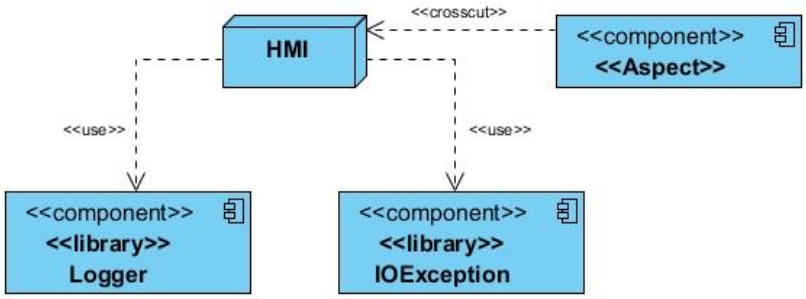


Figura 10. Diagrama de despliegue. Estructura final de integración.

En la figura 10, se representa la estructura final en donde el nodo HMI hace uso de los componentes *Logger* e *IOException.*

HMI: nodo donde se ejecutará el módulo HMI, el cual hará uso de las bibliotecas *Logger* e *IOException* para la definición de trazas y manejo de situaciones de error. El componente *Aspect* se refleja de forma transversal sobre el HMI encargándose de la generación de las trazas y la captura de excepciones que puedan ocurrir en dicho módulo.

3.3 Implementación

En esta fase de implementación las Historias de Usuarios definidas se dividen en tareas más pequeñas, denominadas tareas de ingeniería. Estas tienen el objetivo de realizar un análisis con mayor detalle y una estimación real de su tiempo de desarrollo.

Tarea de Ingeniería	
No. de tarea: 1	**No. de HU: 1**
Nombre de la tarea: Definir los aspectos para generar trazas.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 3
Descripción: Se deben definir los elementos principales de un aspecto (en el archivo *ioManagerTraceDerive.ah*): • Los puntos de corte (*pointcut*) que describen puntos específicos en la ejecución de la aplicación, donde actuará el aspecto. • Los *advice* que implementan la funcionalidad del aspecto para gestionar los diferentes puntos de enlace (*joinpoints),* que se especifican a través de los *pointcuts.* En este caso, en la implementación del *advice*, se determinada el código para generar las trazas.	

Tabla 20: Tarea 1 de la HU 1.

Tarea de Ingeniería	
No. de tarea: 2	**No. de HU: 2**
Nombre de la tarea: Definir el archivo de configuración *logger.conf.*	

Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Descripción: Se realiza la configuración en el archivo *logger.conf* para definir las características que la traza debe reflejar.	

Tabla 21: Tarea 2 de la HU 2.

Tarea de Ingeniería	
No. de tarea: 3	**No. de HU:** 3
Nombre de la tarea: Definir la estructura que deben tener las excepciones en el *XML.*	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0,4
Descripción: Se crea un archivo *XML (exceptionDefine)* en donde se especifica la estructura que seguirá al construir una excepción.	

Tabla 22: Tarea 3 de la HU 3.

Tarea de Ingeniería	
No. de tarea: 4	**No. de HU:** 3
Nombre de la tarea: Parsear archivo.*XML* y obtener información de sus nodos.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Descripción: Se hace uso de las funcionalidades especificadas por la biblioteca *libxml2* para definir una interfaz que permita acceder a los datos previamente declarados en el *XML* obteniéndose la información de las excepciones (nodos).	

Tabla 23: Tarea 4 de la HU 3.

Tarea de Ingeniería	
No. de tarea: 5	**No. de HU:** 3
Nombre de la tarea: Generar la instancia de la clase *IOException.*	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0,6
Descripción: Se crea una instancia de la clase *IOException* haciendo uso de la clase *IOEManager* para obtener los datos registrados en el archivo XML.	

Tabla 24: Tarea 5 de la HU 3.

Tarea de Ingeniería	
No. de tarea: 6	**No. de HU: 4**
Nombre de la tarea: Realizar modificación en el archivo.*XML sobre la* excepción deseada.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Descripción: Se realizan modificaciones en las propiedades de las excepciones definidas en el archivo *XML (*exceptionsDefines.xml*)* para su posterior utilización. Actualizando posteriormente la instancia que hace referencia a dicho archivo.	

Tabla 25: Tarea 6 de la HU 4.

Tarea de Ingeniería	
No. de tarea: 7	**No. de HU: 5**
• **Nombre de la tarea:** Borrar del *XML* la excepción deseada.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Descripción: Se elimina la excepción definida en el archivo *XML (exceptionsDefines.xml)*, actualizando posteriormente la instancia que hace referencia a dicho archivo.	

Tabla 26: Tarea 7 de la HU 5.

Tarea de Ingeniería	
No. de tarea: 8	**No. de HU:** 6
• **Nombre de la tarea:** Definir una interfaz para acceder a los datos previamente declarados en el *XML.*	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Descripción: Definir los métodos necesarios que implementen las funcionalidades que permitan acceder a los datos de las excepciones definidos en el archivo XML. Haciendo uso de la biblioteca *libxml2.*	

Tabla 27: Tarea 8 de la HU 6.

Tarea de Ingeniería	
No. de tarea: 9	**No. de HU: 7**
Nombre de la tarea: Definir aspecto para captura de excepciones.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 0,6
Descripción: Se deben definir los elementos principales de un aspecto (en el archivo *ioManagerTraceDerive.ah*): • Los puntos de corte (*pointcut*) que describen puntos específicos en la ejecución de la aplicación, donde actuará el aspecto. • Los *advice* que implementan la funcionalidad del aspecto para gestionar los diferentes puntos de enlace (*joinpoints),* que se especifican a través de los *pointcuts.* En este caso, en la implementación del *advice*, se determinada el código para capturar las excepciones lanzadas desde los *joinpoints*, y generar las trazas correspondientes.	

Tabla 28: Tarea 9 de la HU 7.

Tarea de Ingeniería	
No. de tarea: 10	**No. de HU:** 7
Nombre de la tarea: Realizar el manejo de las excepciones según su tipo: LP, L, P.	
Tipo de tarea: Desarrollo.	**Puntos estimados:** 1
Descripción: La clase *IOException* se encargará de generar un *log* con su información en caso que la excepción lanzada sea de tipo *logging*, haciendo uso de la biblioteca *escmga-hmi-logger*. De lo contrario se procede a realizar el manejo para las excepciones de presentación y *logging-presentation* a través de los aspectos definidos para ellos.	

