UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 4

TÍTULO:

ARQUITECTURA DE LA HERRAMIENTA ANDRÓMEDA

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

Autores: Pablo Molina Toledo

Henry Rodríguez Estévez

Tutores: MSc. Ismael Armando Nodarse Mora

Ing. Enrique José Altuna Castillo

Ciudad de La Habana, Cuba, junio 2012

"Año 54 de la Revolución"

Declaración de Autoría

Declaramos que somos los únicos autores del trabajo “Arquitectura de la herramienta Andrómeda” y autorizamos a la Facultad 4 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Autor: Pablo Molina Toledo	**Tutor:** MSc. Ismael A. Nodarse Mora
Autor: Henry Rodríguez Estévez	**Tutor:** Ing. Enrique José Altuna Castillo

A nuestros padres por encaminarnos a buscar la libertad

e impulsarnos a superarnos profesionalmente,

a toda nuestra familia y amigos que siempre han estado presente.

"Si tú tienes una manzana y yo tengo una manzana y las intercambiamos, entonces ambos aún tendremos una manzana. Pero si tú tienes una idea y yo tengo una idea y las intercambiamos, entonces ambos tendremos dos ideas."

George Bernard Shaw

Agradecimientos

Pablo

A mis padres, que gracias a ellos hoy su primogénito se les convierte en ingeniero. Por ser ejemplos a seguir en mi vida. Por señalarme el buen camino en todo momento, pero dejarme elegirlo por mí mismo. Por hacerme sentir orgulloso cada vez que hablo de ellos.

A mis abuelas que han sido mis segundas madres y a las que les debo la vida.

A mi tía Baby que tanto me ha enseñado y cuidado, y la que le debo gran parte de mí. Ella sabe que me siento orgulloso de tenerla a mi lado.

A mi primo Roly, que más que mi primo ha sido el hermano mayor que siempre quise tener. Por haber sido cómplice de que eligiera ser ingeniero informático. Ese que siempre estaba ahí para ayudarme y guiarme en mis años de universidad. Por ser esa persona que tienes como meta a seguir en tu vida.

A Guillermo Solenzal (el Guille), por haber confiado en mí cuando comenzó a darme clases y hacerme un espacio en su proyecto y su vida y criarme como un hijo para él. Por compartir tantas fiestas y consejos conmigo.

A todo mi gran grupo de amigos con los que he compartido mi vida en la universidad, el Guille, Sergio, Nilber, Maylín y Alexei, por ser mis primeros grandes amigos y a los cuales llevo conmigo siempre. A mi compañero de tesis, Álvarez, Leo

Al equipo de MENPET, que fue la primera gran familia que tuve y de la cual fui el más chico de todos y al equipo de Dolphin, donde conocí a mis últimos grandes hermanos de la universidad, “los mariscos” Jorge, Jose, Cao y el Yoe.

A mi clan de CDI con el que tan buenos momentos compartí al estar lejos de la patria y compañeros de fiestas y parrandas.

A todos los profesores que me han acompañado y guiado en estos años de universidad. Gracias a todos por compartir conmigo más que su conocimiento y ayudarme a ser mejor.

A mi tutor y compañero de equipo que tanto me ha apoyado.

A todo el que me faltó por mencionar más arriba y que me ayudó en esto. Son muchos los que han formado parte de estos 5 años.

A todos ellos, muchas gracias…

Henry

A mis padres, con la realización de esta tesis no solo se cumple un sueño, este es el resultado de años de sacrificio, del esfuerzo de mis padres; quienes me han dado de sí, lo mejor. A ellos les debo quien soy, gracias por confiar en mí, este trabajo también es de ustedes.

A mi familia toda por estar conmigo en cada paso de mi vida.

A mis amigos Álvarez (The Dog), Dayanis (The Fly), Pablo, Tony, Claudia, “el makina”, Porto, por todo ese calor de amistad que me han transmitido durante todo este tiempo, a Lombillo por su paciencia, dedicación y esmero para conmigo en estos 5 años (principalmente en el primero).

A mi tutor, “el Ismaelillo” por la tremenda dedicación y ayuda brindada para el desarrollo de esta investigación en conjunto con Vázquez, Osdalme y Jorge.

A Haydee, Marianne y Yuri, por la ayuda, objetividad y constructividad de sus señalamientos en el desarrollo de la investigación y por brindarme su amistad y confianza en el transcurso de la misma.

A Carmela, Sosa y Marilys por hacerme sentir parte de la familia y por darme la posibilidad de pasar unas tremendas vacaciones en Santiago de Cuba.

A Mayara por ayudarme con los nervios.

A Dosagues por ser un ejemplo a seguir como profesional y por haber contribuido grandemente en mi formación como investigador y como persona.

A los ¨mariscos¨ Jorge, Nilber, Yoennis, Jose y a la representación de los tiesos en el universo, a Cao, por compartir su amistad conmigo y hacer más llevadera y alegre estos 5 años de carrera.

Resumen

Con el incremento tecnológico de la actualidad el desarrollo de la tecnología inalámbrica ha alcanzado un auge significativo, llevando sus aplicaciones incluso a la automatización de edificios. El diseño del despliegue de una red de sensores inalámbricos en una edificación es una compleja tarea cualitativa y cuantitativamente en cuanto a tiempo y recursos.

El presente trabajo define la arquitectura de software para la aplicación Andrómeda, la cual tiene como objetivo fundamental la generación automática de propuestas de diseño de despliegue que puedan ser utilizadas dentro de las edificaciones. El proceso de definición y desarrollo de la arquitectura fue guiado por la metodología XP. La definición comprende los estilos arquitectónicos, patrones a emplear, las herramientas y tecnologías definidas para el desarrollo, así como la estructura básica de los módulos que se contemplan en la herramienta.

La arquitectura fue validada mediante la técnica de evaluación basada en escenarios, aplicando el método MECABIC y usando como instrumento el Árbol de Utilidades, permitiendo identificar tanto los puntos de riesgos, como las fortalezas y debilidades de la misma. Con la realización de la evaluación se puede constatar la eficiencia del trabajo realizado y la obtención de una arquitectura óptima.

Palabras clave: arquitectura, diseño, MECABIC, programación, sensor

Índice

Índice de figuras

Índice de tablas

Introducción

Una Red Inalámbrica de Sensores y Actuadores (*Wireless Sensor and Actuator Networks*, WSAN por sus siglas en Inglés y en lo adelante WSAN), es una red de pequeños computadores (nodos), equipados con sensores que trabajan con un fin común, ya sea en el monitoreo y/o control. Están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación inalámbrica los cuales permiten formar redes ad hoc sin infraestructura física preestablecida ni administración central conectada a un sistema central. [1]

En estas redes, además de nodos sensores existen nodos actuadores. Los sensores van reuniendo información sobre el medio físico, mientras que los actuadores toman decisiones y ejecutan las acciones apropiadas sobre el entorno.

La evolución de redes de sensores tiene su origen, como casi todos los proyectos científicos, en el campo militar. Se conoce que la investigación en redes de sensores comenzó cerca de 1980 con el proyecto *Distributed Sensor Networks* (Redes de Sensores Distribuidos, DSN por sus siglas en Inglés) de la agencia militar de investigación avanzada de Estados Unidos, *Defense Advanced Research Projects Agency* (Agencia de Proyectos de Investigación de Defensa Avanzada, DARPA por sus siglas en Inglés). [1]

Las investigaciones fuera del ámbito militar datan de finales de los 90, con el proyecto *Smartdust*[1], ubicado en Berkeley, Estados Unidos. Los investigadores participantes en estos estudios, apodaron con el término "*mote*" (mota) a los nodos sensores [1]. Estos nodos sensores disponen, tradicionalmente, de una capacidad de cómputo y de almacenamiento muy pequeña en comparación con los sistemas electrónicos normales. Esto se debe a la necesidad de disminuir al máximo el consumo de energía para prolongar la vida de las baterías.

Las WSAN pueden ser utilizadas en un gran número de aplicaciones. En la actualidad, se encuentran desplegadas en aplicaciones agrícolas, militares, ambientales, para la salud y la construcción civil. [2-4]

[1] http://robotics.eecs.berkeley.edu/~pister/SmartDust/

Son usadas además en la automatización de hogares y edificios inteligentes, tecnología que ha alcanzado un vertiginoso desarrollo en los últimos años. [5]

La presente investigación está centrada en las aplicaciones de las WSAN en los Sistemas de Control de Edificios (en lo adelante SCE), y cómo desarrollar el diseño del despliegue de redes de sensores para los mismos.

Su escasa utilización en algunos tipos de escenarios puede conllevar al surgimiento de complejos problemas. Este es el caso de la difícil tarea de desplegar una WSAN dentro de una edificación. El campo eléctrico dentro de un edificio está integrado por un amplio número de componentes indirectos, factores considerables que no se tienen en cuenta en el caso del diseño de redes para entornos abiertos. Esta es una de las razones por las cuales, en entornos de interior los niveles de la señal fluctúan en mayor medida que en entornos exteriores y por consiguiente, la cobertura inalámbrica se caracteriza por ser compleja y muchas veces puede resultar incontrolable.

