Universidad de las Ciencias Informáticas

Facultad 6

Título: Diseño de un intermediario de peticiones de objetos remotos (ORB) para el sistema SURIA Vision.

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

Autora: Sailín Castillo Rodríguez

Tutor: Cesar Santos Sanabria

Co-tutor: Reynier Pupo Gómez

Junio, 2013

Mira que te mando que te esfuerces y seas valiente; no temas ni desmayes, porque Jehová tu Dios estará contigo en dondequiera que vayas.
Josué 1:9

Reprime del llanto tu voz, y de las lágrimas tus ojos; porque salario hay para tu trabajo.
Jeremías 31:16

Agradecimientos

En primer lugar quiero agradecer a mi Dios TODOPODEROSO, Jehová de los Ejércitos, el gran Yo Soy, que me regaló la salvación y que permitió que yo estuviera hoy aquí. Al único y verdadero Dios que me ha sustentado, dirigido y abierto puertas cuando se habían cerrado. Al que nunca me ha abandonado y nunca me abandonará. A mi escudo y fortaleza, a mi Dios Real, mil gracias.

A mis padres: Ricardo Castillo Valdés y Margarita Rodríguez Santiesteban.

Mamita, te doy muchas gracias porque sin ti no hubiera sido posible alcanzar esta meta. Gracias por alentarme cuando me sentí triste o desanimada. Gracias por preocuparte tanto por mí y darme tu amor y dedicación incondicionalmente. Gracias por enseñarme a vivir en humildad y por ser exactamente como eres. Gracias

Papi, te agradezco haberme enseñado que en la vida hay que abrirse camino luchando, que aunque las circunstancias sean adversas, debemos darle la cara a los problemas y ser valientes. Gracias por enseñarme a ser más fuerte cada día y por confiar en mí. Gracias por cada palabra cálida y amorosa en las situaciones difíciles. De todo corazón, muchas gracias por ser mi papito lindo.

Gracias a mis hermanas Saimar y Sairis que han ayudado a la formación de mi carácter con sus formas tan peculiares de ser. Gracias por quererme y apoyarme. Gracias por ser mis hermanitas menores y admirarme tanto.

Gracias a toda mi familia que ha estado apoyándome durante toda la carrera y tantos años de sacrificio lejos de casa. En especial gracias a mis tías y tíos queridos: María y Yoyi, Grissel y Sixto, Aleida y Héctor y Oneida por darme abrigo y sustento. Gracias por su hospitalidad y amor. Gracias por tantas atenciones y su compañía cuando estuve sin Mami y Papi.

Gracias a Migdy, tía Nora y tía Pastora por sus palabras de aliento y sus mimos. A tía Zaida por la oportuna ayuda con el capítulo 1.

Gracias a mis amigos y hermanos en Cristo: Saffit, Reinier y Danneris. Gracias, porque sin ustedes no imagino mi proceso de santificación. Gracias por su amistad sincera y su ayuda

desinteresada. En especial muchas gracias a Reinier por tantas horas dedicadas a la revisión del documento.

Gracias a mis amigas de siempre: Dana, Norkys, Zenia, Yanet, Rosalí y Gretell. Gracias por los buenos y malos tiempos compartidos, gracias por las llamadas y correos. Gracias por ser mi piquete de siempre.

Gracias a los amigos de la UCI que desde primero han sido parte de mi historia, a Aylín, Yadira (que aunque esté lejos la llevo en mi corazón), Emilio, Ripoll y todos los demás. Muchas gracias por dejarme contar con ustedes cuando los necesité.

A todos mis hermanos de la UCI muchas gracias. En especial a Dailén, Yunexy, Yeni y Angel Delvis. Gracias por sus oraciones y su preocupación por el estado de la tesis. A todos muchas gracias.

Gracias a todas las muchachitas del apartamento: Susana, Kirenia, las mellizas y Nathalie. Gracias porque de una forma u otra también han contribuido a la formación de mi carácter.

Gracias a mi prima Ro y a Elizabeth Gutiérrez que en los momentos críticos del desarrollo de esta tesis intervinieron como amigas y orientadoras. Por el tiempo y la ayuda prestada, gracias.

Gracias a mi tutor, César Santos que estuvo al tanto del desarrollo del trabajo, guiándome y apoyándome en todo. Gracias por enseñarme a abrirme camino investigando y preguntando.

Gracias a todos los profesores que contribuyeron al desarrollo del trabajo, tanto evaluando como orientando.

A todos los que se interesaron y me apoyaron, muchas gracias.

DECLARACIÓN DE AUTORÍA

Declaro ser la autora de la presente tesis que tiene por título: “Diseño de un intermediario de peticiones de objetos remotos (ORB) para SURIA Vision” y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Sailín Castillo Rodríguez

_____

Firma del Autor

Cesar Santos Sanabria

_____

Firma del Tutor

DATOS DE CONTACTO

Breve currículo del tutor: Ingeniero en Ciencias Informáticas con 2 años de experiencia en el tema, 2 años de graduado.

Información de contacto: csanabria@uci.cu

RESUMEN

El trabajo de diploma centra su atención en el software de video vigilancia SURIA Vision, un sistema distribuido que actualmente está en fase de reconstrucción en tecnología Qt y lenguaje C++. Específicamente se plantea el diseño del ORB (*Object Request Broker*)o intermediario de peticiones de objetos del sistema SURIA Vision 2.0, ya que en su versión anterior se incluyó la tecnología *.NET Remoting* para garantizar la interoperabilidad entre componentes y para la versión actual se necesita desarrollar un intermediario sin restricciones de licencia ni muchas libertades para la difusión de su código fuente.

Se realizan investigaciones sobre algunas tecnologías que implementan un intermediario de peticiones de objetos en su solución, tales como CORBA, Web Services, XML-RPC y ICE, que sirvieron para concretar la propuesta. A nivel nacional no se encontraron investigaciones relevantes respecto al desarrollo de ORB, siendo el presente trabajo una solución importante para aplicaciones distribuidas con tecnología Qt y lenguaje C++.

Se presenta la propuesta de arquitectura y diseño del ORB a implementar en un futuro, validando si los mismos son eficientes. Se hace uso de varios patrones de diseño que garantizan un eficiente desarrollo del sistema.

El diseño obtenido garantiza una futura implementación dirigida y eficaz, garantizando la interoperabilidad entre componentes del sistema.

Palabras clave: Sistemas distribuidos, video vigilancia, ORB.

ÍNDICE

INTRODUCCIÓN 1
CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 6

1.1 Principales términos asociados a los ORB 6
1.1.1 Sistemas Distribuidos 6
1.1.2 Middleware 8
1.1.3 ORB 10

1.2 Análisis de las soluciones existentes 12
1.2.1 Proyecciones tecnológicas actuales a nivel mundial 12
1.2.1.1 CORBA 12
1.2.1.2 ICE 15
1.2.1.3 XML-RPC 17
1.2.1.4 Web Services 19
1.2.2 Proyecciones tecnológicas a nivel nacional 21

1.3 Descripción actual del dominio del problema 23
1.3.1 Descripción de la situación problemática 24

1.4 Caracterización de la metodología de desarrollo de software 25
1.4.1 RUP 25

1.5 Tecnologías, herramientas y lenguajes para eldesarrollo del sistema 25
1.5.1UML 26
1.5.2 Herramienta CASE 26
1.5.3 Lenguaje de programación C++ 27
1.5.4 Framework Qt 27

Conclusiones 27

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA 29

2.1 Modelo de Dominio 29
2.1.1 Descripción de los conceptos principales 29
2.1.2 Diagrama del Modelo de Dominio 29

2.2 Especificación de requisitos de software 30
2.2.1 Requisitos funcionales 30
2.2.2 Requisitos no funcionales 32

2.3 Modelo de Sistema. Definición de casos de uso 32
2.3.1 Definición de actores 32
2.3.2 Listado de casos de uso 33
2.3.3 Diagrama de Casos de Uso del Sistema 34

2.3.4 Casos de uso expandidos 34
Conclusiones 39

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA 40

3.1 Análisis de la arquitectura de software. 40
3.1.1 Arquitectura Cliente-Servidor para el ORB 40
3.1.2 Arquitectura por capas para el ORB 41
3.1.3 Arquitectura Cliente-Servidor por Capas para el ORB 43
3.2 Flujos de trabajo de Análisis y Diseño. 43
3.3 Modelo de Análisis. 44
3.3.1 Diagramas de Clases del Análisis. 44
3.3.2 Diagramas de Comunicación. 45
3.4 Modelo de Diseño. 46
3.4.1 Patrones de Diseño/Arquitectura. 46
3.4.2 Fundamentación del empleo de patrones. 46
3.4.3 Patrones. 47
3.4.4 Diagrama de Clases del Diseño 48
3.4.5 Diagramas de Secuencia. 50
3.5 Validación de la solución propuesta 51
3.6 Método de validación de los resultados 52
3.6.1 Técnica de validación usando matriz de trazabilidad 52
3.6.2 Métricas de la calidad de especificación de requisitos (Ausencia de ambigüedad) 53
3.6.3 Métricas para la validación del diseño 54
Conclusiones 59

CONCLUSIONES GENERALES 61
RECOMENDACIONES 62
REFERENCIAS BIBLIOGRÁFICAS 63
BIBLIOGRAFÍA 68
ANEXO 1 ..... ¡Error! Marcador no definido.

ÍNDICE DE FIGURAS

Figura 1 Un sistema distribuido organizado como middleware (Tanenbaum, y otros, 2007) 7
Figura 2 Funcionamiento de un ORB (Rouaux, 2005).....11
Figura 3 Ejemplo de código XML-RPC.....18
Figura 4 Primer ejemplo de XML (Cooper, 2010).....19
Figura 5 Segundo ejemplo XML (Cooper, 2010) 19
Figura 6 Modelo del Dominio 30
Figura 7 Diagrama de Casos de Uso del Sistema.....34
Figura 8 Capas del ORB (Volter, y otros, 2005) 42
Figura 9 Interacción entre patrones del ORB (Volter, y otros, 2005) 43
Figura 10 Diagrama de Clases del Análisis CU Invocar método de objeto remoto 45
Figura 11 Diagrama de Colaboración CU Invocar método de objeto remoto.....45
Figura 12 Diagrama de Clases del Diseño General 49
Figura 13 Diagrama de Secuencia CU Invocar método de objeto remoto (Del lado del Cliente) 51
Figura 14 Diagrama de Secuencia CU Invocar método de objeto remoto (Del lado del Servidor).....51

ÍNDICE DE TABLAS

Tabla 1 Actor del sistema.....33
Tabla 2 CU Invocar método de objeto remoto.....34
Tabla 3 CU Obtener objeto remoto 34
Tabla 4 Descripción del CU Invocar método de objeto remoto.....37
Tabla 5 Descripción del CU Obtener objeto remoto 39
Tabla 6 Clasificación de Clases 55
Tabla 7 Criterios de evaluación de la métrica TOC 56
Tabla 8 Tamaño Operacional general del diseño.....56
Tabla 9 Resultado de la aplicación de la métrica TOC.....57
Tabla 10 Criterios de evaluación de la métrica RC.....58
Tabla 11 Criterios y categorías obtenidos en la aplicación de la métrica.....59
Tabla 12 Resultados obtenidos de la aplicación de la métrica RC 59

INTRODUCCIÓN

En la actualidad existen gran variedad de servicios que garantizan una mejor calidad de vida para los seres humanos y que a su vez, demuestran el progreso alcanzado por las Tecnologías de la Información y las Comunicaciones (TIC). Desde la radio, la televisión y la telefonía convencional hasta la Internet, la televisión de alta definición y la telefonía móvil, es apreciable como las TIC constituyen un área que se encuentra en constante cambio y desarrollo. Este proceso de perfeccionamiento ha sido posible por la aparición de la tecnología digital, que ha permitido desplegar a niveles superiores el manejo de la información y el conocimiento. **(Cobo Romaní, 2009)**

En la década del 80 con el surgimiento de la computadora personal también surgen las redes de computadoras que ejecutan sistemas distribuidos. Un sistema distribuido, como lo define Tanenbaum, se presenta a sus usuarios como un sistema tradicional con un solo procesador, cuando en realidad está compuesto por varios procesadores (de sus respectivas computadoras host), o sea, es un sistema que contiene varios procesos desarrollándose en varias máquinas a la vez, pero posee la capacidad de parecer un sistema único ante sus usuarios. Los mismos no conocen dónde se están ejecutando sus programas ni dónde se encuentran sus archivos porque el sistema operativo distribuido se encarga de esto de manera automática y eficiente. **(Tanenbaum, 2009)**

Un ejemplo claro de aplicaciones que hoy se desarrollan con sistemas distribuidos son los cajeros automáticos, ya sea por su necesidad de acceso a diferentes aplicaciones dentro del sistema o porque pueden estar ubicados geográficamente en lugares diferentes. Asimismo los sistemas de video vigilancia también constituyen aplicaciones que se desarrollan con el paradigma de los sistemas distribuidos. Para que los mismos sean funcionales es necesaria la implementación de un software que se encargue de manejar las complejidades de comunicación de bajo nivel y del sistema operativo, generando interfaces que faciliten la creación de sistemas distribuidos sobre plataformas distribuidas **(Qusay H. Mahmoud, 2004)**. El middleware o intermediario de peticiones es el software designado para ejecutar esta función y se encuentra ubicado entre el nivel de aplicación o interfaz de usuario y los niveles de transporte o inferiores, teniendo en cuenta la estructura por capas para la arquitectura Protocolo de Control de Transmisión/Protocolo de Internet (TCP/IP, por sus siglas en inglés).

Entre las soluciones para la interoperabilidad que hoy en día contribuyen al funcionamiento de sistemas distribuidos en el mundo, se encuentran: CORBA (*Common Object Request Broker Architecture*), ICE (*Internet Communication Engine*), XML-RPC (*XML-Remote Procedure Call*), *Web Services* y .*Net*

Remoting. Estas soluciones tienen implementaciones para varios lenguajes de programación tales como Java, C++, Python, PHP y ofrecen funcionalidades para establecer la comunicación entre las aplicaciones de un sistema distribuido mediante objetos remotos.

En Cuba también se han diseñado aplicaciones distribuidas que implementan intermediarios únicos, o sea, software para garantizar la comunicación entre componentes de una aplicación. Tal es el caso del software de video vigilancia Xyma Safe Vision desarrollado en la empresa Desarrollo de Aplicaciones, Tecnologías y Sistemas (DATYS), que usa el lenguaje C# 2.0 en su implementación y Visual Studio 2010.

En el Complejo de Investigaciones Tecnológicas Integradas (CITI) que radica en el Instituto Superior Politécnico José Antonio Echeverría (ISPJAE o CUJAE) también se está trabajando en el área de la integración de aplicaciones, más específicamente en la integración de información para ambientes empresariales, usando el mecanismo de *Message-oriented Middleware* (MOM) en implementaciones de ORACLE.

El centro de Geoinformática y Señales Digitales (GEYSED) de la Universidad de las Ciencias Informáticas (UCI) también ha tomado en cuenta el uso de esta tecnología de avanzada y hoy desarrolla un sistema de video vigilancia que tiene la finalidad de fortalecer la protección en cualquier lugar donde sea desplegado. Su nombre es SURIA Vision y aplica el paradigma de sistemas distribuidos e intermediarios para su implementación.

Por el interés que tienen para la economía del país los proyectos de la UCI, dada su capacidad de comercialización, adquieren especial relevancia la investigación y perfeccionamiento de los sistemas que se desarrollan, siendo este el caso del proyecto SURIA Vision, en el cual se centra este trabajo.

La ejecución de este proyecto incluye la infraestructura tecnológica para soportar un conjunto de cámaras de seguridad, que se puedan gestionar dentro de una red de datos asegurando la visualización, almacenamiento y trasmisión de los flujos de videos generados en cada uno de los dispositivos de adquisición. Por otra parte, se incluye una solución de software que permite la adquisición de los flujos de videos desde los dispositivos, la visualización de los mismos de manera concurrente en una sala de operaciones, además de su almacenamiento y recuperación en un motor de reglas configurables incluido en la solución. **(Redmine, 2011)**

En la primera versión de SURIA se incluyó el middleware *.NET Remoting* para garantizar la comunicación entre componentes mediante objetos remotos. Pero la implementación del intermediario de .NET es propiedad de la compañía Microsoft, por tal motivo había que adquirir su licencia cada vez que la misma

caducara si se deseaba mantener funcionando el software SURIA que lo usaba como intermediario. Para la comercialización con países interesados también habría que tener la licencia actualizada. Sin embargo, para una nación como Cuba obtener la licencia de cualquier software se hace complicado debido al bloqueo económico de los EEUU, que obliga al país a realizar los trámites de compra por medio de terceros países, triplicando el precio de los productos.

Tal motivo ha impulsado una nueva dirección del proyecto y los desarrolladores de SURIA se han planteado reconstruir el software basado en tecnologías libres de licencia. A partir del análisis de aplicaciones desarrolladas nacional e internacionalmente que implementan técnicas similares, y su rechazo porque aplican tecnologías propietarias en su solución, se determinó seleccionar el framework Qt y el lenguaje de programación C++. La elección de estas tecnologías se basa en que el entorno de desarrollo Qt admite, entre otras, la licencia GNU LGPL, la cual permite el desarrollo de software para comercializar. El lenguaje C++, además de ser un potente lenguaje de programación que permite la creación de interfaces bien definidas, es nativo de Qt, y por tanto, el designado para el diseño y futura implementación del intermediario.

En el proceso de desarrollo de la nueva versión de SURIA resulta necesario reconstruir todos los módulos del sistema y el intermediario que garantizará la interoperabilidad en el sistema es un componente clave a desarrollar. La construcción del intermediario requiere de una documentación ingenieril que guíe su desarrollo, pero actualmente en el proyecto no se cuenta con esta documentación. Además se hace difícil desarrollar este software, puesto que los desarrolladores no poseen gran experiencia en la implementación de este tipo de aplicaciones.

Por otra parte, las propuestas de implementaciones de tecnologías como CORBA y XML-RPC para el *framework* Qt, se consideran deficientes por su licencia, que obliga a la difusión del código fuente del programa al que estén adjuntas y además por la complejidad de su implementación. Tampoco se asumieron las aplicaciones desarrolladas en el país sustentadas en técnicas similares, porque aplican tecnologías propietarias a sus soluciones.

Partiendo de la anterior **situación problemática**, queda identificado el siguiente **problema a resolver**: ¿Cómo contribuir a la interoperabilidad del sistema SURIA Vision a través de tecnologías libres?

En aras de resolver el problema planteado se decide: Diseñar un ORB bajo tecnologías sin licencias propietarias que viabilice el funcionamiento del sistema SURIA y su comercialización, lo cual figura como el **objetivo general** de la investigación.

Con la finalidad de cumplir de manera detallada el objetivo general planteado, se definen los siguientes **objetivos específicos**:

- Caracterizar las tecnologías existentes que implementan la interoperabilidad entre componentes remotos con el framework Qt y en lenguaje C++.
- Diseñar un intermediario que cumpla con las necesidades de SURIA Vision para el manejo de tipos de datos simples (*string, integer, double, char*).
- Generar los artefactos necesarios que determinan el proceso de desarrollo del ORB para su futura implementación.
- Validar el diseño creado.