Tabla 29: Tarea 10 de la HU 7.

3.4 Pruebas

Luego de haber culminado la fase de implementación se imponen una serie de pruebas que validen el correcto funcionamiento del mecanismo propuesto. Cada metodología específica una serie de pruebas para el cumplimiento de dicha validación. En el caso específico de la metodología *XP*, se basa en dos pruebas fundamentales las unitarias y

las pruebas de aceptación. Las primeras son realizadas de forma automática por el desarrollador haciendo uso de diferentes herramientas. El objetivo fundamental de las pruebas unitarias es asegurar el correcto funcionamiento de un método concreto. Seguidamente se reflejan las pruebas de aceptación realizadas para la validación del mecanismo propuesto.

3.4.1 Pruebas de aceptación

Para cada iteración se define un módulo o conjunto de historias que se van a implementar. Al final de este hito se obtiene como resultado la entrega del módulo correspondiente, el cual debe haber superado las pruebas de aceptación que establece el cliente para verificar el cumplimiento de los requisitos. Así se confirma que las historias de usuario se implementaron correctamente, ya que representan un resultado esperado de determinada transacción con el sistema. Para que una historia de usuario se considere aprobada, cada una de estas deberá pasar todas las pruebas de aceptación elaboradas para la misma.

Caso de Prueba de Aceptación	
Código: H1_P1	**HU: 1**
Nombre: Generar trazas.	
Descripción: Se comprueba si el sistema genera las trazas correspondientes, definidas mediante el mecanismo propuesto.	
Condiciones de Ejecución: El desarrollador debe haber definido los puntos de enlace donde actuará el aspecto.	
Entrada/Pasos de Ejecución: Ejecutar alguna de las acciones (*joinpoint*) descritas en los diferentes puntos de corte.	
Resultado Esperado: Debe ser generada la traza, creando un archivo *.log (en caso de no estar creado)* donde se registren las características de la misma.	

Resultado Obtenido: Se genera la traza, registrando en el archivo *.log*, la descripción de la acción realizada.

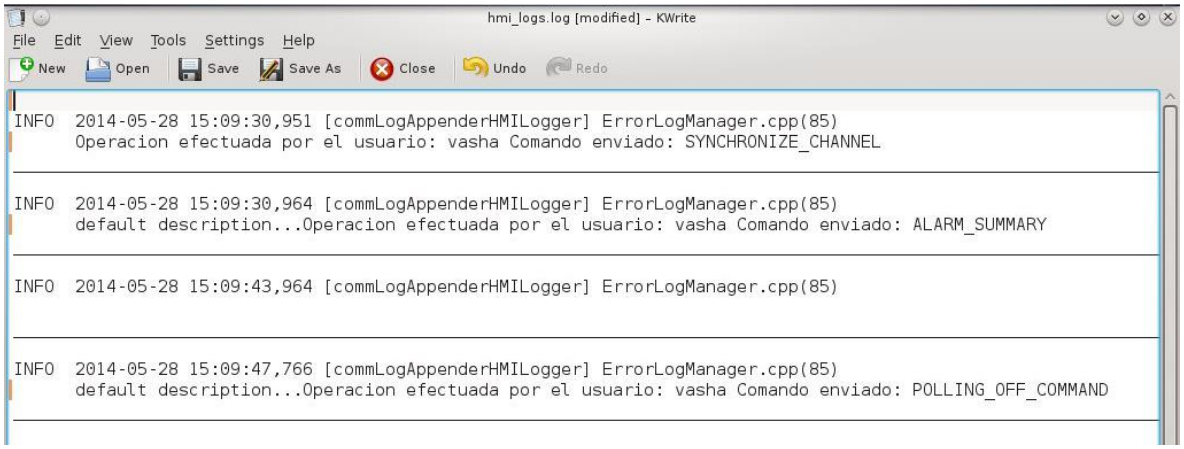


Figura 11. Se muestra la operación que realizó el usuario en la aplicación.

Evaluación de la Prueba: Se realiza la prueba obteniéndose resultados satisfactorios.

Tabla 30. Prueba de aceptación para la HU "Generar trazas".

Caso de Prueba de Aceptación	
Código: H2_P2	**HU: 2**
Nombre: Almacenar trazas.	
Descripción: Se comprobará si se genera un archivo *(.log)* en donde se almacenarán las características de las trazas generadas bajo estándares definidos en la biblioteca *Log4cxx.*	
Condiciones de Ejecución: Se debe realizar alguna acción en el módulo HMI.	
Entrada/Pasos de Ejecución: -	
Resultado Esperado: Debe ser registrada en un archivo (*.log*) las características de la traza creada.	
Resultado Obtenido: Se registra la traza en el archivo (*.log*), con la descripción de la acción realiza y, observada desde el código del aspecto.	

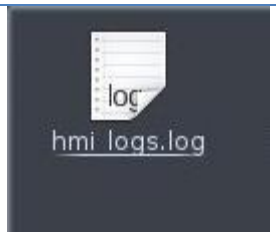


Figura 12. Archivo que contiene las trazas generadas.

Evaluación de la Prueba: Se realiza la prueba obteniéndose resultados satisfactorios.

Tabla 31. Prueba de aceptación para la HU "Almacenar trazas".

Caso de Prueba de Aceptación	
Código: H3_P3	**HU: 3**
Nombre: Definir excepción a través del *XML.*	
Descripción: Se comprueba si se definen correctamente nuevas excepciones en el archivo *XML*, respetando la estructura para la creación de las mismas.	
Condiciones de Ejecución: -	
Entrada/Pasos de Ejecución: -	
Resultado Esperado: Se establezcan correctamente las excepciones en el archivo XML, según la estructura definida para las mismas.	
Resultado Obtenido: Luego de lanzar una excepción que no había sido registrada, quedó definida correctamente su estructura en el archivo *XML*.	

```
<?xml version="1.0" encoding="UTF-8"?>
<!--LEYENDA:
Códigos: <E0>, <E1>,..., <En>: códigos, donde 'n' es el número del error asociado.
Idiomas: <es>: español   <en>: ingles   <ptg>: portugues ....
Tipos: <LP>: Logging-Presentation    <L>: Logging   <P>: Presentation
-->
<exceptions LastCode="133">
<E1 Lang="es">
 <type value="L"/>
 <es message="Operación no permitida"/>
</E1>
<E5 Lang="es">
 <type value="L"/>
 <es message="Error de entrada/salida"/>
</E5>
<E12 Lang="es">
 <type value="L"/>
 <es message="No se pudo asignar memoria"/>
</E12>
<E14 Lang="es">
 <type value="L"/>
 <es message="Dirección incorrecta"/>
</E14>
<E32 Lang="es">
 <type value="L"/>
 <es message="Tuberia rota"/>
</E32>
```

Figura 13. Definición de las excepciones siguiendo la estructura establecida.