En la actualidad existen varias investigaciones que enfrentan el problema antes mencionado. Algunas soluciones se basan en el empleo de herramientas software que faciliten, de forma efectiva, el despliegue de sensores en el interior de edificaciones. A pesar de que estas propuestas ofrecen relevantes soluciones en este ámbito de exploración, aún existen enfoques que no han sido abordados y deficiencias sin resolverse, los cuales representan campos de investigación abiertos a nuevas innovaciones.

Las principales deficiencias detectadas en los estudios realizados a las herramientas actuales son:

- Inadecuada selección de los modelos de propagación de la señal de radiofrecuencia (RF).
- En las simulaciones no se tiene en cuenta el tipo de material y los obstáculos presentes en las edificaciones.
- Solo tienen en cuenta la cobertura y la conectividad interna de la red a la hora de generar las propuestas de despliegue.
- Las topologías de red obtenidas no satisfacen los diseños requeridos por los SCE actuales.

Producto a las deficiencias anteriores, se hace necesario crear el diseño de una arquitectura de software, lo suficientemente flexible. Dicha arquitectura será empleada en una aplicación, la cual llevará el nombre Andrómeda, y que permitirá generar de forma automática, el número y la posición de los dispositivos de una WSAN, obteniendo así topologías de red de tipo estrella, árbol y malla que satisfagan el diseño de la

red. Esta herramienta, una vez desarrollada permitirá, entre otras cosas, que el diseñador de la red no necesite un elevado conocimiento de comunicaciones inalámbricas y diseño de WSAN, para poder realizar la difícil tarea de desplegar una red de sensores dentro de una edificación.

Producto de lo antes expresado, se plantea el siguiente ***problema a resolver***: ¿Cómo garantizar la organización funcional de la herramienta Andrómeda y las relaciones entre sus módulos y componentes?

El ***objeto de estudio*** de la investigación se centra en el área de la Arquitectura de Software. De aquí se deriva que el ***campo de acción*** sea específicamente la arquitectura de la herramienta Andrómeda.

Con el propósito de encontrar una solución al problema planteado se define como ***objetivo general***:

Definir una arquitectura de software haciendo uso de estilos y patrones arquitectónicos que permitan el desarrollo y mantenimiento de la herramienta Andrómeda, facilitando el diseño del despliegue de WSAN en entornos interiores.

Teniendo para ello los siguientes ***objetivos específicos***:

1. Seleccionar las herramientas y tecnologías que permitan la definición de la arquitectura de la herramienta Andrómeda.
2. Seleccionar una arquitectura que satisfaga los requerimientos de la herramienta Andrómeda.
3. Desarrollar la arquitectura en base a la metodología de desarrollo seleccionada.
4. Validar la arquitectura definida.

Para dar cumplimiento a los objetivos específicos y al objetivo general se han propuesto las siguientes tareas de investigación:

1. Análisis del estado del arte de herramientas similares a la aplicación a desarrollar, así como las tecnologías utilizadas.
2. Valoración de las metodologías, herramientas de desarrollo, *frameworks* y lenguajes a utilizar en el análisis, diseño e implementación de la herramienta.
3. Definición de las listas de reservas del producto y aspectos no funcionales del mismo.
4. Selección de los estilos y patrones arquitectónicos a utilizar en la herramienta a desarrollar.
5. Confección de la arquitectura candidata según las tecnologías y herramientas empleadas.
6. Desarrollo del diseño arquitectónico y las bases estructurales de los patrones seleccionados.
7. Validación de la arquitectura desarrollada.

Finalmente, para guiar el desarrollo de la solución se plantea la siguiente ***idea a defender***: Si se identifican y se seleccionan correctamente herramientas y tecnologías para la confección de una arquitectura de software, se establecen los requisitos para su posterior diseño, desarrollo y satisfactoria validación, entonces, se podrá desarrollar una herramienta de software que permita el diseño del despliegue de WSAN en entornos interiores, lo que posibilitará un incremento de la productividad y el ahorro considerable del tiempo de trabajo y recursos a la hora de desplegar una red de sensores dentro de una edificación.

El documento está organizado en capítulos, de la siguiente forma:

Capítulo 1: Se hace un análisis de los principales conceptos relacionados con el dominio y se realiza un estudio del estado del arte de las herramientas similares. Además, se analizan las distintas herramientas, tecnologías y metodologías a utilizar.

Capítulo 2: En este capítulo se hace alusión a las fases de Exploración y Planificación de la metodología XP aplicadas al desarrollo de la herramienta, así como la presentación de los artefactos que generan las mismas. Queda reflejada la estimación de tiempo de las iteraciones a desarrollar, así como las entregas al cliente del producto.

Capítulo 3: Se plantea cuál será la arquitectura seleccionada para el desarrollo del sistema. Se especifica cómo será la representación arquitectónica y se describe la arquitectura planteada por la metodología de desarrollo utilizada. Es aquí donde se describe la fase de Codificación en la que se detallan las Tareas de Ingeniería a llevar a cabo por los desarrolladores y los Diagramas de Clases que se tienen de la propuesta de solución.

Capítulo 4: Se valida la arquitectura utilizando la técnica basada en escenarios y aplicando el Método de Evaluación para Arquitecturas de Software Basadas en Componentes, MECABIC. Utilizándose como instrumento el Árbol de Utilidades, permitiendo identificar tanto los puntos de riesgos, los de no riesgos, los puntos sensibles y los *tradeoff* de la arquitectura.

Capítulo 1. Fundamentación teórica

1.1 Introducción

El objetivo de este capítulo es realizar un análisis de los principales conceptos relacionados con el dominio de las aplicaciones para el diseño de redes de sensores en los SCE. Además, se lleva a cabo un estudio de algunos sistemas similares existentes, vinculados al campo de acción. Finalmente, se describen las herramientas, tendencias de desarrollo de software y tecnologías a emplear en la creación de la propuesta.

1.2 Arquitectura de Software

1.2.1 Definición de Arquitectura de Software

Tratar de definir un término como la Arquitectura de Software es siempre una actividad compleja. Actualmente no existe una definición ampliamente aceptada por el Instituto de Ingeniería de Software[2] que la defina. Para comprender la diversidad de puntos de vista, dados a este concepto se analizarán algunas de las definiciones más admitidas en la industria del software.

Se procederá a analizar la definición más oficial que se tiene de Arquitectura de Software y que se ha adoptado por varios grupos de trabajo que lideran el mercado de software a nivel mundial (tal es el caso de Microsoft). Esta definición está recogida en el documento de la IEEE Std. 1471-2000, la cual plantea que:

“La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente; y los principios que orientan su diseño y evolución”. [6]

Esto sienta las bases para la comprensión de la disciplina. La arquitectura capta la estructura del sistema en términos de componentes y la forma en que interactúan. También define todo el sistema de reglas de diseño y considera cómo un sistema puede cambiar.

[2] http://www.sei.cmu.edu/architecture/start/glossary/community.cfm

Otro punto de vista a tener en cuenta, es el de algunos de los principales pensadores de este campo como son Len Bass, Paul Clements y Ken Bass, quienes plantean que:

“La Arquitectura de Software de un programa o sistema de computadora es la estructura del sistema, que comprende los elementos de software, las propiedades externamente visibles de esos elementos, y las relaciones entre ellos”. [7]

Esto se basa un tanto en la anterior definición dada por la IEEE, especialmente en lo que se refiere al papel de la abstracción en una arquitectura y de sus múltiples puntos de vista (estructuras del sistema). Por último es válido destacar una de las definiciones procedentes del influyente trabajo de David Garlan y Mary Shaw:

“La Arquitectura de Software va más allá de los algoritmos y estructuras de datos de la computación, diseño y especificación de la estructura global del sistema que emerge como un nuevo tipo de problema.

Está compuesta por cuestiones estructurales que incluyen el grueso de la organización y control global de la estructura; protocolos para la comunicación, la sincronización y el acceso a datos; asignación de funciones a los elementos de diseño; distribución física; la composición de elementos de diseño, escala y el rendimiento, y la selección entre las alternativas de diseño”. [8]

Todas estas definiciones tienen muchos puntos en común. En resumen todos se basan fundamentalmente en la composición del software y representación abstracta en pequeños subsistemas que se relacionan entre sí. Esta abstracción centra su atención en ciertas cuestiones, tales como la escalabilidad, la distribución, entre otras consideradas: metas de la arquitectura.