El **objeto de estudio** en el cual se centra la investigación es: Los intermediarios de peticiones de objetos remotos (ORB) desarrollados en tecnología Qt y lenguaje C++, enmarcados en el **campo de acción**: Interoperabilidad entre los componentes remotos de SURIA Vision bajo tecnología Qt y lenguaje C++.

Como **idea a defender** del trabajo de diploma se plantea: Al diseñar un ORB bajo tecnologías libres se facilitará el funcionamiento e interoperabilidad de la nueva versión del sistema SURIA Vision así como su comercialización.

El desarrollo investigativo del trabajo se basó en los **métodos teóricos**: **análisis histórico-lógico** y **analítico-sintético**, y los **métodos empíricos**: **observación** y **entrevista**. El análisis histórico-lógico sirvió para estudiar de forma analítica la trayectoria histórica real de los fenómenos relacionados a los *middlewares*; su evolución y desarrollo. El método analítico-sintético permitió buscar la definición de los conceptos asociados a la investigación (sistemas distribuidos, *middleware, ORB* y las proyecciones tecnológicas actuales utilizadas tales como: CORBA, ICE, XML-RPC y los Servicios Web), orientados a los rasgos que los caracterizan y los distinguen. La observación se utilizó en la realización de valoraciones y obtención de información sobre el funcionamiento de sistemas similares al que se desarrolla, para tener una idea de cómo debe ser el propio. Este método se utilizó en toda la investigación, pues se recogió información de todos los aspectos tratados en la tesis desde el punto de vista de otros autores, así como sus definiciones y resultados. La entrevista como método empírico fue utilizada debido a la necesidad de la autora de conocer sistemas cubanos con un patrón de funcionamiento similar al que se diseñó en el trabajo de diploma, permitiendo adquirir información relevante para la fundamentación teórica de la tesis.

El trabajo de diploma cuenta con un total de 3 capítulos que se describen como sigue:

Capítulo 1: Fundamentación teórica: Incluyó un estado del arte del tema de los sistemas distribuidos y *middlewares*, a nivel internacional, nacional, enfatizando en la UCI. Asimismo se hizo referencia a las tecnologías usadas en la actualidad para el desarrollo de este tipo de sistemas y las metodologías y tecnologías en las que se apoya la autora para la solución del problema que se enfrenta.

Capítulo 2: Características del Sistema: En este capítulo quedaron definidas las características del sistema a diseñar, los requisitos funcionales y no funcionales del mismo, los casos de uso del sistema, así como los diagramas que los modelan, para un mejor entendimiento del sistema a desarrollar.

Capítulo 3: Arquitectura, Análisis y Diseño del Sistema: En este capítulo quedó conformada la arquitectura del sistema, así como los resultados del proceso de Análisis y Diseño, y los diagramas elaborados durante el desarrollo de este flujo. También se realizó la validación del diseño elaborado.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Este capítulo tiene como objetivo hacer una introducción al tema de la investigación, enfatizando principalmente en las bases técnicas necesarias para el entendimiento del funcionamiento del software a desarrollar, siendo de vital importancia destacar los intermediarios de peticiones de objetos, que son la esencia de los sistemas distribuidos. Así como una explicación de los términos sistema distribuido y *middleware*, como capa fundamental del proceso de comunicación entre aplicaciones. También se incluye una panorámica de las proyecciones tecnológicas actuales que implementan un ORB para la comunicación interna. Además se dará una fundamentación de las metodologías, herramientas y tecnologías a usar en el diseño del *software*.

1.1 Principales términos asociados a los ORB

Antes de adentrarse en el tema de los ORB, es preciso conocer los principales términos asociados a los intermediarios de peticiones de objetos remotos, para lograr un mejor entendimiento de su funcionamiento y aplicación.

1.1.1 Sistemas Distribuidos

Una de las definiciones citadas en el libro de Tanenbaum “Sistemas Operativos Modernos, Segunda parte”, destaca que un sistema distribuido es“… aquel que se ejecuta en una colección de máquina sin memoria compartida, pero que aparece ante sus usuarios como una sola computadora”. **(Tanenbaum, 2005)**

En el mismo libro hace referencia a otros autores planteando que un sistema distribuido “es aquel que se ejecuta en una colección de máquinas enlazadas mediante una red pero que actúan como un uniprocesador virtual”. **(Tanenbaum, 2005)**

Su más reciente libro publicado sobre el tema, *Distributed System. Principles and Paradigms*, proporciona una definición más rebuscada y que será la usada para el desarrollo del presente trabajo. Un sistema distribuido es “… una colección de computadoras independientes que aparecen a sus usuarios como un único sistema coherente” **(Tanenbaum, y otros, 2007)**. Es decir, que no se debe ver al sistema compuesto por elemento heterogéneos (aunque así sea), cuyo protocolo de acceso depende del sistema operativo de cada ordenador, de su estructura de ficheros o de los comandos que en cada ordenador se utilicen para acceder a sus componentes, sino como una serie derecursos a los que se accede como si

fueran locales. El objetivo principal de los sistemas distribuidos es el compartimiento fácil y eficiente de los recursos entre múltiples usuarios.

Para entender mejor la definición antes citada sería conveniente concentrarse en las caracteríticas de los sistemas distribuidos. Según **(Tanenbaum, y otros, 2007)**, los sistemas distribuidos se caracterizan por: ocultar la presencia de computadoras y formas de comunicarse al usuario. Esta importante característica permite que las diferencias entre las diferentes computadoras y las formas en que se comunican unas con otras estén, en su mayoría, ocultas al usuario, lo que sostiene la organización interna del sistema distribuido. Otra carácterística es: poseer un mecanismo consistente y uniforme de comunicación. Aquí se evidencia que usuarios y aplicaciones pueden interactuar con un sistema distribuido de una forma consistente y uniforme, independientemente de dónde y cuándo tenga lugar la interacción. Deben además ser relativamente fáciles de expandir y escalar, ya que esta característica es una consecuencia directa de tener computadoras independientes.

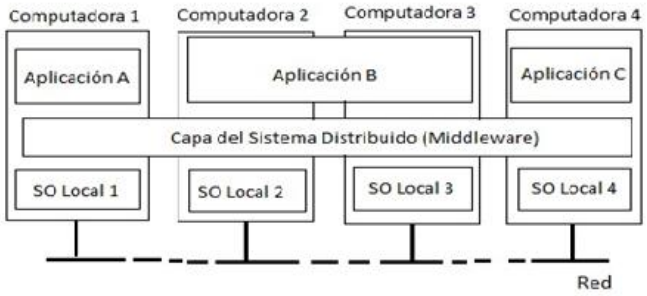

Figura 1 Un sistema distribuido organizado como middleware (Tanenbaum, y otros, 2007)

La Figura 1 muestra cuatro computadoras conectadas en red y tres aplicaciones, de las cuales la aplicación B es distribuida a través de las computadoras 2 y 3,en cada aplicación esofrecida la misma interfaz. El sistema distribuido provee los medios para la comunicaciónde una única aplicación distribuida con cada una de las otras, pero además, los medios para dejar que las diferentes aplicaciones internas se comuniquen. Al mismo tiempo oculta, lo mejor y de la forma más razonable, las diferencias en el hardware y el sistema operativo de cada aplicación. **(Tanenbaum, y otros, 2007)**

Los sistemas distribuidos tienen un buen número de puntos a su favor. Pueden ofrecer una buena proporción precio/desempeño, ser altamente confiables y aumentar su tamaño de manera gradual al aumentar la carga de trabajo. Además el uso de un sistema distribuido garantiza un gran poder de

cómputo al usarse varios microprocesadores. Permiten la compartición de datos y dispositivos a través de la red y garantizan la flexibilidad al difundir la carga de trabajo entre las máquinas disponibles.

Sin embargo, el área de los sistemas distribuidos aún es un asunto en desarrollo por lo que posee varias **desventajas** según **(Tanenbaum, 2005)**. La red se puede saturar o causar otros problemas y una vez que el sistema llega a depender de la red, la pérdida o saturación de la misma, puede negar algunas de las ventajas que el sistema distribuido debía conseguir. Otra desventaja es la seguridad, ya que si se obtiene acceso a la red (de forma ilegal) se consigue un acceso sencillo a una gran cantidad de información, lo cual requiere establecer unos mecanismos de acceso muy seguros.

Aplicaciones de los sistemas distribuidos

Muchas son las aplicaciones de los sistemas distribuidos, las cuales varían desde la provisión de eficiencia de programación para redes de telecomunicación, de capacidad de cómputo a grupos de usuarios, hasta sistemas bancarios, comerciales y aplicaciones multimedia.

El presente trabajo centra su atención en el sistema de video vigilancia SURIA Vision, el cual es un sistema distribuido, ya que posee varios módulos o aplicaciones que necesitan colaborar entre sí, para que los usuarios finales sean capaces de distinguir el sistema como un todo. Además debe ser capaz de brindar facilidades de ejecución tales como almacenamiento y accesibilidad constante a un gran monto de video generado por las cámaras que están en funcionamiento todo el tiempo. También es preciso que el sistema permita añadir nuevas cámaras al sistema a medida que se haga necesario, que sea capaz de ejecutar las operaciones a gran velocidad y que distribuya la carga de trabajo entre los diferentes módulos que están interconectados entre sí. Estas son características propias de los sistemas distribuidos que catalogan al sistema SURIA Vision como tal.

1.1.2 Middleware

Para hablar de *middleware* es importante destacar que surge con el papel esencial de manejar la complejidad y heterogeneidad de los sistemas distribuidos. El *middleware* ofrece abstracciones de programación que ocultan algunas de las complejidades de la capa de red y del sistema operativo. **(Oberle, 2006)**

Un *middleware* es un *software* situado entre el nivel de aplicación (interfaz de usuario) y los niveles de transporte e inferiores que suele presentarse como el interfaz o *Application Program Interface,* Interfaz de Aplicación de Programa *(*API) de los distintos sistemas operativos. El *middleware* permite la adecuación de la aplicación a un lenguaje general entendible por las aplicaciones que comparten ese lenguaje. **(García Sánchez, y otros, 2005)**

Según **(Calvo Gordillo, 2007)** *middleware* no es más que una designación genérica para referirse al *software* de conectividad situado entre la capa de aplicación y el sistema operativo, que proporciona una *API* de alto nivel con servicios que facilita la creación de aplicaciones distribuidas sobre plataformas heterogéneas. El mismo tiene como objetivos: abstraer las aplicaciones distribuidas de redes de comunicación subyacentes (complejas y heterogéneas), de sistemas operativos y de lenguajes de programación; simplificar la creación de aplicaciones distribuidas complejas; y reducir el tiempo y esfuerzo utilizado en el desarrollo de aplicaciones distribuidas.

En pocas palabras, *middleware* es el *software* que permite el acceso de una manera uniforme a los recursos del sistema en todas las plataformas, usando interfaces estándares de programación y protocolos que se sitúan entre la aplicación y el *software* de comunicaciones y sistema operativo.

Existe una gran variedad de *middleware,* desde los muy simples hasta los complejos. Todos ellos tienen en común la facilidad de ocultar las complejidades y diferencias de los disímiles protocolos de red y sistemas operativos. **(Stallings, 1997)** Aunque existe una amplia gama de *middleware*, estos se pueden clasificar como sigue:

- **Remote Procedure Call (RPC)**: El cliente realiza una llamada a procedimientos que están corriendo en máquinas remotas.
- **Publish/Subscribe**: Estos tipos de monitores *middleware* activan y entregan información relevante para los suscriptores.
- **Message Oriented Middleware (MOM)**: Los mensajes enviados al cliente se recogen y se almacenan hasta que son solicitados, mientras el cliente continúa con otros procesos.
- **Object Request Broker (ORB)**: Este tipo de *middleware* permite que los clientes envíen objetos y soliciten servicios en un sistema orientado a objetos.

A continuación se muestra una breve descripción de ellos.

RPC (Llamada a Procedimiento Remoto)

Es un protocolo que permite que programas de máquinas diferentes interactúen mediante la semántica de los procedimientos de llamada/retorno, como si los dos programas estuvieran en la misma máquina. Es decir, un programa puede ejecutar código en una máquina remota sin preocuparse por las comunicaciones entre ambos. **(Qusay H. Mahmoud, 2004)**

Es muy usado dentro del paradigma cliente/servidor, donde el cliente inicia el proceso solicitando al servidor que ejecute algún procedimiento, el cual devuelve el resultado de la operación al cliente. El objetivo es que la sintaxis del cliente y el servidor permanezcan igual, como si estuvieran en la misma máquina.

Publish/Subscribe (Publicador/Subscriptor)

Tiene como objetivo mantener desacoplado al publicador de un mensaje con su subscriptor. El destino de un mensaje es un asunto o tópico, no un consumidor en particular. Por lo que este mecanismo se usa cuando un sistema necesita mandar un mensaje a todos los sistemas interesados en recibirlo. Los subscriptores pueden estar subscritos a uno o más tópicos. **(Qusay H. Mahmoud, 2004)**

En resumen, este *middleware* activa y entrega información relevante a los subscriptores y es muy útil cuando un grupo de aplicaciones desean notificar a las otras aplicaciones de una ocurrencia en particular.

MOM (Middleware Orientado a Mensajes)

Es un *software* que reside en ambos lados de la arquitectura Cliente/Servidor y generalmente apoya las llamadas asíncronas entre el cliente y el servidor de aplicaciones. Las colas de mensajes que se generan, proveen almacenamiento temporal cuando el programa destino está ocupado o desconectado. Provee mecanismos para crear, manipular, almacenar y comunicar los mensajes, que son una combinación de datos e información de control. **(Qusay H. Mahmoud, 2004)**

Este tipo de *middleware* es muy apropiado para la actualización de información y la sincronización y coordinación de procesos.

En la actualidad, los *middleware* y los servicios que incorporan, hacen que su uso sea fácil de aprender y rápido de desarrollar, implementando aplicaciones para todo tipo de entornos. Se encuentran en continua evolución y mejora lo cual provoca que se extienda su rango de influencia.

1.1.3 ORB

Un ORB (*Object Request Broker*) es un tipo de *middleware* que recibe peticiones de un proceso que está en una máquina y le permite realizar una llamada a un método de un objeto que se encuentra en una máquina remota. **(Garrido Fuentes, 2011)**

En términos más simples, un ORB crea una conexión cliente/servidor entre dos objetos a través de una red. Su objetivo es permitir a las aplicaciones, que pueden estar corriendo en sistemas operativos completamente diferentes, intercambiar o actuar sobre la información. **(NETWORK WORLD, 2006)**

Ni los objetos, los programadores que los crean, ni los usuarios finales que los usan necesitan conocer nada sobre los otros objetos en la red porque el ORB es diseñado para manejar las interacciones, intercambiarlas y actuar sobre las mismas. Específicamente, los ORB:

1. Definen las interfaces de una aplicación a fin de que otras aplicaciones las puedan usar.
2. Descubren aplicaciones y sus interfaces asociadas en algún otro sitio de la red.
3. Permiten a las aplicaciones enviar y responder mensajes desde unas hacia otras.

El ORB es el encargado de realizar el llamado *marshalling*, es decir, el proceso de transformar la representación en memoria de un objeto en un formato de datos que sea adecuado para la transmisión. **(Garrido Fuentes, 2011)**

También se puede definir como un mecanismo de comunicaciones usado en un ambiente de computación distribuida orientado a objetos en el cual los módulos de programas pueden escribirse en cualquier lenguaje de programación y todavía pueden proveer servicios a otras aplicaciones.En un esquema podría representarse de la forma:

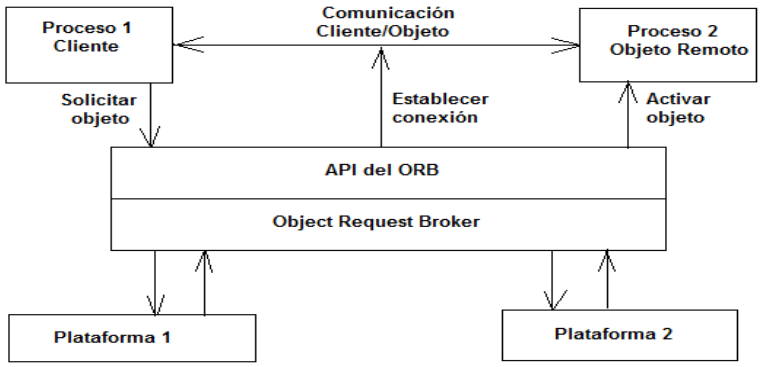

Figura 2 Funcionamiento de un ORB (Rouaux, 2005)

En la Figura 2 un cliente desea invocar un servicio ofrecido por algún objeto. Entonces, solicita al ORB que ubique el objeto que provee ese servicio. Cuando el ORB lo encuentra, prepara la implementación para dar servicio a la solicitud en cuestión. Finalmente, el ORB instruye al cliente y al servidor para que se conecten y empiecen a dialogar. Todo este proceso tiene lugar internamente en el ORB y resulta completamente transparente para el desarrollador. Para él, simplemente se está llamando a un método de un objeto. **(Rouaux, 2005)**

El lugar físico del objeto que provee la respuesta carece de importancia, y para el usuario, la aplicación da la apariencia de estar libre de irregularidades, aunque los servicios diversos pueden estar viniendo de varias partes diferentes de la red **(base, 2012)**. Básicamente un ORB, permite o facilita la comunicación entre objetos.

Proporciona facilidades tales como la localización de un objeto remoto dada una referencia al mismo y el ordenamiento de los parámetros y valores de retorno e invocación a métodos remotos. Implementa el mecanismo de direccionamiento de objetos, o sea, dan aspectos de objetos a los procesos conectados a él.

1.2 Análisis de las soluciones existentes

Para obtener una idea general de las implementaciones de los ORB, se hace necesario el análisis de algunas soluciones existentes en el mundo y en Cuba que los implementan. Por ello, el estudio de las proyecciones tecnológicas de actualidad será muy efectivo en el desarrollo del software a diseñar.

1.2.1 Proyecciones tecnológicas actuales a nivel mundial

Hoy en día muchas empresas reconocidas a nivel mundial se han encargado de proponer soluciones para una mejor comunicación entre los componentes de aplicaciones remotas. Cuatro de ellas han sido motivo de estudio para el desarrollo de este trabajo por las facilidades de ejecución que proveen. Las mismas son: *CORBA, XML-RPC, ICE y* los *Web Services* las cuales serán expuestas a continuación.

1.2.1.1 CORBA

CORBA (*Common Object Request Broker Architecture*, Arquitectura común de intermediarios de peticiones a objetos)es un sistema muy conocido que se basa en objetos en tiempo de ejecución. Es un sistema cliente/servidor, en el que los procesos cliente en máquinas cliente pueden invocar operaciones

en objetos ubicados en máquinas servidores (posiblemente remotos). Aplica en su solución el método de llamada a procedimientos remotos.