Evaluación de la Prueba: Se realiza la prueba obteniéndose resultados satisfactorios.

Tabla 32. Prueba de aceptación para la HU "Definir excepción a través del XML".

Caso de Prueba de Aceptación	
Código: H4_P4	**HU: 4**
Nombre: Modificar la excepción a través de *XML.*	
Descripción: Se comprueba si el desarrollador puede realizar alguna modificación en la excepción creada.	
Condiciones de Ejecución: Se debe crear al menos una excepción con anterioridad.	
Entrada/Pasos de Ejecución: El desarrollador realizará la modificación deseada en el archivo *XML*.	
Resultado Esperado: La excepción pueda ser modificada en el *XML* por el desarrollador, y quedar lista para su reutilización.	
Resultado Obtenido: Se modificaron (en tiempo de ejecución) las características de la excepción, pudiendo luego ser utilizadas satisfactoriamente.	

```
<?xml version="1.0" encoding="UTF-8"?>
<!--LEYENDA:
Códigos: <E0>, <E1>,..., <En>: códigos, donde 'n' es el número del error asociado.
Idiomas: <es>: español   <en>: ingles   <ptg>: portugues ....
Tipos: <LP>: Logging-Presentation    <L>: Logging   <P>: Presentation
-->
<exceptions LastCode="133">
<E1 Lang="es">
<type value="L"/>
<es message="Operación denegada"/>
</E1>
<E5 Lang="es">
<type value="L"/>
<es message="Error de entrada/salida"/>
</E5>
<E12 Lang="es">
<type value="L"/>
<es message="No se pudo asignar memoria"/>
</E12>
<E14 Lang="es">
<type value="L"/>
<es message="Dirección incorrecta"/>
</E14>
<E32 Lang="es">
<type value="L"/>
<es message="Tubería rota"/>
</E32>
```

Figura 14. Se modifica el mensaje correspondiente a la excepción 133.

Evaluación de la Prueba: Se realiza la prueba obteniéndose resultados satisfactorios.

Tabla 33. Prueba de aceptación para la HU "Modificar la excepción a través de XML".

Caso de Prueba de Aceptación	
Código: H5_P5	**HU: 5**
Nombre: Eliminar excepción mediante *XML.*	
Descripción: Se comprueba si el desarrollador puede eliminar alguna de las excepciones creadas, evitando conflictos con la aplicación al ser ejecutada.	
Condiciones de Ejecución: Se debe haber creado al menos una excepción.	
Entrada/Pasos de Ejecución: -	
Resultado Esperado: Se logra eliminar la excepción deseada, sin crear ningún conflicto con el sistema.	
Resultado Obtenido: Se elimina (en tiempo de ejecución) una excepción definida en el archivo XML, y el desempeño del sistema no es afectado.	

```
<?xml version="1.0" encoding="UTF-8"?>
<!--LEYENDA:
Códigos: <E0>, <E1>,..., <En>: códigos, donde 'n' es el número del error asociado.
Idiomas: <es>: español   <en>: ingles   <ptg>: portugues ....
Tipos: <LP>: Logging-Presentation    <L>: Logging   <P>: Presentation
--->
<exceptions LastCode="133">
<E1 Lang="es">
<type value="L"/>
<es message="Operación denegada"/>
</E1>
<E5 Lang="es">
<type value="L"/>
<es message="Error de entrada/salida"/>
</E5>
<E12 Lang="es">
<type value="L"/>
<es message="No se pudo asignar memoria"/>
</E12>
```

Figura 15. Son eliminadas las excepciones 14 y 32.

Evaluación de la Prueba: Se realiza la prueba obteniéndose resultados satisfactorios.

Tabla 34. Prueba de aceptación para la HU "Eliminar excepción mediante XML".

Caso de Prueba de Aceptación	
Código: H6_P6	**HU: 6**
Nombre: Lanzar excepción.	
Descripción: Se comprueba si el desarrollador puede hacer uso de las nuevas formas de emitir las excepciones introduciendo el código de la excepción definida anteriormente o emitiendo un mensaje.	
Condiciones de Ejecución: -	
Entrada/Pasos de Ejecución: El desarrollador lanzará las excepciones bajo las normas definidas. *throw CDA:: Exception:: IOException(..)*	
Resultado Esperado: Al ejecutar la aplicación, si se cumplen las condiciones para el lanzamiento de la situación de error definida, el sistema debe ser capaz de realizar dicha acción.	
Resultado Obtenido: En situaciones de error establecidas, se lanzan las excepciones correspondientes, accediendo satisfactoriamente a su definición en el archivo *XML*.	

Evaluación de la Prueba: Se realiza la prueba obteniéndose resultados satisfactorios

Tabla 35. Prueba de aceptación para la HU "Lanzar excepción".

Caso de Prueba de Aceptación	
Código: H7_P7	**HU: 7**
Nombre: Capturar excepción.	
Descripción: Se comprueba si el mecanismo es capaz de capturar las situaciones de errores que se generan, posibilitando que la aplicación prosiga su ejecución.	
Condiciones de Ejecución: El desarrollador debe definir los puntos de corte establecidos para la captura de las excepciones que se presenten en la ejecución de la aplicación.	
Entrada/Pasos de Ejecución: -	
Resultado Esperado: El mecanismo deberá ser capaz de capturar las excepciones lanzadas, permitiendo que la aplicación continúe ejecutándose.	
Resultado Obtenido: Se captura satisfactoriamente la excepción lanzada posibilitando que la aplicación continúe su ejecución. La excepción generada es capturada y manejada según su tipo.	
Evaluación de la Prueba: Se realiza la prueba obteniéndose resultados satisfactorios.	

Tabla 36. Prueba de aceptación para la HU "Capturar excepción".

En la metodología XP, la aplicación de pruebas se realiza de forma iterativa para comprobar la correcta implementación de las HU. Luego de aplicarse para cada una de las iteraciones, se obtienen los siguientes resultados:

Iteración 1.