1.2.2 Historia de la Arquitectura de Software

Los antecedentes de la Arquitectura de Software se remontan a la década de 1960, su historia no ha sido tan continua como la del campo más amplio en el que se inscribe, la Ingeniería del Software. Entre los primeros científicos en hacer planteamientos que se acercaran a lo que hoy se conoce como Arquitectura de Software se destaca Edsger Dijkstra de la Universidad Tecnológica de Holanda en 1968, quien propuso que se hiciera una estructuración correcta de los sistemas de software antes de lanzarse a programar. Más tarde Fred Brooks Jr. y Ken Iverson, en 1969, llamaban arquitectura a la estructura conceptual de un sistema en la perspectiva del programador. En 1975 Brooks utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el

arquitecto es un agente del usuario, igual que lo es quien diseña su casa, empleando una nomenclatura que ya nadie aplica de ese modo. Contemporáneo a estos, otro precursor importante, David Parnas, demostró que los criterios seleccionados en la descomposición de un sistema impactan en la estructura de los programas y propuso diversos principios de diseño que debían seguirse a fin de obtener una estructura adecuada.

En la década de 1980 fueron perfeccionadas las técnicas descriptivas, las notaciones formales y para la caracterización de lo que sucedería en la siguiente década, quedó inscrita en la historia de la especialidad la siguiente frase: “La década de 1990, creemos, será la década de la Arquitectura de Software…”. Esta rama de la ciencia de la computación quedó en estado de vida latente durante unos cuantos años, hasta comenzar su expansión explosiva con los manifiestos de Dewayne Perry de AT&T Bell, Laboratorios de New Jersey y Alexander Wolf de la Universidad de Colorado. Puede decirse que Perry y Wolf fundaron la disciplina, puesto que el primer estudio en que aparece la expresión Arquitectura de Software como se conoce hoy fue realizado por ellos en 1992. [9]

En los últimos 15 años ha habido un gran aumento en el realce de este campo de la Ingeniería de Software. Actualmente se encuentra en una etapa de formación constante y están surgiendo nuevos aportes que desarrollan y amplían la disciplina.

1.2.3 Importancia de la Arquitectura de Software

La necesidad del manejo de la arquitectura de un sistema de software nace con los sistemas de mediana o gran envergadura, que se proponen como solución para un problema determinado [10]. Esto supone que entre mayor sea el tamaño de la solución mayor será el número de requisitos involucrados en el mismo.

La división de un sistema en módulos bien estructurados posibilita una comunicación mutua entre los desarrolladores de un software, utilizando la arquitectura como base para crear un entendimiento mutuo y comunicarse entre sí. Esto es sumamente importante para tomar futuras decisiones y formar un consenso común respecto al desarrollo.

Mediante la arquitectura se pueden tomar decisiones tempranas de diseño, sobre un sistema, lo cual tiene un peso importantísimo para la mitigación de riesgos potenciales, evitando que ocurran futuros desastres a gran escala.

La descripción arquitectónica proporciona diagramas que brindan una representación constructiva del sistema, indicando los componentes y las dependencias entre ellos, los cuales constituyen una guía para el desarrollo.

Un buen diseño arquitectónico promueve la reutilización a gran escala de una cantidad importante de componentes y *frameworks*. Esto reduce los costos de diseño y la cantidad de código se simplifica. [10]

Sin lugar a dudas, un correcto diseño arquitectónico posibilita la evolución del sistema, pues estima los posibles cambios y los costos de las modificaciones a las que se puede someter un sistema, permitiendo comprender el grado de mejoramiento que este puede alcanzar.

1.2.4 Definición de Estilos y Patrones Arquitectónicos

La diferencia entre estilos y patrones arquitectónicos no ha sido aclarada. Bengtsson [11] plantea la existencia de dos grandes vertientes, que surgen de la discusión de los términos. Shaw y Garlan [8] utilizan indistintamente los términos estilo arquitectónico y patrón arquitectónico. Por otro lado, Buschmann [12] establece diferencias sutiles entre ambos conceptos. De cualquier forma, los estilos y los patrones establecen un vocabulario común y brindan soporte a los ingenieros para conseguir una solución que haya sido aplicada con éxito anteriormente, ante ciertas situaciones de diseño. Además, su aplicación en el diseño de la arquitectura del sistema es determinante para la satisfacción de los requerimientos de calidad.

Estilo arquitectónico

Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural. En particular, según los autores, un estilo arquitectónico define tanto un vocabulario de tipos de componentes y conectores (como en el caso de filtros y tubos), como un conjunto de restricciones sobre cómo combinar esos componentes y conectores:

- Sirven para sintetizar estructuras de soluciones.
- Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas.
- Definen los patrones posibles de las aplicaciones.
- Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales. [6, 9]

Patrón arquitectónico

Buschmann [12] define patrón como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución. En líneas generales, un patrón sigue el siguiente esquema:

- Contexto. Es una situación de diseño en la que aparece un problema de diseño.
- Problema. Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
- Solución. Es una configuración que equilibra estas fuerzas, esta abarca:
  - Estructura con componentes y relaciones.
  - Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc.

El Anexo 1 muestra en forma de resumen el nivel de abstracción entre estas clasificaciones.

1.2.5 Estilos y patrones arquitectónicos

La dinámica incontenible de la producción de patrones en la práctica de la Arquitectura de Software ha desarrollado un auge incontenible y exhaustivo en los últimos tiempos. De esta forma se ha atenuado la idea de que los patrones constituyen uno de los paradigmas del diseño arquitectónico. Cada uno de estos posee una historia y una fundamentación distinta y presenta, como todas las cosas en este terreno, sus sesgos, sus beneficios y sus limitaciones.

Todo el mundo acepta que existen diversas clases de patrones y estilos. Cada autor que escribe sobre el asunto agrega una clase diferente, y los estándares en vigencia no hacen ningún esfuerzo para poner un límite a la proliferación de variedades y ejemplares.

A pesar de la diversidad de estilos y taxonomías para agrupar los mismos, aún no hay una definición estándar para su agrupación. Es por ello que para esta investigación se tratará de agrupar los estilos en conjuntos minimalistas que permitan su comprensión en lugar de aplicar una notación formal más elaborada. La notación se establecerá entonces en términos de lo que Shaw y Clements llaman “*boxology*” [13]. Demás está decir que la descripción de los estilos puede hacerse también en términos de Lenguajes Descriptivos de Arquitectura (ADLs por sus siglas en Inglés) y las respectivas herramientas que se asocian con ellos.

Los estilos que habrán de describirse a continuación no aspiran a ser todos los que se han propuesto, sino apenas los más representativos, vigentes y relacionados con el problema a resolver.

1.2.5.1 Estilos de llamada y retorno

Este grupo enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala.

Modelo – Vista – Controlador

Este estilo separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes. [14]

En él, el **modelo** administra el comportamiento y los datos del dominio de aplicación, responde a instrucciones de información sobre su estado o para cambiar el mismo, en dependencia de si las acciones fueron formuladas desde la **vista** o el **controlador**. Por su parte la **vista** maneja la visualización de la información, y el **controlador** interpreta las acciones o eventos, e informa al **modelo** y/o a la **vista** para que cambien según resulte apropiado.

El soporte de múltiples vistas y la adaptación al cambio figuran como las ventajas más significativas de este estilo. Sin embargo puede llegar a ser un tanto complejo, debido a que introduce nuevos niveles de indirección.

Arquitecturas en Capas

Garlan y Shaw definen el estilo **Arquitectura en Capas** como una organización jerárquica, tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior [9]. Las restricciones topológicas del estilo pueden incluir una limitación, más o menos rigurosa, que exige a cada capa operar sólo con capas adyacentes, y a los elementos de una capa entenderse sólo con otros elementos de la misma.

Las ventajas de dicho estilo son obvias. Primero que nada, soporta un diseño basado en niveles de abstracción crecientes, lo cual permite a los desarrolladores la división de un problema complejo en una secuencia de pasos incrementales. En segundo lugar, el estilo admite muy naturalmente optimizaciones y refinamientos. Por último, proporciona amplia reutilización. Su principal desventaja radica en que los cam-

bios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, si se utiliza una modalidad relajada.

El ejemplo más característico es el modelo OSI con sus siete niveles: nivel físico, vínculo de datos, red, transporte, sesión, presentación y aplicación; en cuyas capas se provee de servicios a la capa superior.

Programa principal y subrutina

Este estilo está basado en la definición y uso de relaciones entre sus componentes y conectores. El estilo en sí usa implícitamente una estructura de subsistemas. Su razonamiento jerárquico provoca la modificación en cascada, en donde se tiene que los cambios en una subrutina implican cambios en las subrutinas invocadas. La meta de este estilo es incrementar el desempeño distribuyendo el trabajo en múltiples procesos.

Arquitecturas Orientadas a Objetos

Los componentes de este estilo son los objetos o instancias de los tipos de dato abstractos. Según la caracterización de David Garlan y Mary Shaw [8], los objetos representan una clase de componentes que ellos llaman *managers*, debido a que son responsables de preservar la integridad de su propia representación. Un aspecto fundamental es que la representación interna de un objeto no es accesible desde otros objetos.

En las arquitecturas Orientadas a Objeto (OO), los componentes del estilo se basan en principios OO, tales como: el encapsulamiento, la herencia y polimorfismo. Sus interfaces están separadas de las implementaciones; y en cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases.