Es una especificación definida por el OMG (*Object Management Group*) para la creación y uso de objetos remotos, cuyo objetivo es proporcionar interoperabilidad entre aplicaciones en un entorno distribuido y heterogéneo. Es conocido como un tipo de *middleware*, ya que no efectúa las funciones de bajo nivel necesarias para ser considerado un sistema operativo. A pesar de que debe funcionar sobre sistemas operativos tradicionales, efectúa muchas de las operaciones que tradicionalmente se han considerado del dominio de los sistemas operativos para entornos distribuidos. **(Drake, 2008)**

CORBA es una arquitectura de comunicaciones entre sistemas heterogéneos que soporta construcción e integración de tecnologías de diferente fabricante. Puede agrupar antiguas y nuevas aplicaciones de software. Está basada en un gestor de peticiones a objetos comunes y permite interoperabilidad entre aplicaciones en máquinas remotas en un entorno distribuido. Es una plataforma que tiene funcionalidad de sistema abierto y que requiere para cada lenguaje soportado una interfaz estandarizada entre CORBA y la herramienta de programación. **(Drake, 2008)**

Para que sea posible que un cliente en una plataforma determinada invoque a un servidor en una plataforma distinta, CORBA implementa un ORB entre el cliente y el servidor para permitir que sus especificaciones coincidan. Cada objeto CORBA se describe mediante una definición de interfaz en un lenguaje conocido como IDL (*Interfaz Definition Language*, Lenguaje de Definición de Interfaz), el cual indica qué métodos debe exportar el objeto y qué tipos de parámetros espera cada método. Al crearse un objeto CORBA también se crea una referencia a este objeto y se devuelve al proceso que lo creó. De esta forma el proceso identificará al objeto para las subsiguientes invocaciones de sus métodos.

Se analizaron algunas implementaciones de CORBA para el lenguaje C++, las cuales se resumen a continuación.

MICO

Según **(MICO: An open source CORBA implementation, 2004)**, MICO es una implementación completa de CORBA 2.2 escrita en C++. Usa licencia libre. Ha sido diseñada para propósitos educacionales y todo su código fuente se encuentra disponible bajo notificaciones de derechos de autor del proyecto GNU. Ofrece una interfaz para insertar y extraer tipos de datos construidos que no eran conocidos en el tiempo de compilación **(Kwiatkowski, et al., 2001)**. Al ser esta especificación una implementación completa de CORBA, su grado de dificultad es bien elevado, así como el esfuerzo de los desarrolladores para aplicarlo

y entenderlo. Por tal motivo en el proyecto SURIA esta tecnología no es viable por la ausencia de programadores experimentados en el uso de *middlewares.*

ORBit

ORBit es una implementación de CORBA la cual es usada en el proyecto GNOME. Está distribuida bajo licencia GPL (licencia libre que obliga a la difusión del código fuente del programa para compartirlo con la comunidad de *software* libre) **(Birney, et al., 2003)**. Tiene implementaciones para C, C++, Perl, Python, Ada. Permite la comunicación con otros ORB que cumplan el estándar CORBA sin problemas. No obstante, su licencia no se adecúa a las necesidades existentes en SURIA.

ACE+TAO

ACE+TAO es el nombre que recibe el ORB creado en el *framework* ACE para desarrollar TAO, un estándar basado en CORBA. Representa una de las implementaciones de CORBA en tiempo real con mejores prestaciones. Permite la creación de servidores multihilo y tiene servicio para tiempo real. Fue creado por Douglas Schmidt y su equipo de la Universidad de Washington **(Schmidt, 2006)**. Está bajo licencia de código abierto, aunque se puede usar para crear *software* con propósitos comerciales sin tener la obligación de redistribuir el código. Sin embargo, aprender esta especificación para aplicarla en SURIA, requeriría de desarrolladores con experiencia en el uso de esta clase de software, con los cuales no se cuenta en el proyecto.

Ninguna de estas especificaciones se ajusta a la necesidad actual de SURIA, por lo tanto son excluidas de la solución.

Desventajas de CORBA

Aunque CORBA es una tecnología muy potente que se utiliza en diversas áreas de desarrollo como la robótica y la aeronáutica, sin embargo tiene algunas desventajas para su uso en un proyecto como SURIA Vision. Algunas de las mismas se describen a continuación.

Según **(Tanenbaum, 2009)** un problema grave con CORBA es que cada objeto se encuentra solo en un servidor, lo cual significa que el rendimiento será terrible para los objetos que se utilizan con mucha frecuencia en máquinas cliente por todo el mundo.

CORBA busca la estandarización y está abierto a cualquier empresa que desee implementarlo, pero es tan complejo que casi ningún fabricante ha sido capaz de implementar de forma completa la especificación. El resultado ha sido que CORBA ha entrado en declive y se han buscado nuevas

soluciones tales como ICE y *Web Services* **(Drake, 2008).** Un buen ejemplo de cómo el núcleo de CORBA ha llegado a ser extremadamente complicado con el tiempo es el POA (*Portable Object Adapter*, Adaptador de Objetos Portátiles), el cual es un objeto que solo es visible por el servidor. Este requiere el conocimiento de un experto para su correcto uso.

Muchos de los mapeos de lenguaje parecen "antinaturales", guiados al código específico de CORBA que no interactúa fácilmente con bibliotecas estándar o componentes de terceros. A varios programadores les cuesta mucho trabajo aprender y entender las reglas del mapeo.

A menudo, la única manera de llegar a un acuerdo durante el proceso de presentación de una especificación de CORBA es tomar la gran unión de los conjuntos de características de todas las implementaciones propietarias pre-existentes y, de alguna, manera calzarlos en un estándar. Esto da lugar a especificaciones más grandes y complejas de lo necesario, obteniéndose una plataforma mucho más voluptuosa y lenta con una compleja API resultante que se vuelve más difícil de usar. **(ZeroC, 2012)**

CORBA es una tecnología antigua, que ha ido decayendo por el hecho de no tapar todas las necesidades actuales, o porque para cumplir con esas necesidades la aplicación adquiere una complejidad muy alta, que en muchos casos deja de ser rentable mantener, sobre todo si se tiene en cuenta que se utiliza en ambientes donde la escalabilidad y robustez suelen ser factores importantes. **(Castro Fernández, 2010)**

La implementación de CORBA en un sistema como SURIA Vision no sería la más factible dada la inexperiencia de los desarrolladores del proyecto con tecnologías de este tipo. Además, la especificación se ha vuelto complicada por la gran variedad de características que sus implementadores han agregado a través de los años de refinamiento, creando una curva de aprendizaje bien pronunciada.

1.2.1.2 ICE

Según **(Castro Fernández, 2010)** ICE (*Internet Communication Engine* o Motor de Comunicación de Internet)es un middleware libre, con licencia GPL, desarrollado por la compañía ZeroC y cuya motivación es responder a la complejidad de CORBA ofreciendo una funcionalidad similar. Esta funcionalidad no es otra que la de ofrecer un middleware capaz de proveer un escenario distribuido en el cual se pueda desplegar una aplicación orientada a objetos, consiguiendo que una máquina A acceda a la lógica que implementa una máquina B, por medio de llamadas a métodos remotos pertenecientes a objetos alojados en la máquina B. Esto se consigue haciendo que la máquina A tenga acceso a una interfaz que provea de los métodos que tiene ese objeto alojado en B. Esta interfaz además, tiene que ser genérica, es decir,

debe ser nuevamente una interfaz independiente de la plataforma o lenguaje utilizado. De este modo se facilita la escalabilidad además de hacer más versátil el sistema.

ICE surge por tanto con la motivación de cumplir con la funcionalidad que ofrece *CORBA*, siendo independiente de lenguaje y plataforma, pero ofreciendo muchas mayores facilidades a la hora de implementar la solución. Siendo sus objetivos proveer un *middleware* orientado a objetos, disponible para plataformas heterogéneas, facilitar la utilización de distintos paradigmas(uno de ellos el de publicación-suscripción), hacer una plataforma más sencilla, favoreciendo la implantación, uso y aprendizaje de la misma, lograr que la plataforma sea eficiente en cuanto al uso de *CPU*, memoria y ancho de banda y que la plataforma sea segura por defecto, sin que haya necesidad de añadir parches posteriormente. **(Castro Fernández, 2010)**

Algunas de las principales características de *ICE* son: se puede utilizar en entornos heterogéneos, gracias a la posibilidad de hacer que los datos intercambiados sean independientes de plataforma y lenguaje. Permite la comunicación asíncrona en su modalidad cliente-servidor, lo que implica, que un cliente utiliza su *proxy* para realizar una llamada a un método remoto que se encuentra en el servidor. Como cualquier llamada normal, se pueden incluir ciertos parámetros, y el hilo del cliente queda bloqueado hasta que el servidor termina de ejecutar ese método y devuelve una respuesta. Utiliza como servicios que provee su plataforma a: *Freeze, IceGrid, IceBox, IceStorm, IcePatchyGlacier,* que son un añadido al núcleo central de *ICE* y suministran a este de nuevas funcionalidades que en general suponen una mayor comodidad para el desarrollo.

Desventajas de ICE

A pesar de que es un *middleware* poderoso, esta tecnología no aplica al software que se quiere diseñar para Video Vigilancia, ya que su licencia GPL haría que el software en su totalidad también tuviera licencia GPL. Esta licencia permite instalar y usar un programa GPL en un ordenador o en tantos como el usuario determine, sin limitación. También permite modificar el programa para adaptarlo a las especificidades del usuario y además, permite distribuir el programa GPL tal cual o después de haberlo modificado. (Negri, 2005) Estas características son indeseables en un producto para comercializar, por tanto ICE no es aplicable a SURIA Vision.

1.2.1.3 XML-RPC

Según **(Kidd, 2001)**XML-RPC (XML-*Remote Procedure Call*, XML llamada a procedimiento remoto), es una sencilla vía de hacer llamadas a procedimientos remotos sobre Internet. Puede ser usado con Perl, Java, Python, C, C++ y PHP. Tiene implementaciones disponibles para *Unix*, *Windows* y *Macintosh*.
Cuando aparece XML-RPC, ya se encontraba en el mercado CORBA y su concepto esencial era eliminar todas las complejidades que esta tecnología traía consigo. Por lo tanto XML-RPC define de manera sencilla cómo enviar el nombre de un método y una lista de argumentos de un sistema a otro. La idea principal de XML-RPC es que un documento XML es utilizado para indicar el nombre de un método y la lista de argumentos que este necesita. Este documento XML es enviado a un servidor web utilizando HTTP POST. El servidor procesa el documento XML y ejecuta el método solicitado, retornando luego el resultado en otro documento XML. **(Cerami, 2002)**
En otras palabras, XML-RPC es un protocolo de llamadas a procedimientos remotos que usa XML para codificar las llamadas y HTTP como mecanismo de transporte. **(Tomlinson, et al., 2010)**
XML-RPC consiste de tres partes relativamente pequeñas:

- Modelo de datos XML-RPC: es un conjunto de tipos usados en el paso de parámetros, valores de retorno y fallas (mensajes de error).
- Estructuras de petición XML-RPC: es una petición de tipo HTTP POST que contiene información de métodos y parámetros.
- Estructuras de respuesta XML-RPC: son respuestas de tipo HTTP que contiene valores de retorno o información de fallas.

El modelo de datos es usado por las estructuras de petición y de respuesta. La combinación de las tres partes define una llamada de procedimiento remoto completa. **(Cerami, 2002)**
Algunas implementaciones de XML-RPC para el framework Qt y el lenguaje C++ se analizaron en la búsqueda de una solución a la problemática presentada. Las mismas se sintetizan como sigue.

Libiqxmlrpc

Es una implementación del estándar XML-RPC en C++. Creada por Anton Dedov. Usa licencias BSD 2-clause y LGPL 2.1. Es multiplataforma y aunque funciona para Windows aún no hay un procedimiento de construcción bien definido para la misma **(Dedov, 2013)**. Se requiere de la instalación previa de las

bibliotecas libxml2, OpenSSL y Boost (ver Glosario de términos). SURIA es un sistema multiplataforma, por lo cual se requiere de un intermediario que garantice la interoperabilidad en cualquier sistema operativo. Esta implementación es deficiente para Windows por lo que no se inserta al software.

Qutexr

Es una biblioteca de Qt 3.1 sencilla basada en XML-RPC. Incluye un cliente y un servidor para la validación de la especificación. Usa licencia GPL. Al estar basado en Qt 3.1 el desarrollador tiene que implementar algunas características para adecuarlo a versiones superiores del framework **(Doxygen, 2003)**. Por tal motivo y por su licencia esta solución no es factible de aplicar al proyecto SURIA, pues en el mismo se trabaja con Qt 4.

Libmaia

Es una biblioteca para Trolltechs Qt 4. Sus autores Karl Glatz y Sebastian Wiedenroth aún trabajan en la corrección de errores y en la adicción de funcionalidades. Usa licencia FreeSBD que es compatible con GPL y es aplicable a proyectos LGPL/GPL **(Wiedenroth, 2008)**. Por ser un software en desarrollo no es confiable su uso en el sistema SURIA.

Desventajas de XML-RPC

Demasiado cargado de declaraciones. **(Cooper, 2010)**

```
<struct>
    <member>
        <name>ROOM_ID</name>
        <value>
            <int>1</int>
        </value>
    </member>

    <member>
        <name>CODE</name>
        <value>
            <string>MR-101</string>
        </value>
    </member>

    <member>
        <name>NAME</name>
        <value>
            <string>Math Room</string>
        </value>
    </member>

    <member>
        <name>CAPACITY</name>
        <value>
            <int>30</int>
        </value>
    </member>
</struct>
```

Figura 3 Ejemplo de código XML-RPC

En la Figura 3 se muestra la declaración de una variable en XML-RPC y se hace evidente la cantidad de código que este protocolo genera. Sin embargo la misma declaración para XML queda resumida a dos líneas de código en la Figura 4, e incluso a una sola línea, como se muestra en la Figura 5.

```
<room><ROOM_ID>1</ROOM_ID><CODE>MR-101</CODE>
<NAME>Math Room</NAME><CAPACITY>30</CAPACITY></room>
```

Figura 4 Primer ejemplo de XML (Cooper, 2010)

```
<room ROOM_ID=1 CODE=MR-101 NAME="Math Room" CAPACITY=30 />
```

Figura 5 Segundo ejemplo XML (Cooper, 2010)

Muchas implementaciones de XML-RPC fallan al producir verdaderos objetos de nivel de lenguaje del tipo que los programadores esperan en lugar de los que requieren. Ejemplo: búsqueda de campos en tiempo de ejecución o sintaxis extra.

No permite la comunicación con lenguajes de programación que usen otro protocolo como SOAP. **(Milardovich, et al., 2005)**

No permite la elección de los caracteres a usar, mientras que en SOAP se puede elegir entre US-ASCII, UTF-8 y UTF-16.

1.2.1.4 Web Services

Los servicios web son un caso particular de mecanismo estándar para implementar la interacción entre los componentes *software*, mediante la invocación de métodos remotos, según plantea **(Rodríguez Archilla, 2010)**. El mismo autor agrega que los servicios web suponen una interconexión punto a punto que, por sí sola, no proporciona la capacidad de integración y flexibilidad frente a cambios que se necesitan en los sistemas informáticos de grandes organizaciones.

Según **(Saffirio, 2006)** el término *Web Services* describe una forma estandarizada de integrar aplicaciones web mediante el uso de *XML (Extensible Markup Language)*, *SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language)* y *UDDI (Universal Description, Discovery and Integration)* sobre los protocolos de la Internet. *XML* es usado para describir los datos y además es una

especificación desarrollada por *W3C*. Permite a los desarrolladores crear sus propios *tags*, que posibilita habilitar definiciones, transmisiones, validaciones, e interpretación de los datos entre aplicaciones y entre organizaciones.

SOAP se ocupa de la transferencia de los datos y es un protocolo de mensajería construido en *XML* que se usa para codificar información de los requerimientos de los *Web Services* y para responder los mensajes antes de enviarlos por la red. Los mensajes *SOAP* son independientes de los sistemas operativos y pueden ser transportados por los protocolos que funcionan en la Internet, como son: *SMTP (Simple Mail Transfer Protocol* o Protocolo simple de transferencias de correo), *MIME (Multi-PurposeInternet Mail Extensions* o Extensiones de correo de Internet de propósitos múltiples) y *HTTP (Hyper Text Transfer Protocol* o Protocolo de transferencias de hipertexto).

WSDL se emplea para describir los servicios disponibles y es un lenguaje especificado en *XML* que se utiliza para definir los *Web Services* como colecciones de punto de comunicación capaces de intercambiar mensajes. El *WSDL* es parte integral de *UDDI* y parte del registro global de *XML*, en otras palabras es un estándar de uso público (no se requiere pagar licencias ni derecho de autor para usarlo). Por último *UDDI* se ocupa de conocer cuáles son los servicios disponibles y es un directorio distribuido que opera en la web para dar permiso a las empresas de publicar sus *Web Services*, para que otras empresas conozcan y utilicen los *Web Services* que publican. Opera de manera análoga a las páginas amarillas.

Uno de los usos principales de los *Web Services* es permitir la comunicación entre las empresas y entre las empresas y sus clientes. Los mismos permiten a las organizaciones intercambiar datos sin necesidad de conocer los detalles de sus respectivos Sistemas de Información. También permiten a distintas aplicaciones, de diferentes orígenes, comunicarse entre ellas sin necesidad de escribir programas costosos, ya que la comunicación se hace con *XML*. Los *Web Services* no están ligados a ningún Sistema Operativo o Lenguaje de Programación. **(Saffirio, 2006)**

Los *Web Services* han crecido en popularidad y han comenzado a mejorar los procesos de negocios. De hecho, algunos postulan que los *Web Services* están generando la próxima evolución de la Web.

Desventajas de los Web Services

Aunque son muchas las ventajas de este tipo de middleware, los mismos presentan algunas desventajas remarcables.

- Serializan objetos a través de XML los cuales están contenidos en los mensajes SOAP provocando así que solo se puedan manipular elementos que puedan ser completamente expresados en XML. **(Chand, 2007)**
- Rendimiento. El exceso de código que genera XML puede causar que la serialización de SOAP sea más lenta que la del formato binario. Adicionalmente, la manipulación del tipo de dato *string* es muy lenta cuando se compara al procesamiento de los bits individuales de un flujo binario **(Chand, 2007).** Se habla de conversión a binario porque es el formato que se pretende aplicar en la propuesta a presentar, aprovechando las facilidades que ofrece el framework Qt para la misma.
- Los servicios web son un modelo de programación sin estado, lo que implica que las peticiones entrantes son manipuladas de manera independiente. Además cada vez que un cliente invoca un servicio web, un nuevo objeto es creado para servir la petición. Para mantener el estado entre las peticiones el programador debe implementar su propia solución lo cual puede ser costoso debido a que los servicios web usan recursos extensivos de memoria. **(Sami, 2010)**

Para usar este tipo de tecnología se necesita establecer un servidor web. En SURIA, realizar esta tarea supondría de mucho esfuerzo por parte de los programadores debido a la cantidad de módulos existentes en el proyecto, que resultaría en una pérdida de tiempo. Sumado a las desventajas mencionadas, los servicios web se excluyen de la solución.

1.2.2 Proyecciones tecnológicas a nivel nacional

El uso de sistemas distribuidos y middleware para la intercomunicación se ha extendido al ámbito nacional. Por tal motivo algunas asociaciones, entre la variedad de empresas e instituciones que se dedican a la producción de software, se han interesado en el desarrollo de esta clase de software que ha ganado terreno en el entorno mundial.