- Las trazas obtenidas no reflejan el usuario que realizó la acción registrada.

Iteración 2.

- Existen problemas a la hora de mostrar en la traza, el mensaje asociado a la excepción lanzada.
- El sistema es capaz de capturar las excepciones lanzadas, pero su ejecución se interrumpe.

Todas estas no conformidades son corregidas y son probadas nuevamente, obteniéndose resultados satisfactorios.

3.5 Conclusiones parciales

En el presente capítulo fueron abordadas diferentes aristas orientadas a la codificación y realización de las pruebas definidas para la validación de la solución propuesta. Se desarrollaron diferentes tareas que permitieron darle cumplimiento a las historias de usuario especificadas, cumpliendo de manera satisfactoria las pruebas de aceptación realizadas. Con la culminación de este capítulo se da por terminada la propuesta que argumenta el presente trabajo investigativo.

Conclusiones generales:

Con la realización de este trabajo se dotó al SISCP- GALBA, específicamente a su módulo HMI, de un mecanismo para la generación de las trazas y excepciones, que establece nuevos conceptos que permiten una mayor flexibilización a la hora de definir y emitir situaciones de error durante la ejecución del sistema.

Como constancia del cumplimiento al objetivo planteado al inicio de la investigación, se reflejan a continuación otros elementos significativos:

- Con el uso de la POA disminuyen los efectos que traen consigo las incumbencias transversales. Este enfoque fue utilizado en el SISCP-GALBA, específicamente en el módulo HMI, lo que posibilitó la generación de trazas y captura de excepciones en una zona específica del mismo (infraestructura transversal). Al igual, se logró que estos procesos se efectuarán de forma homogénea, al seguir, en el caso de las trazas, los estándares establecidos. Lo anterior permitió, poder llevar a cabo satisfactoriamente, procesos de auditorías al sistema.
- Con el uso de los métodos convencionales no se pudiese evitar obtener un código disperso por toda la aplicación o, por lo menos no de forma sencilla, si se habla en términos de implementación.
- Se posibilitó que el desarrollador contase con un manejo declarativo de las excepciones, brindándole un nuevo mecanismo para emitir las situaciones de error, sin dejar de tener en consideración los existentes hasta el momento.

Recomendaciones:

- Aplicar los resultados de la presente investigación a los demás subsistemas del SISCP-GALBA.
- Realizar un análisis de otros aspectos arquitectónicos que igualmente afectan de manera transversal al sistema; dígase gestión de la seguridad, conexiones a bases de datos, gestión de memoria (aspecto crítico en aplicaciones basadas en C++).
- Utilizar una interfaz gráfica para la edición y validación de los archivos xml, que contienen las descripciones de las excepciones y puntos de enlaces, respectivamente.

Referencias bibliográficas

1. **José Luis Romeral, Josep Balcells.** *Autómatas programables.* s.l. : Serie Mundo Electrónico, Marcombo editores.

2. [En línea] 2014. www.gedlc-ulpgc.es/docencia/excepciones.html..

3. **Cesar de la Torre Llorente, Unai Zorrilla Castro, Miguel Angel Ramos Barroso, Javier Calvarro Nelson.** *Guía de Arquiectura N- Capas orientado al dominio con .NET. 4.0. .* 2010.

4. Universidad Regiomontana. [En línea] 2014. www.ur.mx..

5. **Msc. Oiner Gómez Baryolo, Dra C. Isabel García Rodríguez.** [En línea] www.acimed.sld.cu..

6. *Generación de trazas con Log4j.* 2010.

7. [En línea] http://logging.apache.org/log4j/docs/documentation.html..

8. **Paez, Nicolás Martín.** *Utilización de programación orientada a aspectos en aplicaciones enterprise .* Universidad de Buenos Aires : s.n., 2007.

9. **LSC. Maria Zabala Hurtado, M.C Rene Edmundo Cuevas Valencia.** *Artefactos de diseño en el paradigma orientada a aspectos. .* 2013.

10. **Maria Celeste Gastaldo, Martin Santamaria.** *Programación Orientada a Aspectos - POA. .* Rosario, Argentina : s.n., 2008.

11. **Quintero, Antonia Maria Reina.** *Visión General de la Programación Orientada a Aspectos.* Universidad de Sevilla : Departamento de Lenguajes y Sistemas Informáticos, 2005.

12. **Joskowicz, Ing. José.** *Programación Orientado a Aspectos. .* 2008.

13. **Guillen., Salvador Manzanares.** *Programación Orientada a Aspectos. Una experiencia práctica con AspectJ.* Facultad de Informática de la Universidad de Murcia : s.n., 2005.

14. **Spinczyk, Olaf.** [En línea] www.aspectc.org.

15. **Pressman, Roger.** *Ingeniería del Software. Un enfoque práctico. 6ta Edición.*

16. **Sommerville, Ian.** *Ingeniería del software, Séptima Edición. .* Madrid : Pearson Educación S.A., 2005.

17. **Elhaide, Rodrigo H.** *Metodologías Ágiles y POA. Buenos Aires, Argentina : s.n. 1330-0081.*

18. **Spinczyk, Olaf.** *AC++ Manual del compilador.* septiembre, 2013.

19. [En línea] www.codigoqt.com..

20. **Marco Bartolome Sints. XML:Lenguaje de Marcas Extensible. .** [En línea] mayo de 2013. www.mclibre.org...

21. *Desarrollo Portable con GNU Autotools.*

22. **Craig., Larman.** *UML y Patrones. Introducción al análisis y diseño Orientado a Objetos.* s.l. : Prentice Hall.

23. **Asensio, Rafael Menendez Barzanallana.** *Herramientas CASE. Ingeniería de software. .* s.l. : Informática Aplicada a la gestión Pública, 2011.

24. *Ingenieria del software. Curso 2013- 2014. Guión Visual Paradigm para UML. . .*

25. *0120_1 Estándares de codificación para C++ Proyecto SCADA Guardián del ALBA. .*

26. **Spinczyk, Olaf.** *AspectC++ Quick Reference. .* 2012.

27. **BARYOLO., OINER GÓMEZ.** *TESIS DOCTORAL CAEM: MODELO DE CONTROL DE ACCESO PARA SISTEMAS DE INFORMACIÓN EN ENTORNOS MULTIDOMINIOS.* La Habana, Universidad de las Ciencias Informáticas : s.n., 2012.