Entre sus cualidades se tiene que, permite la modelación de objetos y a través del encapsulamiento modificar la implementación el mismo sin afectar a sus clientes. Fundamentalmente, un objeto es una unidad reutilizable en el entorno de desarrollo. Entre las limitaciones, el principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto, se debe conocer su identidad, por ende, cuando un objeto es modificado se deben modificar además los objetos que lo invocan.

Arquitecturas Basadas en Componentes

Los sistemas basados en este estilo se basan en principios definidos por una ingeniería de software específica. Un componente de software, es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas. [9]

Las interfaces de estos componentes están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan cierto régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución.

En cuanto a las restricciones, puede admitirse que una interfaz sea implementada por múltiples componentes.

1.2.5.2 Estilos de código móvil

Este grupo de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico.

Arquitectura de Máquinas Virtuales

Este estilo también ha sido llamado como intérpretes basados en tablas. De hecho, todo intérprete involucra una máquina virtual implementada en software.

Las aplicaciones inscriptas en este estilo simulan funcionalidades no nativas al hardware y software en que se implementan, o capacidades que exceden a las capacidades del paradigma de programación que se está implementando. [9]

La ventaja fundamental de este estilo es que con una máquina virtual común el programa evita la redundancia de motores compitiendo por recursos y unifica *debuggers* y *profilers*.

Este estilo comprende dos sub-estilos, los cuales son los intérpretes y los sistemas basados en reglas. Ambas variedades abarcan, sin duda, un extenso espectro que va desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación.

Intérpretes

Se pudiera decir que un intérprete incluye un seudo-programa a interpretar y una máquina de interpretación. El seudo-programa a su vez incluye el programa mismo y el análogo que hace el intérprete de su estado de ejecución. La máquina de interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución. De este modo, un intérprete posee por lo general cuatro componentes: (1) una máquina de interpretación que lleva a cabo la tarea, (2) una memoria que contiene el seudo-código a interpretar, (3) una representación del estado de control de la máquina de interpretación, y (4) una representación del estado actual del programa que se simula.

En el Anexo 2 se abordan otros estilos arquitectónicos que no se incluyen en esta sección pues sus características no están vinculadas directamente con la aplicación a desarrollar, sin embargo, fueron agregados pues forman parte del objeto de estudio de la investigación.

1.2.6 Patrones de diseño

Para el desarrollo de la herramienta se debe realizar un estudio de los distintos patrones de diseño a fin de que sean usados convenientemente según corresponda.

1.2.6.1 Patrones GRASP

GRASP es un acrónimo que significa *General Responsibility Asignment Software Patterns* (patrones generales de software para asignar responsabilidades). Los patrones GRASP describen principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Estos tipos de patrones son detallados en el Anexo 3. [15]

1.2.6.2 Patrones GoF

GoF es un acrónimo que significa Gang of Four (“Banda de los cuatro”) que hace referencia a los autores del libro *Design Patterns* donde definieron un catálogo con 23 patrones básicos. Los patrones de diseño se clasifican en 3 grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento. Los patrones **creacionales** tratan con las formas de crear instancias de objetos. El objetivo de estos patrones es de abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados. Los **estructurales** por su parte describen cómo las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicio-

nados pueden ser incluso objetos simples u objetos compuestos. Finalmente, los patrones de **comportamiento** ayudan a definir la comunicación e interacción entre los objetos de un sistema. El propósito de este patrón es reducir el acoplamiento entre los objetos. [16]

1.2.7 Representación de la Arquitectura de Software

Comúnmente una arquitectura de software se documenta a través de un conjunto de vistas, donde cada vista representa un aspecto o comportamiento particular del sistema. Dos de los artículos de mayor relevancia que abordan el tema del uso de vistas son el conocido, "Modelo de 4+1 vistas de la arquitectura de software" de Philippe B. Kruchten [17], y el de Robert L. Nord y compañía titulado: "La arquitectura de software en aplicaciones industriales" [18]. Ambos artículos fueron publicados en el año de 1995. El primer artículo es el más conocido, quizás esto se deba a que la propuesta de Kruchten es parte fundamental de la metodología del Proceso Unificado de Rational (RUP), que en la actualidad es una de las metodologías que goza de gran popularidad.

Hoy en día es común que se utilice alguno de estos enfoques. Sin embargo, la forma de documentar una arquitectura ha evolucionado significativamente. La tendencia actual sobre esta práctica se centra en dos aspectos principales:

- Los arquitectos deben documentar las vistas que sean de mayor utilidad y no ajustarse a un número fijo de vistas, como es el caso en las propuestas de Kruchten y Nord.
- Documentar la arquitectura tomando en cuenta los intereses y necesidades de las personas involucradas en el proyecto, estos intereses se traducen como las cualidades que el sistema resultante debe poseer.

Esta nueva tendencia está respaldada por dos grandes institutos, uno de ellos es el Instituto de Ingeniería del Software (SEI[3]) con su propuesta, "Vistas y más allá de éstas, enfoque para la documentación de arquitecturas de software" [19] y el otro es el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) con el estándar, "IEEE 1471-2000, Prácticas recomendadas para la descripción arquitectónica de sistemas de software de gran demanda" [20], elaborado por el comité de estándares de la IEEE.

[3] Es un instituto federal de investigación y desarrollo, fundado por Congreso de los Estados Unidos para desarrollar modelos de evaluación y mejora en el desarrollo de software. [http://www.sei.cmu.edu/]

1.2.8 Evaluación y validación de la Arquitectura de Software

Todos los diseños arquitectónicos implican desventajas en las cualidades del sistema, ya que estas dependen en gran medida de las decisiones arquitectónicas, por lo que garantizar la calidad del producto final, está a menudo, estrechamente relacionado con avalar la calidad de la arquitectura y esto es posible si se realiza una evaluación de la misma [10]. El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad.

Bass[4] establece una clasificación de los atributos de calidad en dos categorías:

- Observables vía ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. (ver Anexo 4)
- No observables vía ejecución: aquellos atributos que se establecen durante el desarrollo del sistema. (ver Anexo 5)

¿Cuándo evaluar una arquitectura de software?

La evaluación se realiza cuando esta se encuentra especificada totalmente y no se ha iniciado su implementación. La arquitectura puede ser evaluada en cualquier momento de desarrollo. Existen dos variaciones útiles para realizar esta evaluación, la evaluación temprana y la evaluación tardía. [21]

1.2.8.1 Métodos de evaluación

El costo de corregir un error encontrado en los requisitos o en las primeras fases del diseño es, en orden de magnitudes, menor que, el costo del mismo error encontrado en las pruebas. La arquitectura es el producto de la fase inicial del diseño, y sus efectos en el sistema o en el producto es decisivo. La evaluación de una arquitectura es una forma barata para evitar desastres y futuros fracasos de diferentes tipos de sistemas de software. Hoy día existen una gran variedad de métodos de evaluaciones que sirven de guía a los involucrados en el desarrollo de un sistema para la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones. A continuación se brinda una breve descripción de los principales métodos.

[4] Miembro Senior del personal técnico en el Instituto de Ingeniería de Software (SEI), Estados Unidos.

Método ATAM

El Método de Análisis de Acuerdos de Arquitectura (*Architecture Trade-off Analysis Method*, ATAM) recibe su nombre debido a que no sólo revela qué tan bien una arquitectura satisface los objetivos particulares de calidad, sino que proporciona también una idea de cómo los objetivos de calidad interactúan entre sí y la forma en que se balancean unos con otros. Este método al ser estructurado, permite que el análisis sea repetible, lo que ayuda a asegurar que las preguntas correctas respecto a la arquitectura se puedan encontrar en etapas tempranas al diseño. El ATAM está inspirado en tres áreas diferentes: la noción de los estilos arquitectónicos, el análisis de los atributos de calidad y el Método de Análisis de la Arquitectura (SAAM, por sus siglas en inglés).

La parte principal de la ATAM está agrupada en 4 fases. La fase de Presentación, de Investigación y análisis, la fase de Pruebas y la fase de Presentación de informes. [22]

Método de Análisis de Diseños Intermedios

El Método de Análisis de Diseños Intermedios (ARID por sus siglas en inglés) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura [10]. Este método surge a partir de la combinación de lo mejor de *Active Design Review* (ADR) y los métodos de evaluación basados en escenarios como el ATAM, ARID, llena un nicho en el espectro de las técnicas de revisión de diseño. [22]

Kazman[5] propone que tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura [22]. De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una Arquitectura de Software. Al igual que el ATAM, ARID tiene 9 pasos principales, lo que en este caso el ARID los distribuye en 2 fases principales, la fase de Actividades Previas y de Revisión. [10]

[5] Profesor adjunto en el Departamento de Ciencias de la Computación en la Universidad de Waterloo, Canadá y Miembro Senior del Personal Técnico en el Software Engineering Institute de Carnegie Mellon University (Pittsburgh, Pensilvania Estados Unidos).