XYMA Safe Vision

Una de las primeras empresas cubanas que ha dado buenos frutos en el tema de los sistemas distribuidos es Desarrollo de Aplicaciones, Tecnologías y Sistemas (DATYS), la cual sacó a la palestra pública su software XYMA Safe Vision, Sistema de Circuito Cerrado de Televisión, en el año 2006. Se trata de un sistema de video vigilancia basado en el protocolo IP mediante cámaras de video digitales conectadas a

redes de datos. Admite además, cámaras analógicas mediante servidores de videos. Su objetivo es el monitoreo en tiempo real y la vigilancia de instalaciones y exteriores. **(DATYS, 2011)**

Está compuesto por varios módulos interconectados, que se comunican y comparten la base de datos donde se almacena el sistema. Entre sus prestaciones tiene la video-vigilancia, desde los más diversos escenarios: monitoreo y grabación de video con audio, revisión y manejo de grabaciones de videos, grabaciones de forma manual, programada, continua y por detección de movimiento, sistema de alertas, de análisis inteligente de video y de reportes, con posibilidades de configuración. **(DATYS, 2011)**

La aplicación fue ideada para clientes con sistema operativo Windows en sus versiones XP SP2, 2003 server SP1 ó 7.Establece la comunicación entre componentes a través de sockets y mediante la arquitectura TCP/IP. Este software no incluye en su solución ningún middleware en particular. El sistema está desarrollado en el IDE Visual Studio 2010 y el lenguaje C# 2.0.

Se ha comprobado en diversos entornos de hotelería, inmigración aeroportuaria, emisor de moneda y tiendas, y en instituciones del Ministerio del Interior en general. Actualmente el software se encuentra en su versión 2.9, la cual está en desarrollo.

Se contactó a varias instituciones del país que podrían tener instalados sistemas distribuidos tales como la oficina central de la Empresa de Telecomunicaciones de Cuba (ETECSA), la empresa Softel, el GADI en la Aduana del puerto de La Habana, la Jefatura de la Aduana, el Centro de Estudios de Investigaciones y Sistemas (CEIS) del Instituto Superior Politécnico José Antonio Echeverría(CUJAE), así como el Complejo de Investigaciones Tecnológicas Integradas (CITI) del Ministerio del Interior (MININT). También se contactó con la facultad de Matemática y Computación de la Universidad de la Habana (UH). Esta investigación arrojó los siguientes resultados:

- En ETECSA se usa software propietario adquirido mediante la comercialización, ya sea con empresas cubanas o con extranjeras, según Yunel del Toro Montoya, Jefe del Grupo de Informática de esta institución.
- En la empresa Softel Soluciones Informáticas, actualmente no se está desarrollando ninguna aplicación distribuida. Se trabaja en el desarrollo de aplicaciones para instituciones pequeñas.
- En el GADI de la Aduana del puerto se tiene instalado el sistema distribuido de video vigilancia SURIA Vision 1.0. El contacto que proporcionó la información fue el Jefe del Centro de Automatización de la Aduana, Carlos Anasagasti Angulo.

- En el CEIS no se ha desarrollado ninguna aplicación distribuida, pero en el CITI existen dos proyectos que aplican el paradigma de sistemas distribuidos en su implementación. Tales proyectos son Observatorio Tecnológico Inteligente y Plataforma para la Integración de la Información, ambos del Programa de Investigación 1 Informática Aplicada del CITI. En el proyecto Plataforma para la Integración de la Información se usan los Servicios Web y otras tecnologías para lograr la integración de aplicaciones.
- En el grupo *Web Oriented Object* (WEBOO) de la UH, el Dr. Miguel Katrib y su grupo de desarrollo hacen uso de los Servicios Web.

La UCI como centro de referencia de desarrollo de software altamente calificado también ha elaborado sistemas computacionales distribuidos, siendo un grupo de especialistas del Departamento de Construcción de Componentes (CEDIN) de la facultad 5, creadores del sistema de tipo *Supervisory Control and Data Acquisition* (SCADA, Supervisión, Control y Adquisición de Datos), "Guardián del ALBA". Software que surge para automatizar y gestionar los procesos industriales orientados a la producción de petróleo y gas de la empresa Petróleos de Venezuela - Sociedad Anónima (PDVSA). Este sistema incluye la implementación ACE+TAO.

Otro sistema de relevancia desarrollado en la UCI es SURIA Vision, del centro GEYSED de la facultad 6.

1.3 Descripción actual del dominio del problema

El proyecto SURIA centra su atención en el proceso de video vigilancia para fortalecer la seguridad en las instituciones donde sea desplegado. La ejecución del proyecto incluye la infraestructura tecnológica para soportar un conjunto de cámaras de seguridad, que se puedan gestionar dentro de una red de datos asegurando la visualización, almacenamiento y trasmisión de los flujos de videos generados en cada uno de los dispositivos de adquisición. Por otra parte, se incluye una solución de software que permite:

- La obtención de los flujos de videos desde los dispositivos.
- La visualización de los flujos de vídeos de manera concurrente en una sala de operaciones.
- El almacenamiento y recuperación de los flujos de vídeos basados en un motor de reglas configurables incluido en la solución.

El *software* en su totalidad incluye un módulo para la visualización de cámaras (Visor), un subsistema para la grabación de los flujos de video provenientes de las cámaras en el sistema (Grabador) y un módulo para la recuperación de las grabaciones (Recuperador). Además cuenta con un módulo que garantiza la

comunicación entre todos los módulos o subsistemas de forma general (Gestor), un subsistema de análisis de video (Análisis) y un componente que garantiza la configuración del gestor, gestión de usuarios y roles, gestión de permisos a las cámaras y gestión de cámaras (Administración).

SURIA Vision está desplegado en la ADUANA del puerto desde el año 2011, y ha sido acogido con gran aceptación debido a las prestaciones que ofrece. Ha mostrado un alto nivel de satisfacción por parte del cliente.

1.3.1 Descripción de la situación problemática

Actualmente el sistema SURIA se encuentra en un proceso de migración o reconstrucción a tecnología libre, debido a los fines comerciales que se ha trazado la empresa ALBET (comercializadora de todos los productos de software elaborados en la UCI) con la aplicación. Esta reconstrucción se debe a que el software fue implementado en sus primeras versiones con el lenguaje C# 2.0, el IDE Visual Studio y se incluyó además el middleware de *.NET Remoting* para garantizar la interoperabilidad entre sus diversos componentes.

Por tanto, las tecnologías usadas para la creación de la nueva versión libre de SURIA son el framework Qt y el lenguaje C++, decisión tomada a nivel de los directivos del departamento GEYSED. Las mismas son muy factibles para el desarrollo de aplicaciones de alto valor agregado.

Una de las funcionalidades de SURIA a desarrollar es el ORB. En el proceso de investigación de una solución factible a este problema, se tuvieron en cuenta diversas implementaciones de intermediarios en lenguaje C++ a nivel mundial y nacional, para agregarlas a la solución, sin embargo, varias razones las hacen inutilizables para el sistema. Algunas de las cuales se especifican a continuación:

- La falta de experiencia de los desarrolladores del proyecto en el tema del uso e implementación de middlewares.
- Uso de licencia GPL en su especificación.
- Complejidad de implementación (que unido a la falta de experiencia de los desarrolladores revocan el uso de determinadas soluciones)

Adicionalmente el proyecto no cuenta con una documentación ingenieril que guíe el proceso de desarrollo del ORB.

Por tales motivos se precisa el diseño de un ORB, generando toda la documentación ingenieril necesaria para su posterior implementación, así como garantizando el uso de tecnologías libres en su desarrollo.

Para lo que se plantea el diseño usando el framework Qt y el lenguaje C++. También haciendo uso de patrones que garanticen el correcto funcionamiento del ORB y que dinamicen el proceso.

1.4 Caracterización de la metodología de desarrollo de software

Según **(Pressman, 2010)** las metodologías de desarrollo de *software* son el marco de trabajo que colecciona un conjunto de pasos y procedimientos que se deben seguir para organizar, controlar y planear el proceso de desarrollo de un software. En la actualidad existen variedad de metodologías (clasificadas en dos tipos: robustas y ágiles) que brindan soluciones a los desarrolladores. Su uso es degran importancia en la gestión eficiente de un proyecto. Entre las metodologías ágiles más comunes se puede citar a *Scrum*, *XP (Extreme Programming)* y *Crystal Methodologies*, las cuales tienen características específicas en el tipo de proyecto a desarrollar. Una metodología de tipo robusta es *RUP (Rational Unified Process)*, la cual genera al detalle toda la documentación y los artefactos de cada proceso del desarrollo del *software*, siendo la metodología que se ajusta al proyecto por las características descritas a continuación.

1.4.1 RUP

La metodología RUP (Proceso Unificado de Desarrollo de Software) es la escogida luego del análisis entre otras existentes,ya que es muy flexible y puede modificarse y ajustarse a las necesidades objetivas del trabajo. RUP se adapta con facilidad a cualquier ambiente en el que se desarrolle. Vale destacar además que es utilizada en el proyecto Video Vigilancia. También es la metodología aplicada por *Visual Paradigm* que será la herramienta CASE seleccionada para el proceso. Otro motivo de la elección es el monto de información a archivar que requiere el exitoso diseño del ORB, el cual permite una documentación gráfica de vistas de clases, y de casos de uso.RUP además posee una vasta documentación que permitirá su mejor utilización y servirá de guía en el uso de UML.

1.5 Tecnologías, herramientas y lenguajes para el desarrollo del sistema

Para el adecuado diseño del software en cuestión es preciso escoger las herramientas, tecnologías y lenguajes de programación más convenientes a usar según la naturaleza del sistema. Las mismas serán cruciales para obtener un software de calidad. Aunque ya se ofreció un preámbulo de las tecnologías a usar, el epígrafe argumenta la selección.

1.5.1UML

El Lenguaje de Modelado Unificado posee una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases y objetos, hasta la implementación y configuración con los diagramas de despliegue. *UML* sirve para el modelado completo de sistemas complejos, tanto en el diseño de los sistemas *software* como para la arquitectura *hardware* donde se ejecuten. Además es un método formal de modelado, lo cual le aporta las siguientes ventajas según (Jacobson, y otros, 2000): mayor rigor en la especificación y permite realizar una verificación y validación del modelo realizado.

UML se ha convertido en el estándar tan deseado para representar y modelar la información con la que se trabaja en las fases de análisis y, especialmente, de diseño. Los objetivos de *UML* son muchos, pero se pueden sintetizar sus funciones **(G. Booch, 1999)**: (1) Visualizar: *UML* permite expresar de una forma gráfica un sistema de forma que otro lo puede entender. (2) Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.

La selección de este lenguaje para modelar el diseño del ORB a realizar, se basa en que es necesario que los desarrolladores de SURIA tengan una descripción entendible y detallada del conjunto de clases y componentes a implementar posteriormente.

1.5.2 Herramienta CASE

Según **(INEI, 1999)** las Herramientas *CASE (Computer Aided Software Engineering* o Ingeniería de Software Asistida por Computadora) son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de *software* y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un *software*. Las herramientas *CASE* también permiten a los analistas tener más tiempo para el análisis y diseño minimizando el tiempo para codificar y probar.

Visual Paradigm

La herramienta *Visual Paradigm* será usada en el proceso de desarrollo del *software*, ya que el proyecto Video Vigilancia la usa por su disponibilidad en múltiples plataformas, por su diseño centrado en casos de uso y enfocado al negocio que genera un *software* de mayor calidad. También por sus capacidades de

ingeniería directa e inversa, su modelo y código que permanece sincronizado en todo el ciclo de desarrollo, la disponibilidad de múltiples versiones, para cada necesidad. Además de tener licencia: gratuita y comercial, ser fácil de instalar y actualizar, posee compatibilidad entre ediciones y soporte de UML en su versión 2.1.

1.5.3 Lenguaje de programación C++

Para el desarrollo del trabajo se utilizará el lenguaje C++, que es un lenguaje de programación orientado a objetos y permite realizar aplicaciones comercializables sin la necesidad de pagar una licencia. C++ ofrece gran cantidad de recursos, como son las formas de encapsulamiento, la sobrecarga de operadores y funciones, la herencia y el polimorfismo.
Un componente creado con este lenguaje obtiene mejores resultados en cuanto a la velocidad de trabajo. Es versátil, flexible y eficiente, cualidades que serían aprovechadas en la rápida implementación del ORB. Además, es el lenguaje en el que se han desarrollado las funcionalidades del proyecto Video Vigilancia en su versión libre.

1.5.4 Framework Qt

Qt es un framework de desarrollo multiplataforma que permite la creación de aplicaciones con interfaz gráfica o de consola. Presenta una librería de clases de C++ intuitiva, es portable desde sistemas de escritorio hasta sistemas operativos embebidos. Es un entorno de desarrollo multiplataforma alcanzando un alto rendimiento en las mismas. Por tal motivo este es el framework que se usará para el diseño eficiente de las clases del ORB. Además de ser el utilizado en el desarrollo del sistema SURIA Vision.

Conclusiones

- ✓ El análisis de los diferentes conceptos asociados a la investigación llevó a la conclusión de que el sistema SURIA Vision es un sistema distribuido, que aunque tiene varios módulos independientes se relacionan entre sí en mutua cooperación y se presentan ante los usuarios finales como una única aplicación. El middleware que se usó en sus primeras versiones para la creación del intermediario de peticiones de objeto fue .*NET Remoting* el cual está en proceso de cambio a tecnología Qt y lenguaje C++.

- ✓ Al observarse las características del sistema SURIA y el objetivo que persigue la UCI con su despliegue, que no es otro que la comercialización con países interesados, se concretó en la necesidad de obtener un sistema funcional y libre de licencias en su implementación.
- ✓ Aunque existen otras variantes de *middlewares* para el diseño del intermediario de peticiones, se concluye que:
    - ➢ CORBA, es una tecnología muy difícil de implementar que requiere de desarrolladores de experiencia en el tema de middleware para su correcto despliegue.
    - ➢ ICE usa licencia GNU GPL, la cual hace que el software en su totalidad adquiera dicha licencia y pueda ser libremente modificado y distribuido, lo que no es deseable para ALBET, empresa comercializadora de los *software* producidos en la UCI.
    - ➢ XML-RPC es un protocolo demasiado cargado de declaraciones en su código, haciéndose muy grande y complejo.
    - ➢ *Web Services* solo manipulan elementos XML.
- ✓ En el país aunque existen algunas empresas e instituciones trabajando el tema de los sistemas distribuidos, no se encontró ningún ORB creado en tecnología Qt y lenguaje C++, evidenciándose la relevancia del trabajo.
- ✓ Para desarrollar un ORB con implementación libre de licencias propietarias, el framework Qt con el lenguaje de programación C++ resultan tecnologías apropiadas, así como la herramienta Visual Paradigm con UML, en apoyo de la metodología de desarrollo seleccionada: RUP.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Para comprender el contexto en el cual se desarrolla el sistema, se determinó desarrollar un Modelo de Dominio, donde se expone un marco conceptual y las relaciones entre estas definiciones. Por otra parte, se enumeran los requerimientos funcionales y no funcionales, agrupándose los primeros en Casos de Uso, con el fin de estructurar el Diagrama de Casos de Uso del Sistema.

2.1 Modelo de Dominio

El Modelo de Dominio o Modelo Conceptual, permite de manera visual mostrar al usuario los principales conceptos que se manejan en el dominio del sistema en desarrollo. Un Modelo del Dominio es una representación de las clases conceptuales del mundo real, no de componentes software. El modelo desarrollado no se trata de un conjunto de diagramas que describen clases de software u objetos de software con responsabilidades, sino que puede considerarse como un diccionario visual de las abstracciones relevantes, vocabulario e información del dominio. Aprovechando las bondades de los diagramas UML para representar conceptos, el Modelo de Dominio se presenta en forma de diagrama de clases donde figuran los principales conceptos y roles del sistema en cuestión. **(Semanat Aldana, et al., 2009)**

Un modelo del dominio es una representación de las clases conceptuales del mundo real, no de componentes software. No se trata de un conjunto de diagramas que describen clases software, u objetos software con responsabilidades. **(Larman, 2002)**

2.1.1 Descripción de los conceptos principales

Para una mayor comprensión del Modelo de Dominio estructurado en el epígrafe, se presenta el siguiente marco conceptual con las definiciones identificadas como parte del mencionado modelo:

Servidor: Es la máquina desde la cual se proveen recursos.

Aplicación Servidora: Gestor de recursos almacenados.

Objeto: Instancia que permite la comunicación entre aplicaciones.

Cliente: Máquina donde se encuentra la aplicación que solicita servicios o recursos.

Aplicación Cliente: Realiza las peticiones al objeto de la Aplicación Servidora.

2.1.2 Diagrama del Modelo de Dominio

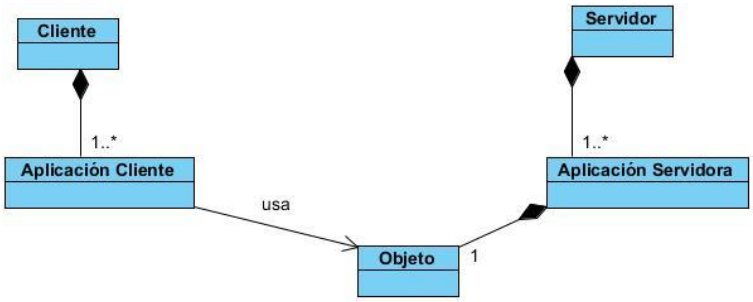

Figura 6 Modelo del Dominio

2.2 Especificación de requisitos de software

La captura de requerimientos constituye un proceso crítico de la ingeniería de software porque entre los puntos más importantes a tener en cuenta durante el desarrollo de un sistema de software se encuentran los requisitos que debe cumplir la aplicación, estos constituyen los cimientos de la solución propuesta y la evidencia para desarrollarla modelación del sistema.

2.2.1 Requisitos funcionales

Se denominan requerimientos funcionales a aquellas capacidades o condiciones con las que la aplicación debe cumplir, los mismos especifican funcionalidades que el sistema debe estar apto para efectuar. Estos deben ser redactados de forma comprensible para garantizar una única interpretación por parte de clientes, usuarios y desarrolladores y deben ser medibles y verificables.

Los requerimientos incluyen las acciones que podrán ser realizadas por el usuario, las acciones ocultas que el software debe ejecutar y las condiciones extremas a determinar por el sistema. **(Jacobson, et al., 2000)**

A continuación se enumeran los requisitos funcionales, o sea, las tareas que el sistema deberá ser capaz de efectuar, las cuales determinan el cumplimiento de los objetivos planteados para esta investigación:

RF1: Describir la interfaz del *CLIENT PROXY* y del objeto remoto.

Se tiene un *Interface Description Language* (IDL) que se encarga de implementar la interfaz del *CLIENT PROXY* del cliente una vez que este haya sido iniciado. Utiliza el patrón *Proxy* para ocultar la comunicación remota.

RF2: Realizar petición.

El cliente solicita un recurso determinado por lo que el *CLIENT PROXY* intercede para que los parámetros de petición sean elaborados por el *REQUESTOR* quien realmente realiza la petición. Esta petición se hace con toda la información de localización necesaria para identificar el objeto capaz de responderla, o sea, su host y puerto, así como su identificador. Con estos parámetros el *INVOKER* es capaz de generar la "Información de Invocación" que será indispensable para establecer la conexión posterior al objeto remoto. Esto sucede del lado del cliente.

RF3: Serializar objetos.

Se debe lograr la serialización de objetos, o sea, la conversión de los datos del objeto a un arreglo de bytes que permita la transmisión de datos a través de la red. Esta acción se realiza en los *MARSHALLERS* que se encuentran del lado del cliente y del lado del servidor.

RF4: Deserializar objetos.

Es el proceso inverso de la serialización y consiste en convertir los datos serializados a su formato original una vez que han sido transportados por la red. Se usa el MARSHALLER.