28. **Alfredo Matteo, Patricia Morantes.** *Modelación de la Programación Orientada a Aspectos. .* Universidad Central de Venezuela : s.n., 2010.

29. *Metodologías ágiles y programación orientada a aspectos. .* Buenos Aires, Argentina : s.n., 2006. . 1330-0081..

30. **Luis M. Echeverry, Luz Elena Delgado.** *Caso práctico de la metodología ágil XP al desarrollo del sofware. .*

Bibliografía

Spinczyk, Olaf Spinczyk and Ute. *Using AspectC++ for QtApplication Development .* 2011.

Spinczyk, Olaf. *AspectC++ Quick Reference.* 2012 : s.n.

***Spinczyk, Olaf**. AC++ Manual del compilador. .* septiembre, 2013.

Zhen Yao, Qi-long Zheng, Guo-liang Chen. *AOP++: A Generic Aspect-Oriented Programming Framework in C++, .* China : s.n.

Durán, Alexei Ramírez. *Módulo de trazas para la realización de auditorías en el Sistema de Manejo Integral de Perforación de Pozos de Petróleo.* La Habana, Universidad de las Ciencias Informáticas : s.n., 2013.

[En línea] www.calcifer.org/documentos/autotools.

[En línea] www.aosd.net.

[En línea] www.gedlc-ulpgc.es/docencia/excepciones.html.

[En línea] crysol.github.10/recipe/distribuir programas con autotools.

[En línea] www.prezi.com/intervomafak/desarrollo de software orientado a aspectos.

[En línea] www.iti.cs.uni.magdeburg.de/iti_db/forschuns/fop/featurec.

BARYOLO, OINER GÓMEZ. *TESIS DOCTORAL CAEM: MODELO DE CONTROL DE ACCESO PARA SISTEMAS DE INFORMACIÓN EN ENTORNOS MULTIDOMINIOS.* La Habana, Universidad de las Ciencias Informáticas : s.n., 2012.

Elhaibe, Rodrigo H. *Metodologías ágiles y programación orientada a aspectos.* Buenos Aires, Argentina : s.n., 2006. 1330-0081.

Luis M. Echeverry, Luz Elena Delgado. *Caso práctico de la metodología ágil XP al desarrollo del sofware.*

Doce, Yanet Nieto. *Sistemas SCADA.* La Habana, Universidad de las Ciencias Informáticas : s.n., 2010.

Msc. Germán Alferez Salinas, Msc. Edward M. Alferez Salinas. *El desarrollo de software orientado a aspecto. Un caso práctico para un sistema de ayuda en línea.* 2008.

Alfredo Matteo, Patricia Morantes. *Modelación de la Programación Orientada a Aspectos.* Universidad Central de Venezuela : s.n., 2010.

Iglesias, Ing. Adriana. *Recolectando métricas para el desarrollo de la programación orientada a aspectos.* 2013.

Tesis Doctoral: Separación dinámica de aspectos independientes del lenguaje y plataforma mediante el uso de reflexión computacional.

José Luis Romeral, Josep Balcells. *Autómatas programables. .* s.l. : Serie Mundo Electrónico, Marcombo editores.

LSC. Maria Zabala Hurtado, M.C Rene Edmundo Cuevas Valencia. *Artefactos de diseño en el paradigma orientada a aspectos.* 2013.

Universidad Regiomontana. . [En línea] www.ur.mx.

Cesar de la Torre Llorente, Unai Zorrilla Castro, Miguel Angel Ramos Barroso, Javier Calvarro Nelson. *Guía de Arquiectura N- Capas orientado al dominio con .NET. 4.0. .* 2010.

Msc. Oiner Gómez Baryolo, Dra C. Isabel García Rodríguez. [En línea] www.acimed.sld.cu..

Maria Celeste Gastaldo, Martin Santamaria. *Programación Orientada a Aspectos - POA. .* Rosario, Argentina : s.n., 2008.

Quintero, Antonia Maria Reina. *Visión General de la Programación Orientada a Aspectos. .* Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla : s.n., 2005.

Joskowicz, Ing. Jose. *Programación Orientado a Aspectos. .* 2008.

Guillen, Salvador Manzanares. *Programación Orientada a Aspectos. Una experiencia práctica con AspectJ. .* s.l. : Facultad de Informática de la Universidad de Murcia, 2005.

Paez, Nicolás Martín. *Utilización de programación orientada a aspectos en aplicaciones enterprise.* Universidad de Buenos Aires : s.n., 2007.

Sommerville, Ian. *Ingeniería del software, Séptima Edición. .* Madrid : Pearson Educación S.A., 2005.

Pressman, Roger. *Ingeniería del Software. Un enfoque práctico.*

[En línea] www.aspectc.org.

CASTILLO, FABIO ENRIQUE. *PROGRAMACION ORIENTADA A ASPECTOS.* UNIVERSIDAD CATOLICA DE COLOMBIA : s.n., 2010.

[En línea] www.codigoqt.com..

[En línea] Marco, Bartolome Sints. XML:Lenguaje de Marcas Extensible. [En línea], mayo de 2013. www.mclibre.org..

Larman, Craig. *UML y Patrones. Introducción al análisis y diseño Orientado a Objetos.* s.l. : Prentice Hall.

Ingenieria del software. Curso 2013- 2014. Guión Visual Paradigm para UML. .

Especificaciones del Sistema de Supervisión y Control Guardián del ALBA desarrollado por PDVSA. 2013.

0120_1 Estándares de codificación para C++ Proyecto SCADA Guardián del ALBA.

Generación de trazas con Log4j.

[En línea] http://logging.apache.org/log4j/docs/documentation.html.

Guía para empresas:seguridad de los sistemas de monitorización y control de los procesos e infraestructuras (SCADA). 2012.

Desarrollo Portable con GNU Autotools.

Asensio, Rafael Menendez Barzanallana. *Herramientas CASE. Ingeniería de software.* s.l. : Informática Aplicada a la gestión Pública, 2011.

Caamaño, Germán Poo. *autotools: Herramientas para la creación de proyectos Open Source .* 2004.

Departamento de Lenguajes y Sistemas Informáticos, escuela técnica superior de ingeniería informática. *Introducción a los Patrones de Diseño Ingeniería, .*

Díaz, Carolina García Antón Günther Rodríguez. *Logs y Auditoría.*

Marta S. Tabares B., Germán H. Alferez Salinas, Edward M. Alferez Salinas,. *Red de Revistas Científicas de América Latina, el Caribe, España y Portugal Sistema de Información Científica.* 2008.