Método de Análisis de Arquitecturas de Software

El Método de Análisis de Arquitecturas de Software (*Software Architecture Analysis Method*, SAAM) es el primero que fue ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. Este método de evaluación se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. [10]

Método de Evaluación para Arquitecturas de Software Basadas en Componentes

El objetivo principal de MECABIC es evaluar y analizar la calidad exigida por los usuarios sobre las Arquitecturas de Software Basadas en Componentes (ASBC). El método adapta diferentes elementos de algunos métodos de evaluación arquitectónica como ATAM, ARID y establece un conjunto de pasos para determinar la calidad de los sistemas de software basados en componentes. Cabe indicar que aunque su estructura y mayor inspiración es ATAM, MECABIC se distingue de este porque incluye orientaciones para generar y discutir escenarios de evaluación iniciales, y un conjunto de preguntas a partir de las cuáles se pueden estudiar las decisiones arquitectónicas consideradas sobre ASBC.

Este método propone un árbol de utilidad inicial basado en el modelo de calidad ISO-9126[6] instanciado para la Arquitectura de Software. La adopción de este modelo por parte del MECABIC permite concentrarse en características que dependen exclusivamente de la arquitectura, además, al ser un estándar facilita la correspondencia con características de calidad consideradas por los métodos estudiados. Los escenarios incluidos en este árbol son específicos para aplicaciones basadas en componentes.

Este método está compuesto por 4 fases principales, la fase de Presentación, la fase de Investigación y análisis, la fase de Pruebas y por último la fase de Generación de la arquitectura final y reporte. [23]

1.2.8.2 Técnicas de evaluación

Para realizar la evaluación de la arquitectura se pueden emplear diversas técnicas existentes, las cuales están contenidas fundamentalmente en dos grupos, las **técnicas cualitativas**, entre las que se encuentran las técnicas de validación basadas en escenarios, los cuestionarios o listas de verificación, y las **téc-**

[6] http://iso25000.com/index.php/iso-iec-9126.html

nicas cuantitativas, haciendo uso de métricas, simulaciones, prototipos, experimentos o modelos matemáticos.

A continuación se identificarán algunas de estas técnicas:

Evaluación basada en escenarios

Kazman describe un escenario como una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con este [22]. El mismo está formado por tres partes:

- **Estímulo**: Parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema.
- **Contexto**: Describe qué sucede en el sistema al momento del estímulo.
- **Respuesta**: Describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado.

La utilización de escenarios como técnica de validación permite concretar y entender atributos de calidad además de ser muy simples de crear y entender, pocos costosos y muy efectivos. Este tipo de técnica cuenta con dos instrumentos de evaluación relevantes: el Árbol de Utilidades (*Utility Tree*) y los Perfiles (*Profiles*).

Árbol de Utilidades

Un Árbol de Utilidades o *Utility Tree* es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno. [22]

Su función es identificar los atributos de calidad más importantes en el software. Estos atributos son identificados previamente por los involucrados en el desarrollo del sistema en el momento de la construcción del árbol, el cual contiene como raíz la utilidad general del sistema. Los atributos de calidad asociados a él componen el segundo nivel del árbol, los cuales son refinados hasta obtener un escenario lo suficientemente preciso para ser analizado y priorizar cada atributo de calidad obtenido, los cuales contienen una serie de escenarios relacionados y una escala de importancia y dificultad por cada uno de ellos.

Perfiles

Se le denomina “perfil”, al conjunto de escenarios generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Los perfiles tienen asociados dos formas de especificación: perfiles completos y perfiles seleccionados.

Los perfiles completos definen todos los escenarios relevantes como parte del perfil. Su uso se reduce a sistemas relativamente pequeños y solo es posible predecir conjuntos de escenarios completos para algunos atributos de calidad.

Los perfiles seleccionados se asemejan a la selección de muestras sobre una población en los experimentos estadísticos. Se toma un conjunto de escenarios de forma aleatoria, de acuerdo con algunos requerimientos. Si bien es informal, permite hacer proposiciones científicamente válidas. [10]

Prototipos

Esta técnica implementa una parte de la arquitectura de software y la ejecuta en el contexto del sistema. Es utilizada para evaluar requerimientos de calidad operacional, como desempeño y confiabilidad. Para su uso se necesita mayor información sobre el desarrollo y disponibilidad del hardware, y otras partes que constituyen el contexto del sistema de software. Se obtiene un resultado de evaluación con mayor exactitud. [10]

Evaluación basada en simulación

La evaluación basada en simulación utiliza una implementación de alto nivel de la Arquitectura de Software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad. Este proceso de evaluación está basado en 5 pasos: [10]

- Definición e implementación del contexto.
- Implementación de los componentes arquitectónicos.
- Implementación del perfil.
- Simulación del sistema e inicio del perfil.

- Predicción de atributos de calidad.

Evaluación basada en modelos matemáticos

Esta técnica de evaluación se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro.

Entre las desventajas que presenta esta técnica se encuentra la inexistencia de modelos matemáticos apropiados para los atributos de calidad relevantes y el hecho de que el desarrollo de un modelo de simulación completo puede requerir esfuerzos sustanciales.

Evaluación basada en experiencia

En muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en factores subjetivos como la intuición y la práctica, la mayoría logran ser justificadas por una línea de razonamiento y pueden ser la base de otros enfoques de evaluación [10]. La evaluación basada en experiencia puede dividirse en dos grupos: la evaluación informal, que es la realizada por los arquitectos de software durante el proceso de diseño y la realizada por equipos externos de evaluación de arquitecturas.

Métricas

Las métricas son interpretaciones cuantitativas impuestas a particulares mediciones observables en la arquitectura, como el ventilador de entrada/salida de los componentes. Las técnicas de medición mejor investigadas dan respuestas a la complejidad global que puede sugerir la ubicación de los posibles cambios o donde el cambio será difícil. [22]

1.2.8.3 Modelos de Calidad

En la práctica, los Modelos Calidad resultan de utilidad para la predicción de confiabilidad y en la gerencia de calidad durante el proceso de desarrollo, así como para efectuar la medición del nivel de complejidad de un sistema de software. La organización y descomposición de los atributos de calidad ha permitido el establecimiento de modelos específicos para efectos de la evaluación de la calidad arquitectónica. [10]

Modelo de McCall

Este modelo describe la calidad como un concepto elaborado mediante relaciones jerárquicas entre factores de calidad, en base a criterios y métricas de calidad. Este enfoque es sistemático y permite cuantificar la calidad a través de las siguientes fases:

- Determinación de los factores que influyen sobre la calidad del software.
- Identificación de los criterios para juzgar cada factor.
- Definición de las métricas de los criterios y establecimiento de una función de normalización que define la relación entre las métricas de cada criterio y los factores correspondientes.
- Evaluación de las métricas.
- Correlación de las métricas a un conjunto de guías que cualquier equipo de desarrollo podría seguir.
- Desarrollo de las recomendaciones para la colección de métricas.

En este modelo, el término factor de calidad define características claves que un producto debe exhibir. Los atributos del factor de calidad que define el producto son los nombrados criterios de calidad. Las métricas de calidad denotan una medida que puede ser utilizada para cuantificar los criterios. [10]

Modelo de Dromey

Este modelo es un marco de referencia (o metamodelo) para la construcción de modelos de calidad, basado en cómo las propiedades medibles de un producto de software pueden afectar los atributos de calidad generales, como por ejemplo: confiabilidad y mantenibilidad. Dromey[7] sugiere el uso de cuatro categorías que implican propiedades de calidad, que son: correctitud, internas, contextuales y descriptivas. Este modelo está compuesto por 5 pasos:

- Especificación de los atributos de calidad de alto nivel.
- Determinación de los distintos componentes del producto a un apropiado nivel de detalle.
- Determinación y categorización de las implicaciones más importantes de calidad de cada componente.

[7] Profesor R. Geoff Dromey**,** director del Instituto de Calidad de Software, fundado en 1991 con sede en Alemania.

- Proposición de enlaces que relacionan las propiedades implícitas a los atributos de calidad, o, alternativamente, uso de enlaces de las cuatro categorías de atributos propuestas.
- Iteración sobre los pasos anteriores, utilizando un proceso de evaluación y refinamiento. [10]

Modelo ISO/IEC 9126

El estándar ISO/IEC 9126 ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software. Este estándar es una simplificación del Modelo de McCall, e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software. Cabe destacar que los factores de calidad que contempla el estándar ISO/IEC 9126 no son necesariamente usados para mediciones directas, pero proveen una valiosa base para medidas indirectas y una excelente lista para determinar la calidad de un sistema. [10]

ISO/IEC 9126 adaptado para arquitecturas de software

Losavio[8] propone una adaptación del modelo ISO/IEC 9126 de calidad de software para efectos de la evaluación de arquitecturas de software. El modelo se basa en los atributos de calidad (Anexo 6) que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad. Se plantea que la característica usabilidad propuesta por el modelo ISO/IEC 9126[9] puede ser refinada para obtener atributos que se relacionan con los componentes de la interfaz con el usuario. Dado que estos componentes son independientes de la arquitectura, no son considerados en la adaptación del modelo. La siguiente tabla presenta los atributos de calidad planteados por Losavio que poseen subcaracterísticas asociadas con elementos de tipo arquitectónico. [24]

1.3 Análisis de las soluciones existentes

En la actualidad existen varias investigaciones que enfrentan el problema del diseño del despliegue de WSAN dentro de edificaciones haciendo uso de herramientas informáticas. Aunque los artículos, libros y demás bibliografía que fue consultada en su mayoría no describen las particularidades arquitectónicas de las soluciones propuestas, sus características (lenguaje de programación, técnica de optimización em-

[8] Dra. Francisca Losavio, principal investigadora del Laboratorio de Tecnología de Software (LaTecS) de la Universidad Central de Venezuela.
[9] ISO 9126 es un estándar internacional para la evaluación de la calidad del software.

pleada, principales funcionalidades, etc.) es considerada una fuente de información de gran utilidad para la presente investigación.