RF5: Buscar objeto remoto.

Para buscar un objeto es preciso tener su localización e identificador. Esta búsqueda localiza el objeto invocado si está disponible.

RF6: Conectar con el servidor.

El *CLIENT REQUEST HANDLER* gestiona las peticiones que realiza el cliente y es quien se conecta al *SERVER REQUEST HANDLER* en la máquina servidora. El patrón *Conector* es el encargado de establecer y configurar las conexiones al servidor remoto. Permite realizar más de una invocación por una misma conexión y un mismo puerto del servidor. Asegura la escalabilidad administrando las peticiones concurrentes y usando el patrón *Reactor* en la multiplexación.

RF7: Gestionar objeto.

Las instancias de los objetos remotos tienen un ciclo de vida específico en dependencia de su funcionalidad. Se usan *STATIC INSTANCES* para representar funcionalidades fijas en el sistema. Interviene el *LIFECYCLE MANAGER.*

RF8: Levantar errores.

En casos donde la petición no pueda ser enviada o donde no se pueda establecer la conexión al servidor se informa del error ocurrido al cliente usando *REMOTING ERRORS*.

RF9: Permitir la ejecución de eventos remotos.
Permitir el uso de *slot* y *signals* para cuando el servidor realice un evento, el cliente lo pueda recibir.

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Los mismos están vinculados con los requisitos funcionales, es decir, una vez se conozca lo que el sistema debe hacer podemos determinar cómo ha de comportarse. Pueden ser clasificados más a fondo, en requisitos de funcionamiento, requisitos de capacidad de mantenimiento, requisitos de seguridad, requisitos de confiabilidad, o uno de muchos otros tipos de requisitos del software.

Soporte

Deberá tenerse instalado el *framework* Qt en su versión 4.0 ó superiores.

Usabilidad

El software a diseñar deberá tener interfaces de aplicación de programas (API), bien definidas que permitan su futura implementación.

Requisitos de software

Framework Qt.
Sistema Operativo multiplataforma compatible al entorno de desarrollo Qt.

Requisitos de hardware

Tarjeta de red FastEthernet TP-Link.

2.3 Modelo de Sistema. Definición de casos de uso

El Modelo de Casos de Uso del Sistema muestra las funcionalidades que se deben implementar en la aplicación y el entorno en el cual funcionará la aplicación mediante el uso de actores y casos de uso. El mismo crea los cimientos necesarios para el desarrollo del proceso de Análisis y el Diseño del software. La identificación de los casos de uso constituye la guía a seguir por el ingeniero de software al frente del desarrollo de un sistema de software. **(Larman, 2002)**

2.3.1 Definición de actores

Los actores del sistema son la representación de aquellas personas o que interactúan de alguna forma con la aplicación y están asociados al cumplimiento de los requerimientos funcionales o procesos que responden a las funcionalidades definidas para el mismo.

Actor	Descripción
Sistema SURIA	Es un rol ficticio que representa al sistema SURIA Vision. Es el encargado de iniciar las funcionalidades del sistema.

Tabla 1 Actor del sistema

2.3.2 Listado de casos de uso

A continuación se muestra un resumen de los casos de uso que se tienen en cuenta para la correcta comprensión de las funcionalidades del sistema:

CU-1	**Invocar método de objeto remoto**
Actor	Sistema SURIA
Descripción	Se encarga de hacer la llamada a la funcionalidad específica del objeto remoto que dará respuesta a la solicitud. Para que la funcionalidad parezca una invocación local se implementan varios patrones que se pueden ver como roles dentro del sistema que garantizan la construcción y envío de peticiones a un objeto remoto específico. Realiza un proceso de búsqueda del objeto en caso de que sean más de uno los objetos que puedan responder la petición. Se encarga de gestionar la conexión a la aplicación servidora y de enviar las respuestas de la petición al cliente. Si suceden errores notifica al cliente.
Referencia	RF2, RF3, RF4, RF6, RF8, CU-2

Prioridad	Crítica

Tabla 2 CU Invocar método de objeto remoto

CU-2	**Obtener objeto remoto**
Actor	Sistema SURIA
Descripción	Se encarga de localizar los objetos remotos y notificar al servidor. Usa el Gestor del Ciclo de vida de los objetos para preguntar cuáles son los objetos que se encuentran disponibles y responde al cliente. Describe las propiedades del objeto remoto que se quiere obtener: su dirección, nombre e identificador.
Referencia	RF5, RF7
Prioridad	Alta

Tabla 3 CU Obtener objeto remoto

2.3.3 Diagrama de Casos de Uso del Sistema

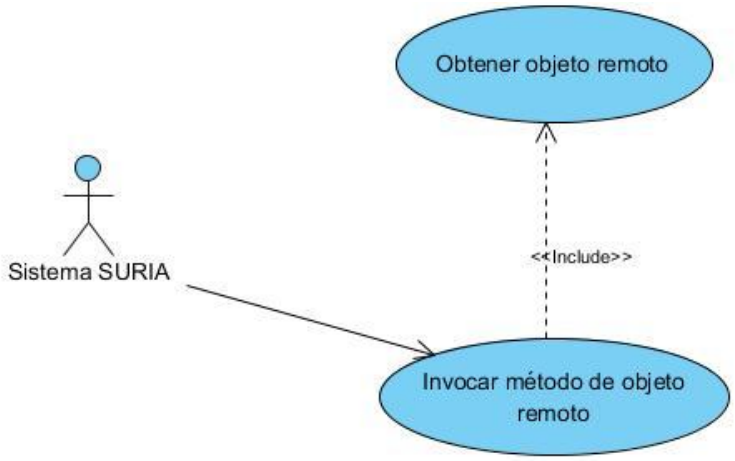

Figura 7 Diagrama de Casos de Uso del Sistema

2.3.4 Casos de uso expandidos

La descripción de casos de uso expandida es una vía para puntualizar las principales características de los mismos, ya que no basta con su representación gráfica.

Caso de Uso	Invocar método de objeto remoto
Actores	Sistema SURIA
Resumen	Este caso de uso inicia cuando el sistema SURIA invoca un método que pertenece a un objeto remoto. Es el encargado de establecer la comunicación con la aplicación servidora que se encargará de proveer los resultados esperados por el cliente.
Precondiciones	De existir la instancia del *Client Proxy*.
Flujo Normal de Eventos	
Sección “Hacer petición”	
Acción del Actor	Respuesta del Sistema
1. El sistema SURIA invoca un método de un objeto remoto desencadenando una serie de sucesos en la aplicación servidora.	2. Recibe la invocación. 3. Obtiene los parámetros de invocación. 4. Deserializa los parámetros obtenidos creando un archivo. 5. Usa archivo para invocar método del objeto remoto. 6. Serializa el resultado obtenido. 7. Elabora la respuesta con los datos serializados.

Flujos Alternos	
Acción del Actor	Respuesta del Sistema
1.El sistema SURIA invoca un método de un objeto remoto. 7. Recibe descripción de error ocurrido.	2. Recibe la invocación. 3. Obtiene los parámetros de invocación. 4. Ocurre un error de deserialización. 5. Levanta un Remoting Error para informar al cliente del error. 6. Describe el error.
No tiene Interfaz Visible	
Sección “Recibir respuesta”	
Acción del Actor	Respuesta del Sistema
2. Recibe respuesta. 3. Deserializa los datos de respuesta creando un archivo. 4. Envía datos deserializados al Client Proxy. 5. El Client Proxy traduce los datos a parámetros entendibles por elsistema SURIA. 6. Envía resultados traducidos.	1. Envía el resultado al cliente.

No tiene Interfaz Visible	
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
4. Recibe notificación de error.	1. Ocurre un error en el envío del resultado al cliente. 2. Identifica tipo de error mediante los *Remoting Errors*. 3. Notifica del error al cliente.
No tiene Interfaz Visible	
Poscondiciones	Se logra ejecutar un método de un objeto remoto.

Tabla 4 Descripción del CU Invocar método de objeto remoto

Caso de Uso	Obtener objeto remoto
Actores	Sistema SURIA
Resumen	Es iniciado por el Caso de Uso “Invocar método de objeto remoto”, ya que representa una inclusión de este. Se encarga de localizar los objetos remotos y notificar al servidor. Describe las propiedades del objeto remoto que se quiere obtener.
Precondiciones	Haberse iniciado el CU Invocar método de objeto remoto
Flujo Normal de Eventos	
Sección “Iniciar proceso de búsqueda”	
Acción del Actor	Respuesta del Sistema
1. El sistema SURIA invoca un método de un objeto remoto.	2. Recibe los parámetros de invocación. 3. Usa los parámetros para saber si el objeto se encuentra disponible.

	4. Envía la respuesta al cliente.
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
1. El sistema SURIA invoca un método de un objeto remoto. 4. Recibe notificación del error ocurrido.	2. Ocurre una interrupción de la red o un colapso del servidor. 3. Levanta un *Remoting Error* y notifica de este al cliente.
No tiene Interfaz Visible	
Sección “Procesar objetos”	
Acción del Actor	Respuesta del Sistema
1. El sistema SURIA invoca un método de un objeto remoto.	2. Recibe invocación. 3. Activa objeto remoto en petición. 4. Almacena en un archivo persistente las instancias que han sido eliminadas temporalmente de la memoria para poderlas restaurar si son solicitadas.

No tiene Interfaz Visible	
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
	1.Ocurre un colapso del servidor. 2. Genera Remoting Error. 3. Informa del error al cliente.
4. Recibe notificación de error.	
No tiene Interfaz Visible	
Poscondiciones	Se obtiene la localización del objeto.

Tabla 5 Descripción del CU Obtener objeto remoto

Conclusiones

Las características del software a diseñar quedaron agrupadas en requisitos funcionales y no funcionales permitiendo crear la base del diseño del mismo. El Modelo del Dominio permitió concebir al sistema de manera general, logrando la abstracción en los principales conceptos asociados.

Se concretaron los casos de uso del sistema, los cuales se representaron en un diagrama el cual contiene, además, la interacción que se desarrolla entre ellos, obteniéndose una generalización del funcionamiento del sistema a través de la descripción de cada uno.

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA

En el presente capítulo se lleva a cabo un análisis de la arquitectura seleccionada para el desarrollo de la investigación teniendo en cuenta que se trata de un sistema distribuido, como parte del flujo de trabajo Análisis y Diseño. El mismo es el flujo cumbre del presente trabajo.

En esta metodología se compactan los procesos de Análisis y Diseño en uno solo, pero cada uno de ellos cumple funcionalidades diferentes, las cuales conforman la vista lógica de la arquitectura de software en la construcción de aplicaciones dirigidas por modelos.

Se realizará el Análisis del sistema, modelando importantes diagramas como son: el Diagrama de Análisis y el Diagrama de Clases del Diseño y posteriormente se procederá a la modelación de la lógica del Negocio mediante el uso de clases, tratándose los principios del Diseño de la aplicación, donde se incluyen los Diagramas de Clases del Diseño y los Diagramas de Secuencia. También se validarán los requisitos de software y el diseño obtenido para comprobar su calidad.

3.1 Análisis de la arquitectura de software.

La arquitectura de software es una vista del sistema que incluye los componentes principales del mismo, la conducta de estos componentes según se les percibe del resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema **(Muñoz Peña, 2011)**. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones. Una de las definiciones que mayor aceptación ha tenido es la de Arquitectura del Software de la IEEE Std 1471-2000 que se define así: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución” **(Casanovas, 2004)**.

3.1.1 Arquitectura Cliente-Servidor para el ORB

Entre las arquitecturas o modelos de comunicación utilizados a nivel de software está la arquitectura P2P, que puede definirse como un sistema donde sus participantes o nodos son iguales y no tienen capacidades especiales que lo diferencien del resto. La aplicación de esta arquitectura implica que no hay un nodo central de procesamiento donde resida de manera única el estado de la simulación. Cada nodo

tiene la responsabilidad de administrar la evolución y de mantener una réplica local del estado de la simulación. En consecuencia, es necesario mantener consistente dicho estado en cada nodo porque los usuarios deben percibir la misma realidad en todo momento. **(Reynoso, et al., 2004)**

Otro modelo utilizado es la arquitectura cliente-servidor, donde el sistema es pensado en términos de nodos clientes que requieren servicios de un nodo servidor. Aplicado este esquema en el ámbito de aplicaciones distribuidas, el estado de la aplicación es mantenido únicamente en el servidor, cada evento producido por un participante (cliente, en los términos del modelo) es enviado al servidor para que este lo procese, evolucione el estado y luego comunique el nuevo estado a todos los clientes. El servidor es responsable de realizar todos los procesos de cálculos y el procesamiento de datos involucrados en los cambios de estado. **(Reynoso, et al., 2004)**

En el diseño del ORB se usará esta arquitectura porque permite un procesamiento centralizado del flujo de eventos en el servidor. Siendo esta característica de gran utilidad en el sistema SURIA. En el caso del ORB será necesario un servidor que facilite la conexión, la serialización y envío de datos y además que garantice una respuesta a los usuarios que solicitan recursos. En el caso del cliente, este debe obtener las referencias de los objetos remotos y para ello es necesario que se establezcan roles, para que el proceso suceda de la manera más transparente posible.

3.1.2 Arquitectura por capas para el ORB

El procesamiento de mensajes en un middleware de objetos distribuidos sigue una arquitectura por capas, como la mostrada en la Figura 8. La arquitectura por capas divide y encapsula las responsabilidades mutuas. Cada capa solo interactúa con las capas inmediatamente superior e inferior.

Figura 8 Capas del ORB (Volter, y otros, 2005)

Capa de Aplicación: Es la capa de más alto nivel de la arquitectura de procesamiento de mensajes de un middleware de objetos distribuidos y consiste de clientes que realizan invocaciones y objetos remotos que le prestan servicio a las invocaciones.

Capa de Invocación: Debajo de la capa de aplicación el CLIENT PROXY, el REQUESTOR y el INVOKER son los responsables de los procesos de serialización y deserialización y de multiplexación y desmultiplexación de invocaciones y respuestas.

Capa de Manipulación de peticiones: Esta capa es la responsable de las tareas básicas de establecer las conexiones y de pasar mensajes entre cliente y servidor.

Capa de Comunicación: Es responsable de definir el flujo básico de mensajes y de manejar los recursos del sistema operativo, tales como conexiones e hilos. En dependencia de la arquitectura esta capa usa el patrón *Reactor* y/o el patrón Acceptor/Connector.

De igual modo, el ORB a diseñar seguirá una arquitectura por capas para el manejo de mensajes, estableciendo diferentes niveles de organización y abstracción que permitan una mejor implementación en

el futuro. Asimismo, los diferentes niveles de las capas permiten el acceso a componentes específicos que serán muy útiles en el diseño.

3.1.3 Arquitectura Cliente-Servidor por Capas para el ORB

En conclusión el ORB a diseñar sigue una arquitectura de tipo Cliente/ Servidor por capas, ya que es preciso que los procesos en el sistema se realicen de una forma ordenada y jerárquica. En cada nivel se establecen responsabilidades que son comunes y que a su vez deben realizarse lo mismo para el lado del cliente que para el lado del servidor. En un esquema quedaría:

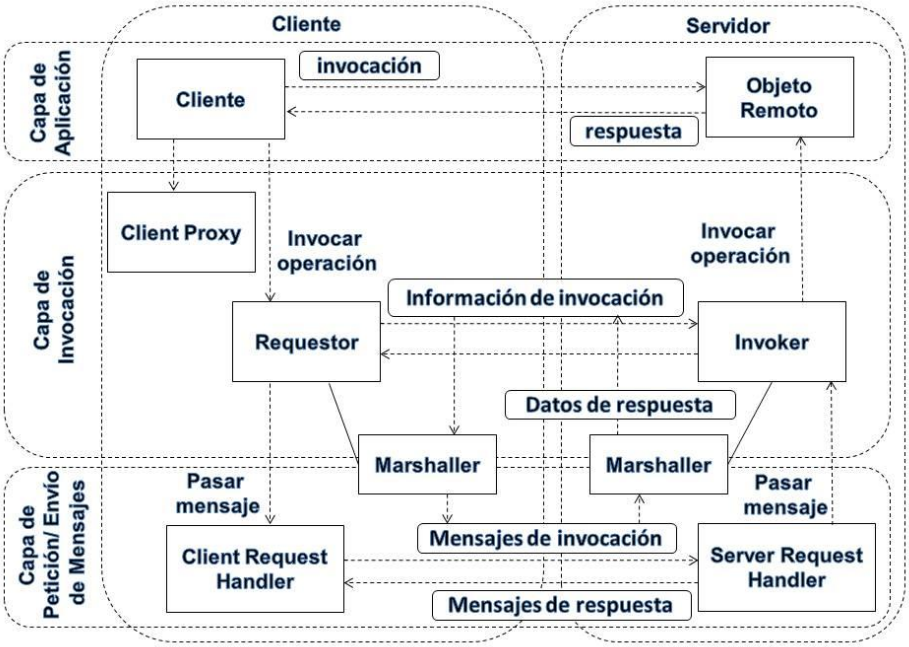

Figura 9 Interacción entre patrones del ORB (Volter, y otros, 2005)

En la Figura 9 se muestra la interacción entre los patrones básicos del diseño del ORB, así como la representación de la arquitectura Cliente-Servidor por Capas.

3.2 Flujos de trabajo de Análisis y Diseño.

El flujo de trabajo de Análisis y Diseño tiene como principal objetivo transformar los requisitos a una especificación que describa cómo implementar el sistema. El Análisis es, fundamentalmente, obtener una apreciación de las funcionalidades que poseerá la herramienta a desarrollar, para lo cual se apega a los requerimientos funcionales. El Diseño, por su parte, logra un mayor refinamiento al tener en cuenta los requisitos no funcionales, y se concentra más en "cómo" el sistema cumple sus funciones. **(Rodríguez Sánchez, 2011)**

Los objetivos de este flujo de trabajo son:

- Transformar los requerimientos en un modelo de diseño del sistema a implementar.
- Definir una arquitectura robusta para el software.
- Adaptar el diseño para que sea consistente con el entorno de implementación. **(Jacobson, et al., 2000)**

3.3 Modelo de Análisis.

El Modelo de Análisis se ocupa del procesamiento de los requisitos funcionales, creando un bosquejo de cómo llevar a cabo el funcionamiento dentro del sistema, incluidas las funciones significativas para la arquitectura. Este modelo sirve como un primer acercamiento al Diseño y es la derivación del análisis de los casos de uso.

El Modelo de Análisis abarca las clases que describen la realización de los casos de uso, sus atributos y las relaciones entre ellas, los cuales se identifican durante la construcción de dicho modelo. Con esta información se edifican los Diagramas de Clases del Análisis, que constituyen los conceptos en un dominio del problema, sin llegar al nivel de detalle que se logra en el Diseño. **(Rodríguez Sánchez, 2011)**

3.3.1 Diagramas de Clases del Análisis.

Los Diagramas de Clases del Análisis muestran la relación entre las clases, centradas en los requisitos funcionales. Las mismas poseen atributos y establecen relaciones de asociación, agregación/composición, generalización/especialización y tipos asociativos con otras clases. En la metodología RUP se clasifican las clases en:

- Clases Interfaz: Modelan la interacción entre la aplicación y los actores.