López, Abdel Pérez. *Desarrollo de un sistema para la gestión de las excepciones y trazas de software en una aplicación implementada sobre Arquitectura .NET.* La Habana : s.n., 2009.

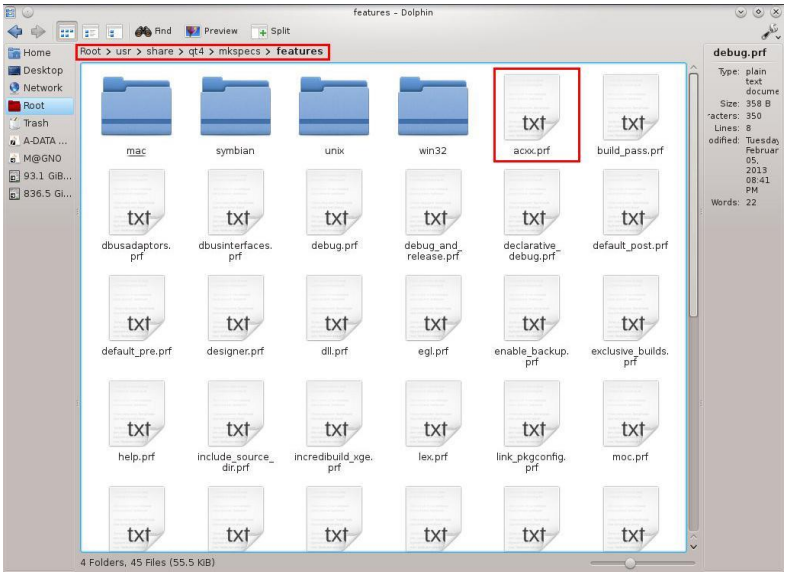
I.SC. Juan Alberto Hernández Martínez, Dr. Ulises Juárez Martíneza. *Revista Digital Universitaria.* 2011.

Egido, Alicia Fernández. *CONFLICTOS ENTRE ASPECTOS EN LA PROGRAMACIÓN ORIENTADA A ASPECTOS.* Universidad Carlos III de Madrid, : s.n., 2009.

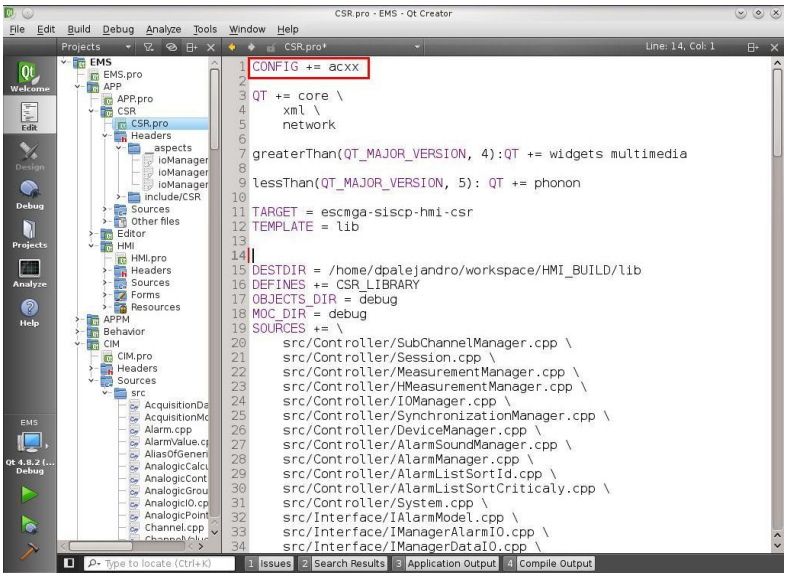
Esquemas XML para intercambio de documentos electrónicos y expedientes electrónicos. Madrid : s.n., 2011.

Anexos

Pasos para compilar la aplicación con AspectC++

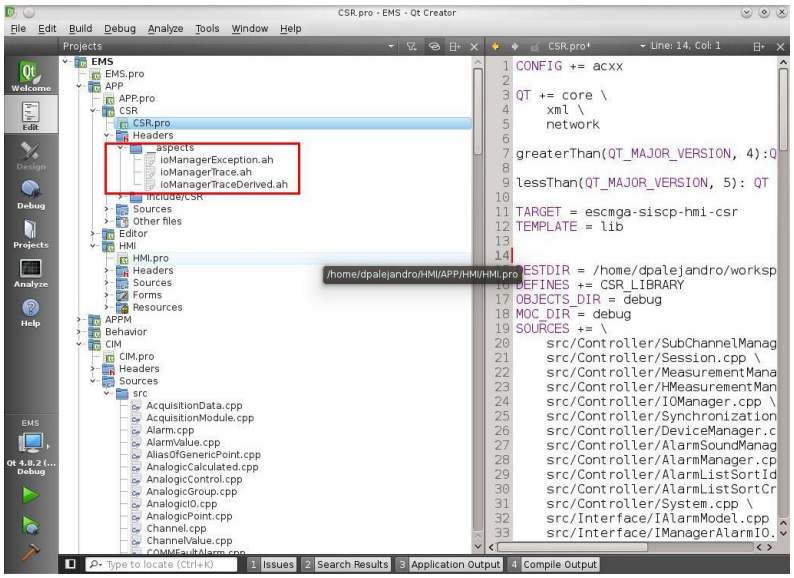


1. Copiar en la siguiente dirección ***Root>usr>share>qt4>mkspecs>features*** el archivo de configuración *acxx.prf* que al ser utilizado por qmake (herramienta para la construcción de los proyectos *Qt*), sustituye g++ por ag++. Propiciando que el proceso de entretejido con los aspectos se realice satisfactoriamente.