Seguidamente se enuncia el estado del arte referente a las herramientas y tecnologías empleadas en el diseño del despliegue de WSAN en entornos interiores.

En la actualidad existen varias investigaciones que enfrentan el problema antes mencionado. Soluciones como las ofrecidas en [25-28], proponen interesantes enfoques para desplegar WSAN.

Para las personas que tienen la difícil tarea de realizar el despliegue de una WSAN, resulta muy conveniente la ayuda de estrategias y guías que le faciliten esta labor, algunas propuestas persiguen precisamente ese fin. Por ejemplo, en [29] se presenta una estrategia que permite desplegar sensores en interiores, para ello se fracciona el área de sensado en subregiones, donde serán desplegados los sensores teniendo en cuenta los obstáculos y la cobertura y conectividad entre los mismos.

Varias investigaciones tienen el objetivo de minimizar el número de sensores necesarios en el despliegue de la red, lo cual influiría de forma favorable en el costo total del proyecto. Existen propuestas de algoritmos que tienen esa misión entre sus principales aspiraciones [30-34].

Otras investigaciones se centran en el desarrollo de herramientas software que faciliten, de forma efectiva, el despliegue de sensores en el interior de edificaciones [35]. Estas herramientas pueden ser de gran utilidad en una etapa pre-despliegue, ya que los diseñadores pueden conocer aspectos claves dentro de un despliegue, como cobertura y conectividad de la red, los cuales anteriormente solo se obtenían siguiendo una estrategia básica de “instalar y probar", con los correspondientes gastos en cuanto a esfuerzo y tiempo que esta metodología puede ocasionar.

Una propuesta muy interesante es la presentada por Guinard et al. en [31], la cual tiene el objetivo de facilitar el despliegue de WSAN en SCE. Para ello se basaron en el desarrollo de una herramienta que incluye un algoritmo de optimización escalable para el diseño de WSAN y una metodología integrada. Esta herramienta fue creada con la finalidad de ayudar a los diseñadores e integradores de sistemas en una etapa previa al despliegue de la red de sensores, realizando una propuesta automática del número y la ubicación de los dispositivos inalámbricos.

Otra de las ventajas que posee la herramienta anteriormente mencionada, es la posibilidad de importar modelos de datos IFC (*Industry Foundation Classes*)[10], propuestas como [36] también incluye esta funcionalidad. Estos modelos permiten conocer importantes detalles de la edificación, por lo tanto el proceso de diseño de la red sería más eficiente. Para los casos en que no se disponga de los modelos IFC, la herramienta posee capacidades de dibujo y la opción de importar diseños directamente de AutoCAD.

En dicha herramienta, el proceso de optimización y diseño automático de la red se produce en dos fases. En primer lugar, se emplea un algoritmo basado en una red neuronal de tipo Gas Neuronal Creciente [37], para generar las posiciones candidatas de los dispositivos inalámbricos. Cada dispositivo posee, además de su posición candidata, un mapa de cobertura de su señal de radio correspondiente; para esta predicción la herramienta se basa en un modelo de propagación basado en el método Trazado de Rayos (*Ray Tracing*). En segundo lugar, se realiza un proceso de optimización el cual retorna el diseño de red propuesto, para ello la herramienta utiliza un algoritmo de optimización distribuido basado en agentes.

La propuesta de herramienta desarrollada por Huang et al. en [32], resulta igualmente interesante. La misma incluye relevantes funcionalidades como: representación de entornos interiores en 3D con la ubicación de los nodos sensores, generación de topologías de red de tipo árbol basadas en el estándar Zigbee [38] y la simulación de la propuesta de red resultante.

Existen otras herramientas, que además de tener presentes restricciones como la cobertura y conectividad interna de la red, tienen en cuenta otras métricas a la hora de generar las propuestas de despliegue, lo cual permite obtener diseños que se ajusten en mayor medida a las necesidades del cliente. En [39] el diseño de red que se genera, tiene presente restricciones como los costos, la latencia de la red y la tasa de errores de paquetes.

Las predicciones realizadas por este tipo de herramientas software, se basan en el empleo de modelos de propagación de la señal de radio. Estos modelos permiten predecir la pérdida en la trayectoria que una señal de RF pueda tener entre el transmisor y el receptor. Existen varios modelos de propagación para entornos interiores, por ejemplo, las herramientas propuestas en [30, 39, 40] utilizan el modelo Multi-Wall y

[10] Modelos que almacenan datos del edificio: dimensiones, paredes, materiales de construcción y otros detalles constructivos.

[31] el método Trazado de Rayos, otras son más abarcadoras y soportan varios modelos de propagación para las simulaciones [36].

Otro punto a su favor que poseen algunas herramientas, es la inclusión de mecanismos que permiten realizar simulaciones que tienen en cuenta el tipo de materiales de construcción del edificio, lo cual dota de mayor precisión al diseño resultante [36, 40]. Otra forma de incorporarle precisión a las simulaciones, es el empleo de algoritmos que tengan presente los obstáculos a los que la señal de RF puede enfrentarse dentro de una edificación. Algunas herramientas como [30, 31] tienen en cuenta los obstáculos en sus simulaciones, sin embargo, otras propuestas no poseen esta importante funcionalidad [26, 33, 41].

En la actualidad, algunos fabricantes desarrollan productos industriales que incluyen funcionalidades relacionadas con el despliegue de redes de sensores [35, 42]. Por ejemplo, la corporación *Daintree Networks* fabricó una herramienta que permite el desarrollo, decodificación, depuración y despliegue de WSAN; esta herramienta es capaz de generar de forma automática la posición de los dispositivos inalámbricos [35]. El fabricante *Spinwave Systems* desarrolló *NetQuest Software*, herramienta que permite realizar simulaciones previas al despliegue de la red [42].

A pesar de que las propuestas anteriores ofrecen relevantes soluciones en este ámbito de exploración, aún existen enfoques que no han sido abordados y deficiencias sin resolverse, los cuales representan campos de investigación abiertos a nuevas innovaciones.

Las propuestas de estrategias y guías que ayudan al diseñador en el despliegue de una WSAN pueden resultar de gran utilidad, la estrategia presentada en [29] persigue esa finalidad. A pesar de sus ventajas, dicha propuesta presenta algunas deficiencias que deben ser señaladas. Su simulación es basada en un modelo de propagación idealizado, donde son predecibles tanto el rango de comunicación como el de sensado, además este modelo plantea que debe existir una línea de visión directa entre los sensores para el proceso de comunicación/sensado. Por su parte, en [27] y [28] se consideran que son iguales los rangos de comunicación y sensado.

Existen otras herramientas que incluyen modelos de propagación para realizar sus predicciones. Propuestas como [30, 39, 40] utilizan el modelo Multi-Wall. A pesar de ser computacionalmente eficiente, este es un modelo de propagación empírico, el cual para poder realizar predicciones más exactas de la propagación de la señal de RF, requiere frecuentemente ser ajustado para determinados entornos. Esta deficiencia provoca atrasos considerables al proceso de diseño. Es importante tener presente que predicciones erró-

neas de la propagación de la señal de radio, puede aumentar considerablemente la tasa de error de paquetes, resultando en un funcionamiento deficiente de la red, es por ello que resulta sumamente importante realizar una correcta selección del modelo de propagación a ser usado en una herramienta.

Otro elemento importante a tener en cuenta en las simulaciones, es la utilización de algoritmos que consideren los obstáculos a los que la señal de RF puede enfrentarse en un entorno interior. Aunque la inclusión de obstáculos dota a las simulaciones de una mayor precisión, las propuestas [26, 29, 33, 41] carecen de esta funcionalidad.

Con la finalidad de cumplir las expectativas de los clientes, resulta importante la inclusión de nuevas restricciones a las propuestas de diseño generadas por varias herramientas. Algunas propuestas como [26] y [39], incorporan restricciones como los costos, el consumo energético, la latencia de la red y la tasa de errores de paquetes. Sin embargo, varias herramientas solo tienen presente la cobertura y la conectividad interna de la red a la hora de generar las propuestas de despliegue [26, 27, 29-33, 36, 41, 43].

La inclusión de nuevos mecanismos a las herramientas, los cuales posibiliten la realización de simulaciones que tengan en cuenta el tipo de materiales de construcción del edificio, son también requeridos. Estos mecanismos permiten la obtención de diseños más precisos. Sin embargo, existen pocos ejemplos de herramientas que soportan dichos mecanismos [36, 40].