- Clases Controladoras: Coordinan la realización de uno o unos pocos casos de uso, regulando las actividades de los objetos que implementan la funcionalidad del caso de uso.
- Clases Entidad: Manejan la información que posee larga vida y que es, frecuentemente, persistente. **(Larman, 2002)**

A continuación se presenta el Diagrama de Clases del Análisis del Caso de Uso "Invocar método de objeto remoto". Otros Diagramas de Clases referentes a los demás Casos de Uso se expondrán en **"Anexos Figura 19"**.

Figura 10 Diagrama de Clases del Análisis CU Invocar método de objeto remoto

3.3.2 Diagramas de Comunicación.

Se presenta el diagrama de comunicación del caso de uso "Invocar método de objeto remoto". Los diagramas restantes se podrán encontrar en **"Anexos 3"**.

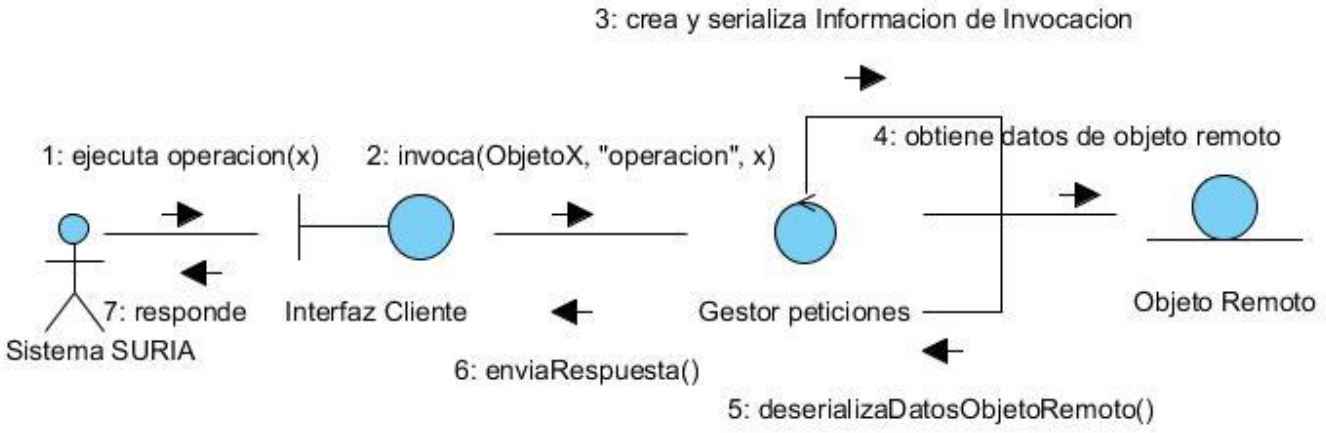

Figura 11 Diagrama de Colaboración CU Invocar método de objeto remoto

3.4 Modelo de Diseño.

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además el modelo de diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación. **(Pressman, 2010)**

3.4.1 Patrones de Diseño/Arquitectura.

Un patrón es "una regla de tres partes, que expresa una relación entre un cierto contexto, un paquete, y una solución". Los patrones son soluciones para los problemas recurrentes. Por consiguiente necesitan ser muy generales, a fin de que pueden ser aplicados a más que un paquete concreto. Sin embargo, la solución debería ser suficientemente concreta para ser prácticamente útil, y debería incluir una descripción de una configuración específica del software. Tal configuración consiste en los participantes del patrón, sus responsabilidades, y sus interacciones. El nivel de detalle de esta descripción puede variar, pero después de leer el patrón, el lector debería saber lo que él tiene que hacer para implementar la solución del patrón. **(Volter, et al., 2005)**

3.4.2 Fundamentación del empleo de patrones.

Los patrones a usar en el diseño del ORB son denominados patrones de software según **(Volter, et al., 2005)**. Destacar los patrones de diseño de los patrones de arquitectura es muchas veces complicado, ya que a menudo la distinción entre ellos depende de la situación o punto de vista.

Usar uno o varios patrones es muy factible en el desarrollo de una aplicación, ya que los mismos permiten tener una estructura de código común a todos los proyectos que implemente una funcionalidad genérica. La utilización de patrones de diseño, permite ahorrar grandes cantidades de tiempo en la construcción de software. Además, brinda mejor imagen de profesionalidad y calidad. **(Jacobson, y otros, 2000)**

En la actualidad el uso de patrones se ha convertido en una práctica común, ya que son empleados y se encuentran en la mayoría de los mejores sistemas informáticos a nivel mundial.

3.4.3 Patrones.

Para el desarrollo del ORB se necesitan varios patrones comunes entre los sistemas distribuidos. Dichos patrones son:

Broker: Separar la funcionalidad de comunicación en un sistema distribuido de su funcionalidad de aplicación aislando toda comunicación relacionada consiste en un *Broker*. El mismo oculta y media toda comunicación entre los objetos o los componentes de un sistema. Un *Broker* consiste de un *Requestor* del lado del cliente para construir y pasar las invocaciones, así como de un *Invoker* del lado del servidor que es el responsable de invocar las operaciones del objeto remoto objetivo. Un *Marshaller* de cada lado de la vía de comunicación gestiona la transformación de las peticiones y respuestas desde sus lenguajes de programación originales a un arreglo de *bytes* que puede ser enviado por el medio de transmisión. **(Volter, et al., 2005)**

Requestor: En la aplicación del cliente su utiliza un *Requestor* para acceder al objeto remoto. Al mismo se le suministra el identificador del objeto, su localización, el nombre de la operación y sus argumentos. El *Requestor* construye una invocación remota partiendo de estos parámetros y la envía al objeto remoto. Oculta los detalles de la comunicación distribuida del lado del cliente a los clientes. Para invocar una operación de un objeto remoto, el cliente pasa los datos necesarios para invocar la operación al *Requestor,* quien envía esta información a través de la red al objeto remoto. Gestiona los detalles del cruzamiento de procesos y del límite de máquina. **(Volter, et al., 2005)**

Client Proxy*:* Es una interfaz común entre la aplicación servidora y el cliente que se genera en el servidor una vez que el cliente se ha encendido. Traduce la invocación del cliente a parámetros entendibles por el *Requestor* y pasa la invocación. Recibe el resultado de la invocación del Requestor y se lo pasa al cliente. Aplica el patrón *Proxy* del GoF (Gang of Four, pandilla de los cuatro).

Invoker*:* Acepta las invocaciones de los clientes desde un *Requestor* Lee la petición y la deserializa para obtener el ID de objeto y el nombre de la operación. Luego despacha la invocación con sus parámetros de invocación deserializados al objeto remoto objetivo. Busca el objeto local correcto y su implementación de operación que ha sido descrita por la invocación remota, y la invoca.

Client Request Handler*:* Se encarga de manipular las peticiones de varios *Requestors* del lado del cliente. Abre y cierra las conexiones de red a las aplicaciones servidoras, envía peticiones, recibe

respuestas y las envía de regreso al *Requestor* apropiado. Se enfrenta a tiempos de espera, problemas de hilos y errores de invocación.

***Server Request Handler**:* Gestiona todos los asuntos de comunicación de la aplicación servidora. Recibe mensajes de la red, combina los fragmentos de mensajes para completar los mensajes y envía los mensajes al *Invoker* correcto para el procesamiento adicional. Gestiona conexiones e hilos.

***Marshaller**:* Permite que cada tipo de dato primitivo usado dentro de invocaciones de objetos remotos sean serializables a un flujo de byte. Usa clases del lenguaje C++ que permiten la serialización a byte tales como *QDataStream*, *QSslSocket* y *QTcpSocket.*

***Remoting Error**:* Detecta y propaga errores dentro del sistema. Distingue los errores que surgen de la distribución y comunicación remota, de errores que se levantan de la lógica de aplicación dentro de los objetos remotos.

Con el fin de distribuir las responsabilidades de las clases y establecer las relaciones entre ellas, para evitar sobrecargas de funcionalidades y que además exista dependencia entre clases, se han aplicado los patrones mencionados a los diagramas de clases elaborados. Estos permiten la reutilización del conocimiento, su uso como guía para el desarrollo y garantiza un producto robusto y acabado.

3.4.4 Diagrama de Clases del Diseño

En los Diagramas de Clases del Diseño se representan, de una manera simple y de fácil comprensión, los atributos y métodos principales de cada una de las clases que componen el sistema, sus relaciones y responsabilidades. Estos diagramas se utilizan para modelar la Vista del Diseño Estática de un sistema, lo que incluye modelar el vocabulario del sistema y las colaboraciones o esquemas. Constituyen también la base para los Diagramas de Componentes y de Despliegue. **(Jacobson, y otros, 2000)**

A continuación se presenta el Diagrama de Clases del Diseño del CU Invocar método de objeto remoto. Los diagramas restantes se incluirán en el documento **“Anexos”**.

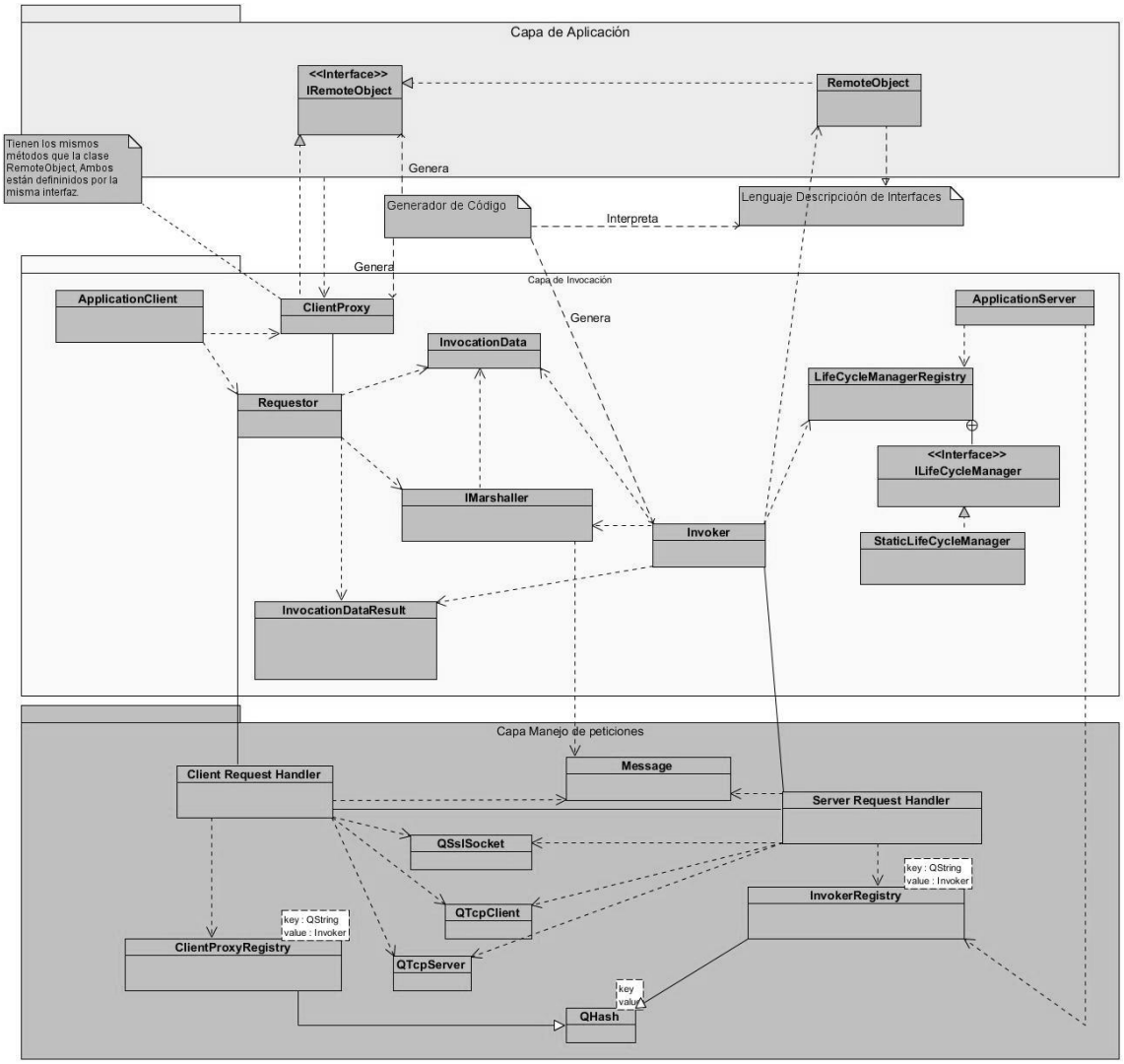

Figura 12 Diagrama de Clases del Diseño General

Descripción de algunas clases

Nombre: ClientProxy	
Tipo de Clase: Interfaz	
Para cada responsabilidad:	
Nombre:	someOperation (param: boolean) : float

Descripción:	Esta operación ejecuta un método del objeto remoto.

Nombre: Requestor	
Tipo de Clase: Controladora	
Para cada responsabilidad:	
Nombre:	invoke (targetObject: QString, method: QString, params: QHash<QString, QVariant>)
Descripción:	Esta operación invoca un método del objeto remoto pasándole por parámetro la dirección física del objeto, el método a ejecutar

Otras descripciones de clases se encuentran en el documento **"Anexos".**

3.4.5 Diagramas de Secuencia.

En los diagramas de secuencia se muestran las interacciones entre objetos mediante transferencia de mensajes entre objetos o subsistemas. Cuando se dice que un subsistema "recibe" un mensaje, esta afirmación significa que es un objeto de una clase del subsistema el que recibe el mensaje. Asimismo cuando un subsistema "envía" un mensaje, realmente es un objeto de una clase del subsistema el que envía el mensaje. El nombre del mensaje debe indicar una operación del objeto que recibe la invocación o de una interfaz que el objeto proporciona. **(Jacobson, et al., 2000)**

Se presenta a continuación el diagrama de secuencia del CU "Invocar método de objeto remoto". Los demás diagramas se pueden obtener en **"Anexos"**. Este diagrama muestra el flujo básico de la invocación de un método contenido en un objeto remoto. La aplicación cliente, en este caso, SURIA, desconoce que está invocando un método remoto, ya que el proceso se realiza de manera transparente, aparentando una llamada a un método local.

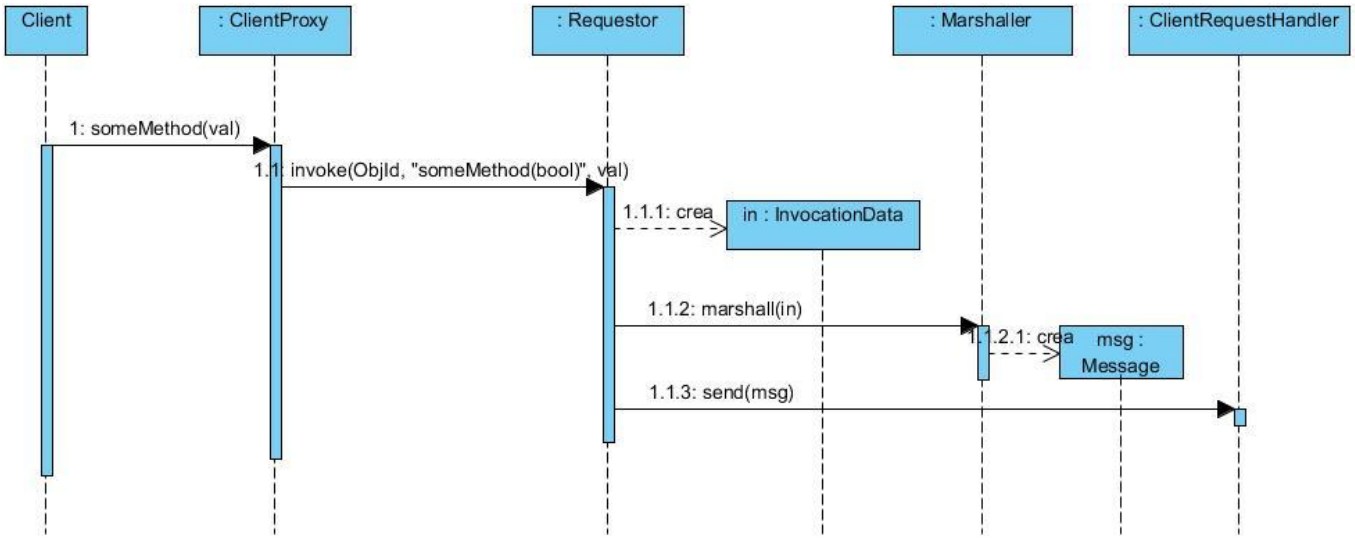

Figura 13 Diagrama de Secuencia CU Invocar método de objeto remoto (Del lado del Cliente)

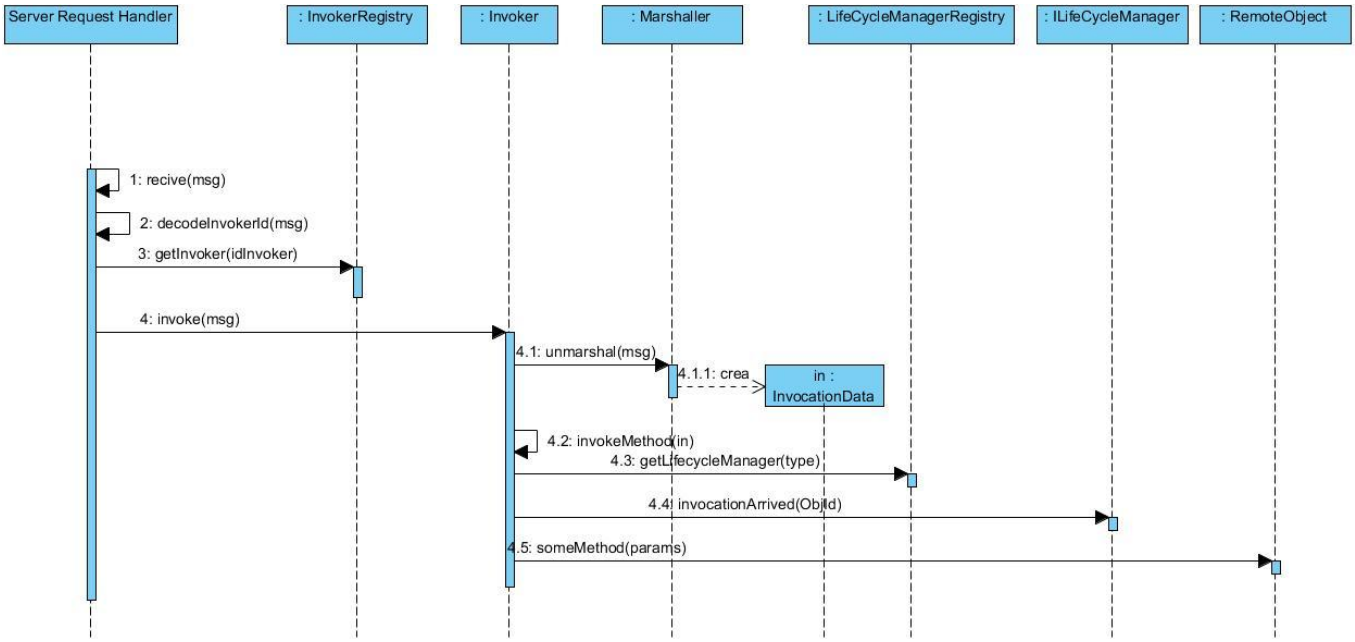

Figura 14 Diagrama de Secuencia CU Invocar método de objeto remoto (Del lado del Servidor)

3.5 Validación de la solución propuesta

El exitoso desarrollo de un software depende, además de la correcta elección de las tecnologías a usar, del empleo de métricas de comprobación que erradiquen problemas en los requisitos o casos de uso del sistema.