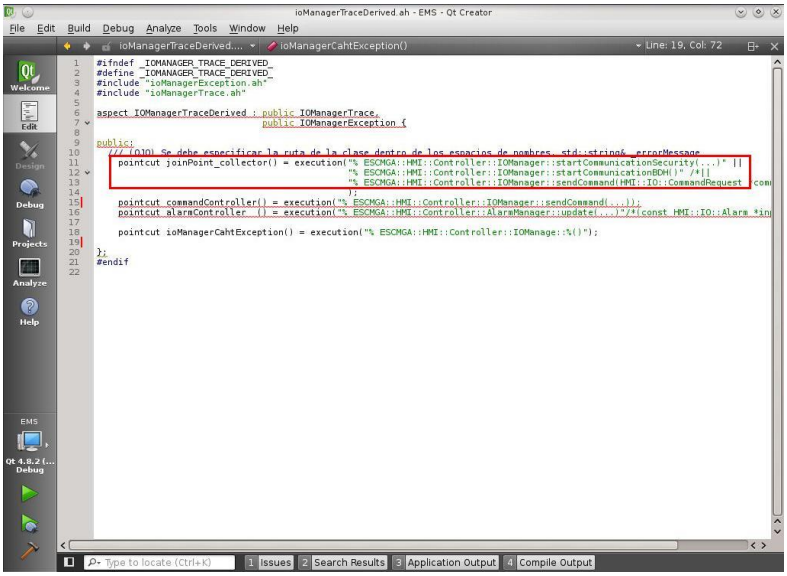


2. Se le indica al proyecto que modifique su variable de entorno para que pueda

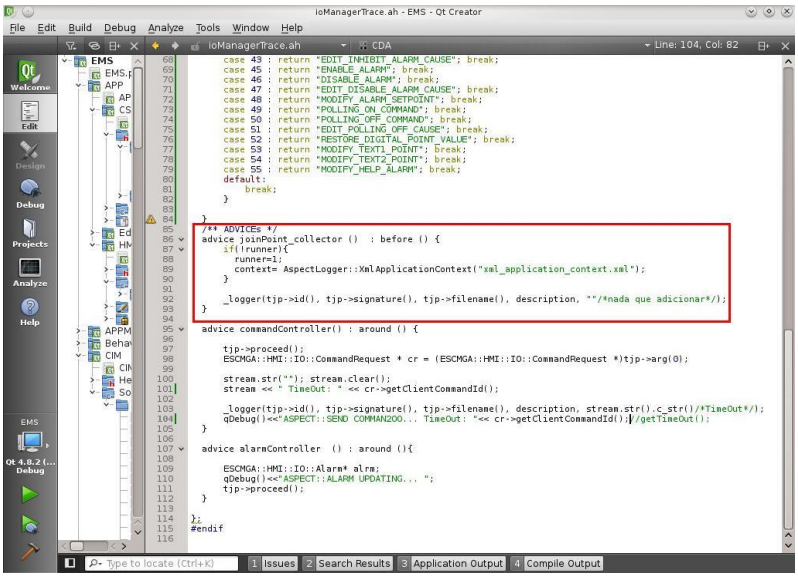
realizar un trabajo con AspectC++.



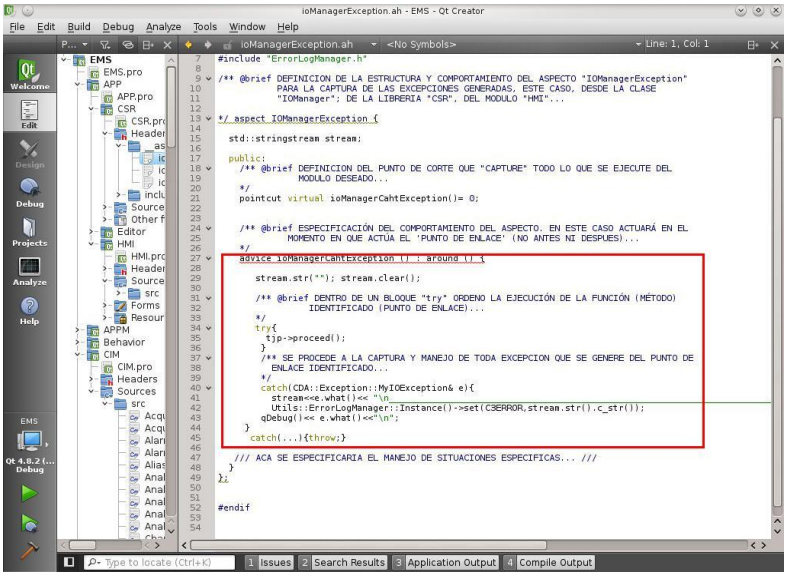
3. Incorporar las cabeceras de aspectos en los proyectos donde serán utilizados.



4. Definir los puntos de corte para las acciones que se aplicarán a cada punto de enlace. Se recuerda que los puntos de corte forman parte esencial del aspecto ya que indican el lugar donde se aplicará el mismo.



5. Definir *advices,* los cuales constituyen la implementación del aspecto. En este ejemplo se estructura el código para capturar las excepciones generadas en los puntos de enlaces.



6. Se realiza el procedimiento anteriormente planteado para la generación de trazas.

La siguiente encuesta fue realizada a 13 de los desarrolladores del módulo HMI. Sus resultados son expuestos posteriormente y aporta a esta investigación elementos significativos que permiten reconocer si la infraestructura propuesta pudiese ser aceptada. Indagándose sobre la POA como una forma de resolver los problemas que se reflejan transversalmente en la aplicación. Igualmente se valoran los conceptos establecidos para que el desarrollador cuente con un manejo declarativo de las excepciones y una internacionalización de las mismas.

La presente encuesta se realiza con el objetivo de obtener un diagnóstico de la situación actual de la gestión de *trazas y excepciones* en el módulo de visualización (HMI) del sistema SISCP-GALBA. Usted como "componente crítico en el desarrollo del mencionado módulo", ha sido seleccionado para colaborar con la investigación "Infraestructura para la configuración de trazas y excepciones en los sistemas SCADA". Un posterior análisis de los resultados mostrará de manera más clara el impacto de la solución propuesta, es decir, conocer desde su punto de vista la necesidad o no de la aplicación del "paradigma" Programación Orientada a Aspecto (AOP) para la gestión de trazas y excepciones en sistemas SCADA. Todo ello como contraposición a los mecanismos y herramientas que usted conoce, con los que desarrolla, y que bien pudieran llegar al mismo resultado.

PREGUNTAS:

1. ¿Conoce algún mecanismo para lograr encapsular las funcionalidades no básicas (generación de trazas, excepciones, seguridad, acceso a bases de datos) de la aplicación?

   Biblioteca Log4j__ Biblioteca Log4cxx__ Contenedores livianos __

   Servidores de aplicaciones y componentes__Programación Orientada a Aspecto__ Otros____

2. ¿Conoce usted sobre el paradigma de Programación Orientado a Aspectos?