El objetivo principal de algunas herramientas de diseño de WSAN, es ayudar al diseñador en una etapa previa al despliegue de la red, sugiriendo de forma automática el número y la posición de los dispositivos inalámbricos [34]. La generación de propuestas de topologías de red resulta de gran utilidad, lo cual permite reducir los esfuerzos en el diseño para los casos en que se realiza un despliegue completamente nuevo de una WSAN dentro de un edificio existente o un diseño antes de que el edificio haya sido construido.

Existen algunas herramientas que permiten generar propuestas de topologías de red [31, 32, 39]. Lamentablemente, las mismas no satisfacen los diseños requeridos por los SCE actuales. La herramienta propuesta en [31] solo soporta topologías single-hop, es decir, solo permite topologías de tipo estrella. En [39] se avanza un poco más, al permitir la generación de topologías de tipo árbol, sin embargo, esta herramienta no permite calcular la posición de los sensores y actuadores, además no considera los tipos de materiales del edificio. A su vez, la herramienta ofrecida por Huang et al. en [32] también permite generar topologías de tipo árbol, no obstante, con respecto a [31] tiene la desventaja de no poseer herramientas

que faciliten el modelado del edificio para los casos en que no se cuente con un modelo IFC que contenga los detalles del mismo, algo similar ocurre con [36]. Otras herramientas, sin embargo, no permiten generar propuestas de topologías [40, 42].

A pesar de las ventajas, desventajas y diferencias existentes entre las herramientas anteriores, existe un común denominador entre ellas, todas tienen el objetivo de facilitarle el trabajo al diseñador de sistemas, ayudándole en la difícil tarea de desplegar una WSAN dentro de una edificación.

1.4 Metodologías de desarrollo de software

Según la Real Academia Española una metodología de desarrollo de software no es más que un conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal. [44]

Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Hoy en día existen numerosas propuestas metodológicas que inciden en el proceso de desarrollo. Ejemplo de ellas son las propuestas tradicionales, centradas específicamente en el control del proceso. Estas han demostrado ser efectivas y necesarias en un gran número de proyectos, sobre todo aquellos proyectos de gran tamaño (respecto a tiempo y recursos). Sin embargo la experiencia ha demostrado que las metodologías tradicionales no ofrecen una buena solución para proyectos donde el entorno es volátil y donde los requisitos no se conocen con exactitud, porque no están pensadas para trabajar con incertidumbre.

Como respuesta a este problema surgieron otras metodologías que tratan de adaptarse a la realidad del desarrollo de software: las metodologías ágiles, que cambian significativamente algunos de los énfasis de los métodos ingenieriles. La diferencia inmediata es que son menos orientadas al documento, exigiendo menor cantidad de documentación para una tarea dada. De muchas maneras son más bien orientados al código, siguiendo un camino que dice que la parte importante de la documentación es el código fuente.

A continuación se realizará una explicación breve de las metodologías existentes más difundidas y que son utilizadas en la Universidad de las Ciencias Informáticas (UCI).

1.4.1 Rational Unified Process

Rational Unified Process (en lo adelante RUP) es uno de los procesos más generales de los existentes actualmente, ya que está pensado para adaptarse a cualquier proyecto, y no tan solo de software. Unifica

completamente a un equipo de desarrollo de software y optimiza la productividad de cada uno de los miembros del equipo, brindándoles la experiencia de los líderes de la industria y las lecciones aprendidas a través de miles de proyectos. Está fundamentado en un enfoque orientado a modelos de desarrollo basado en componentes, utilizando para ello el Lenguaje de Modelado Unificado (*Unified Modeling Language*, en lo adelante UML) el que define técnicas de análisis y diseño que ayudan a la confección de una solución sólida de software.

RUP se caracteriza por ser dirigido por casos de uso, donde los casos de uso definen lo que el usuario desea a partir de la captura de requisitos y la modelación del negocio. Es centrado en la arquitectura, característica que brinda una visión completa del sistema, se describen los procesos del negocio que son más importantes para comprenderlo, desarrollarlo y producirlo de una forma eficaz. Por último, es iterativo e incremental, donde cada fase se desarrolla en iteraciones, de forma tal que se pueda dividir en pequeños proyectos mejorando su comprensión y desarrollo.

Un proyecto realizado siguiendo la metodología RUP se divide en cuatro fases:

1. Inicio
2. Elaboración
3. Construcción
4. Transición

En cada fase se ejecutarán una o varias iteraciones (de tamaño variable según el proyecto) y dentro de cada una de ellas seguirá un modelo de cascada para los flujos de trabajo. Define además nueve disciplinas a realizar en cada fase del proyecto, donde seis de ellas son flujos de trabajo básicos y las otras tres son disciplinas de soporte. (Ver Anexo 7)

1.4.2 Programación Extrema

Programación Extrema *(eXtreme Programming,* XP según sus siglas en inglés) es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje

para enfrentar los cambios. Especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico. [45]

Esta metodología define Historias de Usuario (*User Stories*) como la base del software a desarrollar. Estas historias son escritas por el cliente y describen las interacciones entre los clientes y el sistema, generalmente son complementadas con otro tipo de descripción. A partir de ellas y de la arquitectura que se utilizará se planifican las entregas, así como los objetivos de cada una y las iteraciones con que contará.

Una característica distintiva de XP es la programación en parejas, con el objetivo de que el código sea revisado y validado antes de ser escrito; la refactorización de código está presente durante todo el desarrollo y no es más que escribir el mismo código fuente nuevamente buscando claridad, pero sin cambiar la funcionalidad resultante. En el Anexo 8 se puede observar las fases de desarrollo de esta metodología.

1.4.3 SCRUM

SCRUM define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos. El desarrollo de software se realiza mediante iteraciones, denominadas sprint, con una duración entre 15 y 30 días. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. La segunda característica importante son las reuniones a lo largo proyecto. Estas son las verdaderas protagonistas, especialmente la reunión diaria (*Daily Scrum Meeting*) de 15 minutos, donde cada miembro del equipo contesta 3 preguntas: ¿Qué has hecho desde ayer? ¿Qué es lo que estás planeando hacer hoy? ¿Has tenido algún problema que te haya impedido alcanzar tu objetivo? [46]

SCRUM es un proceso ágil que se enfoca en la entrega del mayor valor de negocio en el menor plazo, teniendo en cuenta las funcionalidades priorizadas por el cliente. Permite inspeccionar software listo para ser liberado en forma rápida y continua (cada dos a cuatro semanas). Por tanto cada dos a cuatro semanas cualquiera puede ver el software funcionando y decidir liberarlo como está o continuar mejorándolo durante otra iteración. En el Anexo 9 se puede observar las fases de desarrollo de esta metodología.

1.4.4 Selección de la metodología de desarrollo

No existe una metodología universal para hacer frente con éxito a cualquier proyecto de desarrollo de software. Toda metodología debe ser adaptada al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.).

RUP es conocido por la robustez de su proceso de desarrollo a largo plazo, SCRUM por su parte es una metodología ágil que enfatiza en reuniones diarias, algo que es innecesario debido a la pequeña dimensión del equipo de desarrollo que siempre está en contacto. Es por la rapidez a corto plazo de las entregas y a las cuestiones que aparecen a continuación, que XP es la metodología seleccionada para el desarrollo de la presente solución:

- Asume que la planificación nunca será perfecta y que los requerimientos cambian a lo largo de todo el ciclo de vida de la aplicación según varíen las necesidades del negocio
- El período de desarrollo es corto, limitado a solamente 4 meses de trabajo continuo.
- El cliente forma parte del equipo de desarrollo.
- Las dimensiones del proyecto son pequeñas.
- Promueve un entorno físico con un ambiente que permite la comunicación y colaboración entre todos los miembros del equipo.

1.5 Lenguaje de programación

Un lenguaje de programación es un lenguaje artificial usado para controlar el comportamiento de una máquina, especialmente una computadora.

Con el objetivo de identificar el lenguaje de programación que será empleado en el desarrollo de la arquitectura, en primer lugar se realizó un análisis para seleccionar los lenguajes más usados en la implementación de aplicaciones enmarcadas dentro del mismo dominio de aplicación de la herramienta a desarrollar. Dicho análisis tuvo en cuenta los principales requisitos funcionales y no funcionales de la misma, siendo la visualización y edición de entornos 3D y la implementación de técnicas de inteligencia artificial como los más determinantes. Teniendo en cuenta estas restricciones, se puede clasificar a la herramienta como: software de aplicación. De una forma más específica, es posible catalogarla como un híbrido entre aplicación gráfica y aplicación de inteligencia artificial.

Posteriormente, se realizaron entrevistas a expertos en el desarrollo de este tipo de aplicaciones y se consultó la bibliografía actualizada sobre dicha temática. Como resultado se obtuvo que la gran mayoría de dichas aplicaciones han sido implementadas en lenguajes como C++, Java, Python, C#, lo cual acota el análisis del lenguaje de programación a ser usado en la implementación de la herramienta a los cuatro antes mencionados. A continuación se presenta un análisis de sus principales características con respecto al tipo de aplicación que se piensa desarrollar.