3.6 Método de validación de los resultados

En todas las investigaciones es muy importante el análisis de los resultados de la propuesta de solución al problema en cuestión, para determinar si los objetivos fueron cumplidos y si los beneficiados del desarrollo del sistema obtendrán lo que esperan.

Para la validación de los resultados del presente trabajo se usaron varios métodos garantizando que fuesen comprobados los artefactos generados durante la investigación.

3.6.1 Técnica de validación usando matriz de trazabilidad

La trazabilidad de requisitos se define como la habilidad para describir y seguir la vida de un requisito hacia sus orígenes o hacia su implementación. Esta técnica es clave para conseguir una exitosa gestión de los requisitos.

La trazabilidad permite conocer los elementos que son afectados al ocurrir un cambio en algún elemento asociado al primero, o sea, cuando algún requerimiento es modificado, todas las relaciones asociadas a ese requerimiento se convierten en “sospechosas”. En tal caso se deben revisar los cambios y determinar si los elementos asociados deben ser cambiados también.

Entre los requerimientos y los casos de uso del sistema existe una trazabilidad directa, ya que los casos de uso no son más que funcionalidades del sistema expresados en diagramas. Estos además, deben referenciar al menos un requerimiento, o sea, cada requerimiento debe quedar reflejado en un caso de uso y cualquier modificación que exista en algún requerimiento puede afectar al caso de uso correspondiente. De la misma forma, si un caso de uso es modificado, se debe revisar esa modificación y ver qué requerimiento pueda estar afectado también. Todo este control es posible mediante la trazabilidad que existe entre ambos elementos.

Para facilitar el trabajo de determinar las relaciones entre los requisitos y el seguimiento de los mismos, se generan matrices de trazabilidad. Cada matriz identifica los requisitos relacionados con uno o más elementos del sistema. Se muestra la matriz de trazabilidad de requisitos a casos de uso del sistema quedando expuesta de manera sencilla la relación entre casos de uso y requisitos.

Requisitos	CU1 Obtener objeto remoto	CU2 Invocar método de objeto remoto
RF1	X	
RF2	X	
RF3	X	
RF4	X	
RF5		X
RF6	X	
RF7		X
RF8	X	
RF9	X	

3.6.2 Métricas de la calidad de especificación de requisitos (Ausencia de ambigüedad)

El primer paso para realizar el análisis de cualquier proyecto de software consiste en la obtención de requisitos (Ingeniería de Requerimientos) los cuales definen lo que se va a producir. Lo que se pretende con una buena Ingeniería de Requerimientos es reducir costos y retrasos del proyecto, mejorar la calidad del software, evitar el rechazo de los usuarios finales.

Para realizar la validación de los requisitos existe toda una lista de características que sugieren el uso de una o más métricas como son: especificidad (ausencia de ambigüedad). **(Pressman, 2005)**

Esta es la métrica a usar en la validación de los requisitos del trabajo y para la ejecución de la misma se tiene que: n_r representa el número de requisitos del sistema. Esta variable se aplica a la fórmula $n_r = n_f + n_{nf}$, donde n_f representa la cantidad de requisitos funcionales del sistema y n_{nf} es el número de requisitos no funcionales.

Luego se mide la especificidad de los requisitos mediante la fórmula:

$Q = n_{ui} / n_r$

Donde n_{ui} es el número de requisitos para el que los revisores tuvieron interpretaciones idénticas. A medida que el valor de Q se acerca a 1 disminuye la ambigüedad de la especificación.

Para proceder al uso de la métrica y validar la propuesta, se hicieron 3 revisiones con 3 revisores consultados. A fin de obtener la menor ambigüedad posible y mayor claridad en los requisitos de forma

que se obtengan las especificidades de un software de calidad que permita la interoperabilidad entre los componentes del sistema SURIA.

Primera revisión

Algunos requisitos funcionales no estaban redactados de manera entendible y presentaban problemas de ambigüedad. Además no describían correctamente la funcionalidad a ejecutarse. Los mismos se listan a continuación: RF1, RF3, RF6, RF7, RF8, RF9. De una totalidad de 9 RF y 4RNF, los errores representaban un 46 por ciento aproximadamente.

Quedando el cálculo de la especificidad:

$Q_1 = 7/13$

$Q_1 = 0.54$ (aproximadamente)

Segunda revisión

Se corrigieron los errores de la primera revisión, pero aún quedaban algunos requisitos que no estaban bien redactados y presentaban ambigüedad. Tal es el caso de los requisitos funcionales RF6, RF7, RF9. Esta vez los errores representaban un 23 porciento.

El cálculo de la especificidad quedó de la siguiente forma:

$Q_2 = 10/13$

$Q_2 = 0.77$ (aproximadamente)

Tercera revisión

Esta vez los revisores comprobaron que los requisitos en su gran mayoría no eran ambiguos. Representando un porciento de error de 7 y obteniéndose un valor para la especificidad de:

$Q_3 = 12/13$

$Q_3 = 0.92$ (aproximadamente)

Luego de aplicada la métrica se eliminó la ambigüedad existente en casi todos los requisitos del sistema, obteniéndose un grado de ambigüedad bastante bajo, y apreciándose que el valor de Q se acerca a 1. Por consiguiente, el nivel de calidad del proceso de refinamiento de requisitos fue alto.

3.6.3 Métricas para la validación del diseño

Para la medición del diseño se usan métricas básicas inspiradas en el estudio de la calidad del diseño orientado a objetos referenciadas por Pressman, teniendo en cuenta que este estudio brinda un esquema sencillo de implementación y que, a su vez, cubre los principales atributos de calidad de software. Las

métricas escogidas para la validación del diseño fueron las métricas orientadas a clases: Tamaño Operacional de Clase (TOC) y Acoplamiento entre Clases (AC).

Tamaño Operacional de Clases (TOC)

El tamaño general de una clase se puede determinar siguiendo los planteamientos descritos a continuación:

- El número de atributos (tanto atributos heredados como derivados de la instancia) que están encapsulados en la clase.
- El número total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.

Estos dos valores son sumados de acuerdo a la clase que se analiza y los resultados son tomados como umbrales que luego son comparados en la tabla que se presenta a continuación para determinar el TOC de cada clase. **(Pressman, 2005)**

Clasificación	**Valores de los umbrales**
Pequeño	<= 20
Medio	>20 y <= 30
Grande	>30

Tabla 6 Clasificación de Clases

La existencia de valores grandes de TOC demuestra que una clase puede tener demasiada responsabilidad, lo cual reduciría la reutilización de la clase y haría complicada la posterior implementación. Ocurre lo opuesto si existen valores de TOC pequeños. Por último se calculan los promedios correspondientes a los diferentes valores para tener una estimación general del sistema. **(Pressman, 2005)**

Atributos	**Categoría**	**Criterio**
Responsabilidad	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	< 2*Promedio
Complejidad de implementación	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio

	Alta	< 2*Promedio
Reutilización	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	< 2* Promedio

Tabla 7 Criterios de evaluación de la métrica TOC

La aplicación de esta métrica arrojó los siguientes resultados:

No.	Clases	Cantidad de atributos	Cantidad de operaciones	Tamaño
1	IRemoteObject	1	1	Pequeño
2	RemoteObject	1	1	Pequeño
3	ApplicationClient	0	2	Pequeño
4	ClientProxy	0	1	Pequeño
5	Invoker	0	3	Pequeño
6	InvocationData	3	3	Pequeño
7	Requestor	0	1	Pequeño
8	IMarshaller	0	2	Pequeño
9	InvocationDataResult	2	0	Pequeño
10	ApplicationServer	0	2	Pequeño
11	LifeCycleManagerRegistry	0	2	Pequeño
12	ILifeCycleManager	0	2	Pequeño
13	ServerRequestHandler	0	3	Pequeño
14	InvokerRegistry	0	2	Pequeño
15	Message	4	0	Pequeño
16	ClientRequestHandler	0	3	Pequeño
17	ClientProxyRegistry	0	2	Pequeño
18	StaticLifeCycleManager	0	2	Pequeño

Tabla 8 Tamaño Operacional general del diseño

Se usó la métrica en un total de 18 clases, donde se obtuvo un promedio de 0.61 atributos y de 1.78 operaciones por clase, aproximadamente. Finalmente el resultado de la métrica se agrupa en la siguiente tabla:

Atributos	Alta	Media	Baja
Responsabilidad	0%	61%	39%
Complejidad de implementación	0%	61%	39%
Reutilización	54%	31%	15%

Tabla 9 Resultado de la aplicación de la métrica TOC

Luego de aplicada la métrica se obtuvo como resultado que las clases del diseño del sistema son pequeñas en su totalidad. Esto muestra que las mismas no tienen grandes responsabilidades, por lo que permite la reutilización de clases demostrando que el sistema tendrá una fácil implementación. Se puede afirmar entonces que los resultados obtenidos por la métrica son buenos.

Acoplamiento entre clases

Esta métrica es un indicador del esfuerzo necesario para el mantenimiento según los autores Chidamber y Kemerer. Los mismos sugieren que cuanto más acoplamiento existe en una clase, más difícil será su reutilización. Además las clases con exceso de acoplamiento dificultan la comprensión y hacen más complicado el mantenimiento, por lo que será necesario un mayor esfuerzo y riguroso testeo.

Las clases deberían ser lo más independientes posible, y aunque siempre es necesaria la dependencia entre clases, cuando las mismas son grandes, la reutilización puede ser más costosa que la reescritura. Al reducir el acoplamiento, se reduce la complejidad, se mejora la modularidad y se promueve la encapsulación. **(Chidamber, y otros, 1997)**

Esta métrica evalúa los atributos de calidad siguientes:

- Acoplamiento: Responsabilidad asignada a una clase en un marco de modelado de un dominio o concepto de la problemática propuesta.
- Complejidad de mantenimiento: Grado de esfuerzo a realizar para desarrollar una mejora o rectificación de algún error del diseño de un *software*. Puede influir indirecta, pero fuertemente en los costes y la planificación del proyecto.

- Reutilización: Grado de reutilización presente en una clase o estructura de clase dentro de un diseño de software.
- Cantidad de pruebas: Número o el grado de esfuerzo necesario para realizar las pruebas de calidad del producto (clase, componente) diseñado.

Los mismos están definidos por los criterios y categorías de evaluación que se muestran en la tabla siguiente:

Atributo	**Categoría**	**Criterio**
Acoplamiento	Ninguno	0
	Bajo	1
	Medio	2
	Alto	>2
Complejidad de mantenimiento	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio
Reutilización	Baja	>2*Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	<= Promedio
Cantidad de pruebas	Baja	<= Promedio
	Media	Entre Promedio y 2*Promedio
	Alta	>2*Promedio

Tabla 10 Criterios de evaluación de la métrica RC

A continuación se muestran los resultados obtenidos una vez que fue aplicada esta métrica al diseño del ORB.

Criterio	**Categoría**	**Cantidad de clases**	**Promedio**
0 dependencias	Muy Bueno	5	28%
1 dependencia	Bueno	5	28%
2 dependencias	Regular	3	19.5%

3 dependencias	Malo	2	5%
>3 dependencias	Muy Malo	3	19.5%
Total			100

Tabla 11 Criterios y categorías obtenidos en la aplicación de la métrica

Atributos	**Alta**	**Media**	**Baja**
Acoplamiento	28%	16%	56%
Complejidad de mantenimiento	17%	28%	55%
Cantidad de pruebas	17%	28%	55%
Reutilización	55%	28%	17%

Tabla 12 Resultados obtenidos de la aplicación de la métrica RC

Se concluye que el diseño propuesto está entre los parámetros aceptables de calidad del software. Los atributos se encuentran en niveles satisfactorios: el acoplamiento es bajo y la complejidad de pruebas, cantidad de pruebas y reutilización se comportan favorablemente con un 55%.

Conclusiones

Con el desarrollo de este capítulo se logró la descripción de la arquitectura del sistema, lográndose una propuesta certera para el mismo. La arquitectura Cliente/Servidor por Capas facilitará la distribución de los roles dentro del sistema, destacando el cliente como el elemento solicitante y el servidor como elemento proveedor. Además se determinaron las clases del Análisis y del Diseño, así como sus relaciones y otros diagramas como los de Colaboración y Secuencia, que proveen una vista general del sistema a implementar posteriormente y su flujo básico de eventos.

Todos los diagramas incluidos en el capítulo se presentan como una guía de sencilla comprensión para los futuros desarrolladores, probadores y encargados del mantenimiento del sistema.

El uso de métricas para la validación del diseño obtenido fue muy útil debido a que demostró que se trata de un diseño confiable que cumple los requisitos para su posterior implementación.

Este capítulo da cumplimiento a estos 3 objetivos específicos:

- Diseñar un intermediario que cumpla con las necesidades de SURIA para el manejo de tipos de datos simples (*string, integer, double, char*).
- Generar los artefactos necesarios que determinan el proceso de desarrollo del ORB para su futura implementación.
- Validar el diseño creado.

CONCLUSIONES GENERALES

Con el desarrollo del presente trabajo de diploma, se logró cumplir con los objetivos específicos trazados, arribándose a las conclusiones siguientes:

Al realizarse el análisis de las diferentes tecnologías que implementan un ORB, se presenciaron algunas limitantes para su implementación en un proyecto como SURIA. Estas limitantes estuvieron enmarcadas en el uso de licencias privativas, en el caso de .NET *Remoting*, licencias GPL, en el caso de ICE, o la decadencia del producto y su complejidad, como es el caso de CORBA. Sin embargo las características de estos sistemas se tuvieron en cuenta para el diseño del ORB para SURIA.

Se analizó la arquitectura de SURIA, concretándose en una arquitectura funcional para el ORB que garantizará la comunicación entre los diferentes módulos del sistema SURIA. Se obtuvo un Modelo del Dominio que permitirá a los implementadores entender el sistema a modo general. Se analizaron los requerimientos funcionales y no funcionales que debe cumplir el ORB y se llevó a un diagrama de Casos de Uso que sirvió para modelar el sistema a diseñar de una manera fácil de entender por los futuros desarrolladores del mismo. Se generaron todos los artefactos del Flujo de Análisis y Diseño de RUP, obteniéndose diagramas de Clases del Análisis, Colaboración, Clases del Diseño y Secuencia, para dar una visión del sistema a implementar posteriormente y además proveer a los desarrolladores del diseño que facilitará su trabajo en la implementación. Se validó el diseño obteniendo resultados satisfactorios.

En forma general se dio cumplimiento al objetivo general planteado ya que se logró el diseño de un ORB que permite la interoperabilidad entre los componentes de SURIA Vision desarrollados bajo la tecnología Qt y el lenguaje C++ en aplicaciones distribuidas.

Realizar el diseño del ORB en tecnología Qt y lenguaje C++ es el aporte científico del trabajo de diplomado, ya que anteriormente no se había desarrollado un intermediario bajo esta tecnología y lenguaje de programación, garantizándose así un sistema libre, pero que se ajusta a las necesidades de SURIA Vision. El sistema diseñado constituye una solución de vanguardia que garantiza el desarrollo de un ORB de sencilla estructura comparado con CORBA, por citar un ejemplo. Incluye las funcionalidades necesarias para lograr la comunicación dentro de un sistema distribuido y aplica tecnologías de desarrollo de software potentes como Qt y el lenguaje C++.

RECOMENDACIONES

Se recomienda para la futura implementación del ORB:

- Profundizar en el estudio de los diagramas correspondientes a la fase de Análisis y Diseño contenidos en los “Anexos” para lograr la implementación correcta del ORB, así como los patrones expuestos en la propuesta de solución aportada.
- Potenciar el aprovechamiento de las facilidades de implementación que ofrece el framework Qt en la implementación del sistema.
- Aplicar la tecnología desarrollada en otros contextos tanto fuera como dentro de la universidad, que tributen al perfeccionamiento de los sistemas distribuidos.

REFERENCIAS BIBLIOGRÁFICAS

Aylagas Romero, Francisco. 2004. **Universidad Politécnica de Madrid.** ***Universidad Politécnica de Madrid.*** **[En línea] 25 de marzo de 2004. [Citado el: 12 de diciembre de 2012.] http://www.dia.eui.upm.es/asignatu/sis_dis/Paco/Introduccion.pdf.**

Basanta Gutiérrez, David y Tajes Martínez, Lourdes. 1999. ***Tecnologías para el desarrollo de Sistemas Distribuidos: Java versus Corba.*** **Murcia : s.n., 1999.**

base, LSU GROK knowledge. 2012. **LSU GROK knowledge base.** ***LSU GROK knowledge base.*** **[En línea] 11 de diciembre de 2012. [Citado el: 20 de febrero de 2013.] http://grok.lsu.edu/article.aspx?articleid=4116.**

Birney, Ewan, y otros. 2003. ***ORBit Beginners Documentation V1.6.*** **2003.**

Calvo Gordillo, Isidro. 2007. ***Seminario de middleware, Grupo de Control e Integración de Sistemas (GCIS).*** **Vitoria-Gasteiz : s.n., 2007.**

Castro Fernández, Raúl. 2010. ***Desarrollo de software de videovigilancia para sistemas embarcados distribuidos con ICE.*** **Madrid : Universidad Carlos III de Madrid. Departamento de Ingeniería Telemática, 2010.**

Cerami, Ethan. 2002. ***Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL First Edition.*** **s.l. : O'Reilly, 2002. ISBN: 0-596-00224-6.**

—. 2002. ***Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL First Edition.*** **s.l. : O'Reilly, 2002. ISBN: 0-596-00224-6.**

Chand, Mahesh. 2007. **C# Corner.** ***C# Corner.*** **[En línea] .NET Remoting versus Web Services, 3 de octubre de 2007. [Citado el: 6 de junio de 2013.] http://www.c-sharpcorner.com/blogs/415/net-remoting-versus-web-services.aspx.**

Chidamber, S. R. y Kemerer, C. F. 1997. ***A metric suite force.*** **s.l. : IEEE Transactions on Software Engineering, 1997.**

Cobo Romaní, Juan Cristóbal. 2009. **El concepto de tecnologías de la información. Benchmarking sobre las definiciones de las TIC en la sociedad del conocimiento.** ***ZER- Revista de Estudios de Comunicación.*** **[En línea] 22 de septiembre de 2009. [Citado el: 24 de mayo de 2013.] http://www.ehu.es/ojs/index.php/Zer/article/view/2636/2182. ISSN: 1137-1102.**

Comunidad Postgres Universidad de las Ciencias Informáticas. 2010. **PostgreSQL Cuba. [En línea] 2010. [Citado el: 26 de Febrero de 2013.] http://postgresql.uci.cu/node/63.**

Cooper, James W. 2002. ***Introduction to Desing Patterns and C#.*** **2002.**

Cooper, Tim. 2010. **stackOverflow.** ***What is the beneffit of XML-RPC over plain XML?*** **[En línea] 14 de septiembre de 2010. [Citado el: 12 de mayo de 2013.] http://stackoverflow.com/questions/1376688/what-is-the-benefit-of-xml-rpc-over-plain-xml.**

Cuesta Rodríguez, Saúl. 2008. **SG. Patrones de casos de uso. [En línea] 2008. [Citado el: 2 de Marzo de 2013.]**