   SI__ NO__

3. ¿Tendría usted algún inconveniente en utilizar la POA, conociendo que esta es capaz de solucionar eficientemente los problemas de incumbencias transversales que radican en el enfoque OO?

   SI__ NO__

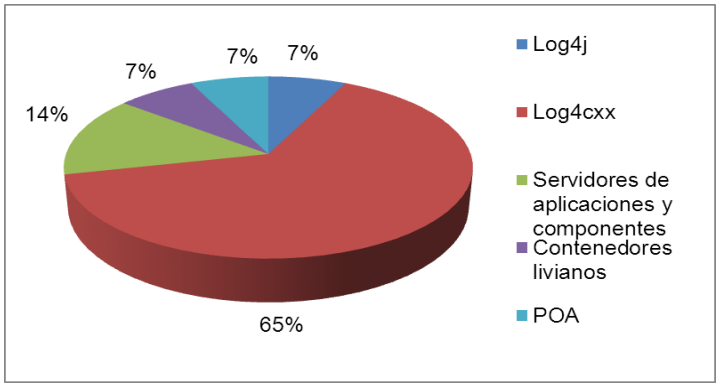
Excepciones:

La elaboración de otros mecanismos que permitan mejorar el manejo de errores que puedan surgir durante la ejecución de la aplicación. Imponen otros criterios que posibiliten una mayor flexibilización de este proceso; dígase a la hora de definir, lanzar y manejar excepciones. De tal manera que el desarrollador pueda especificar propiamente las que sean necesarias, o sea, un manejo declarativo de las mismas. Igualmente ante cada error detectado deberá existir una única respuesta con diferentes opciones de idiomas, este criterio es definido como internacionalización de las excepciones.

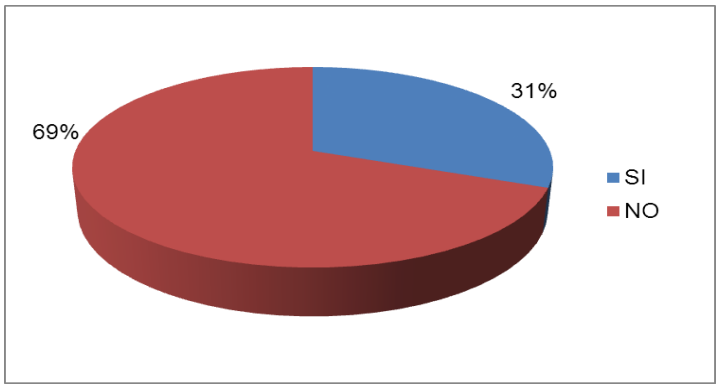
4. ¿Cree usted necesario un manejo de excepciones? SI__ NO__
5. ¿Para usted tendría alguna ventaja contar con una internacionalización de las excepciones y un manejo declarativo de las mismas? SI__ NO__
6. ¿Resulta para usted menos eficiente el tener que declarar previamente las excepciones en un archivo XML, para

luego ser configuradas según su necesidad? SI__ NO__

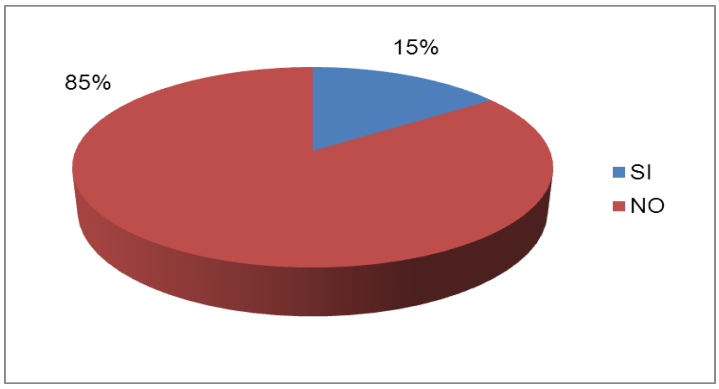
¿Conoce algún mecanismo para lograr encapsular las funcionalidades no básicas (generación de trazas, excepciones, seguridad, acceso a bases de datos) de la aplicación?



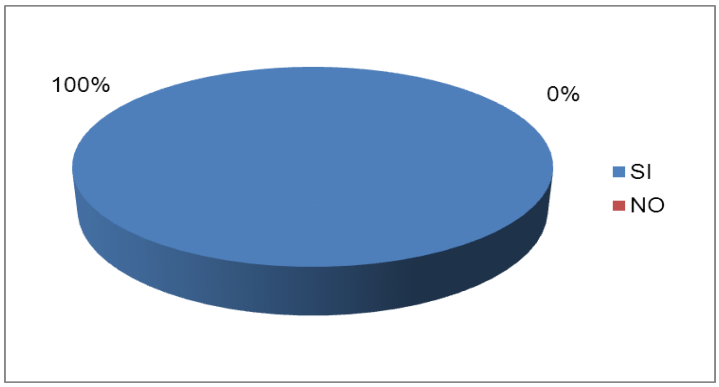
¿Conoce usted sobre el paradigma de Programación Orientado a Aspectos?



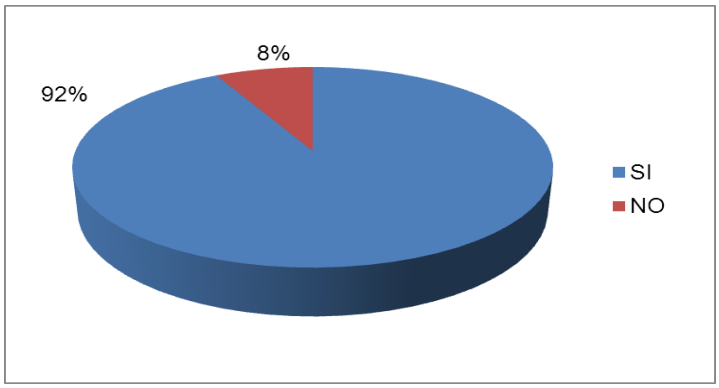
¿Tendría usted algún inconveniente en utilizar la POA, conociendo que esta es capaz de solucionar eficientemente los problemas de incumbencias transversales que radican en el enfoque OO?



¿Cree usted necesario un manejo de excepciones?



¿Para usted tendría alguna ventaja contar con una internacionalización de las excepciones y un manejo declarativo de las mismas?



¿Resulta para usted menos eficiente el tener que declarar previamente las excepciones en un archivo XML, para luego ser configuradas según su necesidad?