Curiel, Mariela. 2009. ***Sistemas Basados en Objetos Distribuidos. CORBA: un caso de estudio.*** **2009.**

DATYS, Desarrollo de Aplicaciones, Tecnologías y Sistemas. 2011. ***XYMA Save Vision Sistema de Cisrcuito Cerrado de Televisión Especificaciones Técnicas V 2.8.5.*** **La Habana : s.n., 2011.**

Dedov, Anton. 2013. **Libiqxmlrpc Free, XML-RPC C++ lib.** ***Libiqxmlrpc Free, XML-RPC C++ lib.*** **[En línea] 29 de mayo de 2013. [Citado el: 6 de junio de 2013.] http://libiqxmlrpc.wikidot.com/.**

Departamento de Sistemas Informáticos y Computación. Universidad de Valencia. ***Rational Unified Process (RUP).***

Doctrine Corporation. **Doctrine. [En línea] [Citado el: 5 de Enero de 2013.] http://www.doctrine-project.org/.**

2011. **DotNet Information.** ***.NET Remoting versus Web Services.*** **[En línea] enero de 2011. http://challagullainfo.blogspot.com/2011/01/net-remoting-versus-web-services.html.**

Doxygen. 2003. **QuteXR Qt based XML-RPC library.** ***QuteXR Qt based XML-RPC library.*** **[En línea] 25 de febrero de 2003. [Citado el: 12 de mayo de 2013.] http://qutexr.sourceforge.net/.**

Drake, José M. 2008. ***Sistemas distribuidos de tiempo real. VIII.1: CORBA Estándar para objetos distribuidos.*** **2008.**

G. Booch, J. Rumbaugh y I. Jacobson,. 2000. ***"El Lenguaje Unificado de Modelado".*** **Madrid : Addison Wesley (Edición en español por la Pearson Educación S.A.)., 2000. ISBN: 84-7829-036-2.**

García Sánchez, Felipe, y otros. 2005. **Tecnologías Middleware: soluciones actuales para grandes empesas y proyectos.** ***Repositorio Digital Universidad Politécnica de Cartagena.*** **[En línea] 2005. [Citado el: 10 de diciembre de 2012.] http://repositorio.bib.upct.es/dspace/handle/10317/349.**

Garrido Fuentes, José Carlos. 2011. **Teoría Programación Avanzada UCO 2011. [En línea] 26 de octubre de 2011. [Citado el: 20 de noviembre de 2012.] http://www.sage-technologies.net/corba/Orbarch.htm.**

Grimes, Fergal. 2002. ***Microsoft .NET for programmers.*** **Greenwich : Manning, 2002. ISBN.**

Grupo Soluciones Innova. **Grupo Soluciones Innova. [En línea] [Citado el: 16 de Mayo de 2013.] http://www.rational.com.ar/herramientas/rup.html.**

INEI. 1999. ***"Herramientas Case: COLECCION CULTURA INFORMATICA".*** **s.l. : Talleres de la Oficina de Impresiones de la Oficina Técnica de Difusión del Instituto Nacional de Estadística e Informática (INEI), 1999.**

Jacobson, Ivar; Booch, Grady; Rumbaugh, James. 2000. ***EL PROCESO UNIFICADO DE DESARROLLO DE SOFTWARE.*** **Madrid : Addison Wesley, 2000. ISBN: 84-7829-036-2.**

Kidd, Erick. 2001. **XML-RPC HOWTO.** ***XML-RPC HOWTO.*** **[En línea] Source Builders, 12 de abril de 2001. [Citado el: 16 de mayo de 2013.] http://tldp.org/HOWTO/XML-RPC-HOWTO/index.html.**

Kwiatkowski, Jan y Przewoźny, Maciej, Cunha, José Cardoso. 2001. **Arquitectura de Sistemas Computacionales Facultad de Ciencias y Tecnologías Universidad Nova de Lisboa.** ***Arquitectura de Sistemas Computacionales Facultad de Ciencias y Tecnologías Universidad Nova de Lisboa.*** **[En línea] 18 de junio de 2001. [Citado el: 12 de mayo de 2013.] http://asc.di.fct.unl.pt/~jcc/pub/2000/PORTO15.pdf.**

Larman, Craig. 2002. ***UML y Patrones Una introducción al análisis y diseño orientado a objetos y al proceso unificado 2da Edición.*** **Madrid : Prentice Hall, 2002. ISBN: 9788420534381.**

León, Sergio. 2010. **Programación desordenada.** ***Programación desordenada.*** **[En línea] 16 de noviembre de 2010. [Citado el: 5 de junio de 2013.] http://panicoenlaxbox.blogspot.com/2010/11/serializacion-en-net.html.**
LexJurídica. 2001-2013. **LexJurídica. Diccionario Jurídico. [En línea] 2001-2013. http://www.lexjuridica.com/diccionario.php.**
Lornel A. Rivas, María Pérez, Luis E. Mendoza, y Anna Grimán. 2007. ***"Herramientas de Desarrollo de Software: Hacia la Construcción de una Ontología".*** **Venezuela : s.n., 2007.**
Martínez Barrera, Crisman. 2002. ***Tecnología CORBA (Common Object Request Broker Architecture).*** **Bogotá : s.n., 2002.**
Martínez, Rafael. 2010. **PostgreSQL- es. [En línea] 2 de Octubre de 2010. [Citado el: 24 de Febrero de 2013.] http://www.postgresql.org.es/sobre_postgresql.**
Mestras, Juan Pavón. 2009. **El patrón Modelo-Vista-Controlador (MVC). [En línea] 2009. [Citado el: 26 de Enero de 2013.] http://www.fdi.ucm.es/profesor/jpavon/poo/2.14.mvc.pdf.**
Puder, Arno. 2004. ***MICO: An open source CORBA implementation.*** **San Francisco : IEEE Computer Society, 2004. ISBN 0740-7459/ 04.**
Microsoft. 2012. **[En línea] 2012. [Citado el: 14 de diciembre de 2012.] http://msdn.microsoft.com/es-es/library/aa289840%28v=vs.71%29.aspx.**
Milardovich, Sergio, y otros. 2005. **WEB NOVA recursos para webmaster.** ***WEB NOVA recursos para webmaster.*** **[En línea] 21 de octubre de 2005. [Citado el: 12 de mayo de 2013.] http://www.webnova.com.ar/articulo.php?recurso=426.**
Ministerio Público de Venezuela. **Ministerio Público. República Bolivariana de Venezuela. [En línea] [Citado el: 24 de Noviembre de 2012.] http://www.ministeriopublico.gob.ve/web/guest/169.**
Muñoz Peña, Mailin. 2011. ***Diseño de la Arquitectura de Software para el Sistema de Captura y Catalogación de Medias.*** **La Habana : s.n., 2011.**
Negri, Esteban. 2005. **PreguntasLinux.** ***PreguntasLinux.*** **[En línea] 19 de julio de 2005. http://preguntaslinux.org/que-es-la-licencia-gpl-t-510.html.**
NETWORK WORLD, Inc. 2006. **networkworld.** ***networkworld.*** **[En línea] 2006. http://www.networkworld.com/details/775.html.**
Oberle, Daniel. 2006. ***Semantic Managment of Middleware.*** **Karlsruhe : Springer, 2006. ISBN.**
Oracle Corporation. 2013. **NetBeans. [En línea] 2013. [Citado el: 16 de Febrero de 2013.] http://netbeans.org/community/releases/73/.**
Pacheco, Nacho. 2011. ***Doctrine 2 ORM Documentation.*** **2011.**
—. 2011. ***Manual de Twig.*** **2011.**
Patrones de Gand of Four. 1995. ***Patrones de Gand of Four.*** **1995.**
Pérez, Yamile y Cruz, Yusmila. 2010. ***Análisis y Diseño del Proceso Quejas y Reclamaciones.*** **2010.**
Ponjuan Dante, Gloria. 1998. ***Gestión de la Información en las organizaciones.*** **1998. ISBN: 956-7782-00-8 .**
Pressman, Roger S. 2010. ***"Software Engineering".*** **7ma. New York : Higher Education, 2010.**
—. 5ta Edición. ***Ingeniería de Software. Un enforque práctico.*** **5ta Edición.**

—. 2005. ***Ingeniería de Software. Un enforque práctico 5ta Edición.*** **2005.**
—. 2010. ***Software Engineering.*** **New York : Higher Education, 2010.**
Productiva, Infreastructura. 2012. ***Arquitectura de Software V2.0 Video Vigilancia SIV 1.0.*** **La Habana : s.n., 2012.**
Puigdemunt, Eduard. **Programación en castellano. [En línea] [Citado el: 23 de Marzo de 2013.] http://www.programacion.com/articulo/curso_de_xml_164/21#namespace-definicion-declaracion.**
Qusay H. Mahmoud. 2004. ***Middleware for Communication.*** **Guelph, Canadá : John Wiley & Sons, Ltd., 2004. ISBN 0-470-86206-8.**
Redmine. 2011. **Paquete de Gestion de Proyectos (GESPRO 12.05).** ***Paquete de Gestion de Proyectos (GESPRO 12.05).*** **[En línea] 2011. [Citado el: 5 de febrero de 2013.] http://gespro.geysed.prod.uci.cu/projects/video-vigilancia.**
Reynoso, Carlos y Kicillof, Nicolás. 2004. ***Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*** **Buenos Aires : s.n., 2004.**
Rodríguez Archilla, Pablo. 2010. ***Web services y arquitecturas orientadas al servicio.*** **Granada : s.n., 2010.**
Rodríguez Sánchez, David. 2011. ***Diseño de herramienta software para la automatización de pruebas de Caja Blanca al departamento de Señales Digitales del centro GEySED.*** **La Habana : s.n., 2011.**
Rouaux, Martín. 2005. ***Diseño y prototipo de middleware basado en CORBA para el bus can.*** **Buenos Aires : s.n., 2005.**
S. Pressman, Rogers. 2000. ***Aprendiendo UML en 24 horas.*** **2000.**
S. R., Chidamber y C. F, Kemerer. 1994. ***"A metric suite forsz".*** **s.l. : IEEE Transactions on Software Engineering, 1994.**
Saffirio, Mario. 2006. **Tecnologías de Información y Gestión de Procesos de Negocios (BPM). [En línea] 5 de febrero de 2006. http://msaffirio.wordpress.com/2006/02/05/%C2%BFque-son-los-web-services.**
Sami, Abdul. 2010. **CODE Project.** ***CODE Project.*** **[En línea] WCF Comparison with Web Servies and .NET Remoting, 9 de enero de 2010. [Citado el: 6 de junio de 2013.] http://www.codeproject.com/Articles/45698/WCF-Comparison-with-Web-Services-and-NET-Remoting.**
Schmidt, Douglas. 2006. **TAO DOC Group.** ***TAO DOC Group.*** **[En línea] 9 de octubre de 2006. [Citado el: 6 de junio de 2013.] http://www.cs.wustl.edu/~schmidt/TAO.html.**
Semanat Aldana, Edmis Deivis y Verdecia Four, Leonor. 2009. ***Sistema de Video Vigilancia.*** **La Habana : Universidad de las Ciencias Informáticas, 2009.**
Solís Portillo, Daniel. 2010. ***Software libre (fuentes abiertas).*** **Madrid : Universidad Carlos III, 2010.**
Sommerville, Ian. 7ma Edición. ***Ingeniería de Software.*** **7ma Edición.**
Stallings, William. 1997. ***Sistemas Operativos segunda edición.*** **Madrid : Prentice Hall, 1997. ISBN: 84-89660-22-0.**
Tanenbaum, Andrew S. 2005. ***Sistemas Distribuidos Modernos, segunda edición.*** **Santiago de Cuba : Impreso en la Empresa Gráfica Haydee Santamaría, Palma Soriano, 2005.**

—. 2009. ***Sistemas Operativos Modernos Tercera Edición.*** **México : PEARSON Prentice Hall, 2009. ISBN: 978-607-442-046-3.**

Tanenbaum, Andrew S. y Van Steen, Maarten. 2007. ***Distributed Systems. Principles and Paradigms, second edition.*** **Amsterdam : Pearson Prentice Hall, 2007.**

Tomlinson, Todd y VanDyk, John K. 2010. ***Pro Drupal 7 Development Third Edition.*** **New York : Apress, 2010. ISBN-13 (electronic): 978-1-4302-2839-4.**

UCI. 2008-2009. ***Conferencia: "Metodologías_de desarrollo_del_Sw".*** **La Habana, Cuba : s.n., 2008-2009.**

Valdivieso, Daniel. 2007. **Ingeniería web.** ***Ingeniería web.*** **[En línea] 14 de junio de 2007. http://sistemas3.wordpress.com/2007/06/14/web-services/.**

Vanhooff, Bert, Preuveneers, Davy y Berbers, Yolande. 2006. ***.NET Remoting and Web Services: A Lightweight Bridge between the .NET Compact and Full Framework.*** **Zurich : ETH, 2006.**

Volter, Markus; Kircher, Michael; Zdun, Uwe. 2005. ***Remoting Patterns. Foundations of Enterprise, Internet and Realtime Distributed Object Middleware.*** **Chichester : John Wiley and Sons, Ltd, 2005. ISBN.**

Wiedenroth, Sebastian. 2008. **wiedi in WONDERLAND.** ***wiedi in WONDERLAND.*** **[En línea] 8 de marzo de 2008. [Citado el: 12 de mayo de 2013.] http://wiedi.frubar.net/blog/2008/03/08/libmaia-xml-rpc-with-qt4/.**

ZeroC. 2012. **Diferences between ICE and CORBA.** ***Diferences between ICE and CORBA.*** **[En línea] ZeroC, Inc., 10 de septiembre de 2012. [Citado el: 2013 de mayo de 17.] http://www.zeroc.com/iceVsCorba.html.**

BIBLIOGRAFÍA

Buschmann, Frank, y otros. 2001. ***Pattern-Oriented Software Architecture A System of Patterns Volume 1.*** **New York : John Wiley & Sons, Inc., 2001. ISBN 0 471 95869 7.**

Dougherty, Chad y Sayre, Kirk, Seacor, Robert C., Svoboda, David, Togashi, Kazuya. 2009. ***Secure Design Patterns.*** **2009.**

Ferrer Mesa, Silenay y de Francisco Hidalgo, Michel Raúl. 2008. ***Diseño de una aplicación para el monitoreo de señales de radio.*** **La Habana : Universidad de las Ciencias Informáticas, 2008.**

Guibert Nápoles, Rosalbis y Cárdenas del Valle, Yusniel. 2008. ***Mecanismos de seguridad para el Middleware de SCADA "Guardián del ALBA".*** **La Habana : Universidad de las Ciencias Informáticas, 2008.**

Hernández León, Rolanfo Alfredo y Coello González, Sayda. 2002. ***El paradigma cuantitativo de la investigación científica.*** **La Habana : EDUNIV Editorial Universitaria, 2002. ISBN: 959-16-0343-6.**

Larman, Craig. 1999. ***UML y Patrones.*** **México : Prentice Hall, 1999. ISBN: 970-17-0261-1.**

Marzo Arbona, Ana Mercedes y Ortiz Hernández, Maridilia. 2009. ***Monitoreo y control remoto de centros tecnológicos.*** **La Habana : Universida de las Ciencias Informáticas, 2009.**

Oberle, Daniel. 2006. ***Semantic Managment of Middleware.*** **New York : Springer, 2006. ISBN-10: 0 387 0 387 27630 0.**

Reyes Llorente. 2009. ***Propuesta de Integración de Aplicaciones y Servicios ocn ESB.*** **La Habana : Centro Universitario José Antonio Echeverría, 2009.**

Schimdt, Douglas, y otros. 2000. ***Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects, Volume 2.*** **New York : John Wiley & Sons, Inc., 2000. ISBN 0471606952.**

Sturm, Jack. ***Desarrollo de soluciones XML.*** **s.l. : Mac Graw Hill interamericana.**

Talia, Domenico, Ziegler, Wolfgang y Ramin, Yahyapour. 2008. ***Grid Middleware And Services, Challenges And Solutions.*** **New York : Springer, 2008. ISBN 978 0 387 78445 8.**

Tanenbaum, Andrew S. y Van Steen, Maarten. 2007. ***Distributed Systems, Principles And Paradigms.*** **New York : Pearson Prentice Hall, 2007. ISBN: 0 13 239 227 5.**

Volter, Markus y Kircher, Michael, Zdun, Uwe. 2005. ***Remoting Patterns, Foundations of Enterprise, Internet and Realtime Distributed Object Middleware.*** **New York : John Wiley and Sons, Ltd, 2005. ISBN 0 470 85662 9.**

Zúñiga Ferreiro, Milene y Tomé Ulloa, Ariamna. 2011. ***Informe técnico sobre la integración de información.*** **La Habana : CITI, 2011.**

GLOSARIO DE TÉRMINOS

Boost: Provee librerías libres para el lenguaje C++. Su licencia aplica a fines comerciales y no comerciales.

GNU GPL: La Licencia Pública General de GNU o más conocida por su nombre en inglés GNU General Public License (o simplemente sus siglas del inglés GNU GPL) es la licencia más ampliamente usada en el mundo del software y garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.

GNU LGPL: La Licencia Pública General Reducida de GNU, o más conocida por su nombre en inglés *GNU Lesser General Public License* (antes *GNU Library General Public License* o Licencia Pública General para Bibliotecas de GNU), o simplemente por su acrónimo del inglés GNU LGPL, es una licencia de software creada por la Free Software Foundation que pretende garantizar la libertad de compartir y modificar el software cubierto por ella, asegurando que el software es libre para todos sus usuarios. Esta licencia permite a los desarrolladores guardarse para sí algunas características del programa como su código fuente, en caso de que sea una aplicación para comercializar.

GoF: Los patrones GoF son conocidos como los patrones de la pandilla de los cuatro (GoF, gang of four). Se clasifican según su propósito en: Patrones de Creación, Patrones Estructurales y Patrones de Comportamiento.

Libxml2: Es el analizador y conjunto de herramientas de XML para el lenguaje C, desarrollado por el proyecto Gnome. Es software libre.

OpenSSL: Es una herramienta que usa los protocolos *Secure Socket Layer* (SSL v2/v3) y *Transport Layer Security* (TLS v1) para la comunicación. Tiene una licencia que permite su uso con propósitos tanto comerciales como no comerciales.

Tag: Es un comando que va insertado en un documento, cuyo fin es especificar cómo el documento o parte de él debe ser formateado o interpretado.

W3C: Abreviación de *World Wide Web* Consortium, es un consorcio de empresas involucradas con la Internet y la Web. La W3C fue fundada en 1994 por Tim Berners-Lee, el arquitecto de la *World Wide Web*.

Los fines de esta organización son los desarrollos de estándares abiertos (que los pueda usar todo el mundo) de modo que la Web evolucione coherentemente.

Signals and Slots: Son usadas para la comunicación entre objetos. Las *Signals* (Señales), son emitidas por un objeto cuando su estado interno ha cambiado de alguna forma que interesa al propietario o cliente del objeto. Las *Slots* (Ranuras), son activadas cuando una señal conectada a ella es emitida.

Multiplexación: En un extremo, los multiplexores son equipos que reciben varias secuencias de datos de baja velocidad y las transforman en una única secuencia de datos de alta velocidad, que se transmiten hacia un lugar remoto. En dicho lugar, otro multiplexor realiza la operación inversa obteniendo de nuevo los flujos de datos de baja velocidad originales. A esta función se la denomina demultiplexar.