1.5.1 C++

Creado en los Laboratorios Bell[11] por Bjarne Stroustrup a mediados de 1980. Es un poderoso sucesor de C. Añade a C el concepto de clases, un mecanismo para proveer tipos de datos definidos por el usuario, también llamado tipo de datos abstractos. C++ soporta el paradigma de la Programación Orientado a Objetos (en lo adelante POO). [47]

Al compilar las aplicaciones realizadas en este lenguaje, se genera código objeto, nativo para cada sistema operativo. Por tal razón no necesita intérpretes por ser compilado, obteniéndose como resultado aplicaciones sorprendentemente rápidas.

Otro punto a tener presente es que permite un control de la memoria y una capacidad de programación de bajo nivel sorprendente, con lo cual se provee un acceso completo al sistema. Esta característica provee una increíble velocidad a la aplicación, pues estaría interactuando directamente con el sistema operativo sin necesidad de intérprete.

Su principal desventaja frente a los lenguajes más actuales, es la ausencia de un “recolector de basura”, teniendo que encargarse el programador de cerrar las referencias de memoria que no vayan a ser usadas. Esta cualidad convierte la tarea de desarrollo algo tediosa, pues mayor control en tu programa implica más dificultad en el desarrollo. Otro punto en contra es que las rutinas de bajo nivel no son portables (que puedan ejecutarse en otro sistema operativo), por lo cual limita el uso de las aplicaciones en otros entornos.

[11] Son varios centros de investigación científica y tecnológica ubicados en más de diez países y que pertenecen a la empresa estadounidense Lucent Technologies.

Es muy utilizado también en la creación de aplicaciones de Inteligencia Artificial. Se han desarrollado librerías para la implementación de algoritmos genéticos, como es el caso de la librería GAlib[12], desarrollada en 1996 por Matthew Wall y su equipo en el MIT[13] en Estados Unidos; la librería aiParts[14], utilizada para desarrollar la Inteligencia Artificial de múltiples problemas de decisión; y la librería EO[15] (*Evolving Objects*) utilizada en la creación de algoritmos evolutivos.

Posee un *framework* de desarrollo llamado Qt[16], que incluye clases, librerías y herramientas para la producción de aplicaciones de interfaces gráficas, que pueden operar en varias plataformas; incluye soporte de nuevas tecnologías como OpenGL, XML, Base de Datos, programación para redes, internacionalización, entre otras. Dispone de herramientas que facilitan la creación de formularios, botones y ventanas de diálogos con el uso del ratón. Otra librería muy utilizada para el desarrollo de aplicaciones gráficas con C++ es *winbgim.h*. Esta librería tiene como objetivo emular la librería *graphics.h* de Borland C++.

1.5.2 Python

Fue creado por Guido Van Rossum en el año 1990. Surgió como un sucesor del lenguaje de programación ABC[17], capaz de manejar excepciones e interactuar con el sistema operativo Amoeba[18].

Es un lenguaje de programación multi-paradigma, permite varios estilos, es dinámico y orientado a objetos. El principal objetivo que persigue es la facilidad, tanto de lectura, como de diseño, además agregar que es de libre distribución. Actualmente se usa en grandes plataformas como: YouTube y Google.

Permite la programación estructural y funcional y se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation[19]. Tiene gran soporte e integración con otros lenguajes y he-

[12] Librería de herramientas para el uso de Algoritmos Genéticos en C++. [http://lancet.mit.edu/galib-2.4/]
[13] Instituto de Tecnología de Massachusetts
[14] http://www.aiparts.org/
[15] Librería de C++ para la solución de problemas de Algoritmos Genéticos [http://www.pdg.cnb.uam.es/DocBioInfo/jjmerelo/]
[16] Aplicación multiplataforma para el desarrollo de aplicaciones basado en C++ [http://qt.nokia.com/]
[17] Desarrollado a principios de los 80 en el Centrum voor Wiskun de en Informática como una alternativa al lenguaje BASIC
[18] Nace como respuesta a la necesidad de desarrollo de software para computación paralela y distribuida. Desarrollado por un grupo de investigadores de la Universidad de Vrije (Amsterdam– Holanda)
[19] Organización encargada de promover, proteger y hacer avanzar el lenguaje de programación Python

rramientas. Entre sus principales ventajas está la integración de varias bibliotecas estándar, es rápido de desarrollar, sencillo y sus bibliotecas hacen gran parte del trabajo.

Como desventaja se puede decir que es un lenguaje interpretado y es más lento en comparación con C++ y Java, además se conoce que no es una gran opción para juegos en 3D de elevado nivel gráfico pues consume gran cantidad de Unidad Central de Procesamiento. No obstante posee implementadas algunas librerías gráficas tales como Soya3D, una versión para Python de la popular librería Ogre y Panda3D (*framework* 3D para el desarrollo de juegos). [48, 49]

Python ha desarrollado un sistema llamado LINDA[20], capaz de crear un canal de comunicaciones entre programas, utilizando lo que se conoce en el entorno de la inteligencia artificial como “sistema de pizarra”, siendo este muy potente y sencillo. Usado en la creación de librerías para la aplicación de técnicas de minería de datos y aprendizaje automático, ejemplo de ello es la librería: *scikit learn*[21]. Cuenta además con una colección de métodos de autoaprendizaje llamado PyPR (Patrón de Reconocimiento de Python) que incluye redes neuronales y procesos gaussianos. En el campo de la bioinformática, se ha empleado una colección de herramientas llamada PySAT[22] que permite el análisis de bases de datos moleculares.

1.5.3 C#

Lenguaje altamente expresivo que se ajusta al paradigma de la POO. Su sintaxis es similar a C++ y Java. Fue desarrollado en gran parte por Anders Hejlsberg[23].

En C# no existe el concepto de función global o variable fuera de una clase u objeto. Por su buen apego a la POO, es posible sobrecargar métodos y operadores. Soporta la definición de interfaces. Ninguna clase puede poseer más de un padre (no se permite la herencia múltiple), pero sí puede suscribir un contrato con diversas interfaces. Permite la definición de estructuras, pero, a diferencia de C++, aquí no son tan parecidas a las clases y poseen ciertas restricciones.

El código C# se compila como código administrado, lo que significa que se beneficia de los servicios de *Common Language Runtime*[24] (librería común en tiempo de ejecución, CLR por sus siglas en inglés). Es-

[20] Sistema basado en computación distribuida [http://pypi.python.org/pypi/linda/0.5.1]
[21] http://scikit-learn.org/stable/
[22] Herramientas de secuencia de análisis [http://www.embl.de/~chenna/PySAT/]
[23] Creador del mítico compilador Turbo Pascal y uno de los diseñadores líder del lenguaje de programación Delphi

tos servicios incluyen la interoperabilidad de lenguajes, recolección de basura, seguridad mejorada y compatibilidad de versiones mejoradas. [50]

Desde el punto de vista de desarrollo gráfico, la funcionalidad de 3D en *Windows Presentation Foundation*[25] (WPF) permite a los desarrolladores dibujar, transformar y animar gráficos 3D en el marcado y en el código de procedimiento. Los desarrolladores pueden combinar gráficos 2D y 3D para crear controles enriquecidos, proporcionar ilustraciones complejas de datos o mejorar la experiencia del usuario de la interfaz de una aplicación.

Con respecto al desarrollo de aplicaciones de inteligencia artificial, C# ha implementado librerías como *IMSL C# Numerical Library* [50], dirigida a la toma de decisiones para las empresas aprovechando el valioso conjunto de datos obtenido durante años, haciendo uso de algoritmos matemáticos y estadísticos. Cuenta además con el *framework* Aforge.NET[26], diseñado para desarrolladores e investigadores en los campos de visión artificial, inteligencia artificial, procesamiento digital de imágenes, redes neuronales, algoritmos genéticos y máquinas de aprendizaje.

Algunas de las principales desventajas que se derivan del uso de este lenguaje es que, en primer lugar se tiene que conseguir una versión reciente del framework .NET para tener actualizaciones de la librería, es necesario además tener algunos requerimientos mínimos del sistema operativo para poder trabajar adecuadamente (como contar con Windows NT 4 o superior, tener más de 4GB de espacio libre para la instalación, etc.)

1.5.4 Java

Es un lenguaje de programación *Open Source* (código abierto) que cumple con el paradigma de la POO. Fue desarrollado por Sun Microsystems a principios de los años 90. A partir del 13 de noviembre de 2006, ya es un proyecto completamente libre, tiene la licencia GPL v2.

---

[24] A diferencia de otros sistemas recientes, basados en máquinas virtuales, el CLR fue diseñado desde el principio para soportar una amplia gama de lenguajes de programación

[25] Proporciona a los programadores un modelo de programación unificado con el que generar experiencias de cliente inteligentes de Windows, en las que se incorpora la interfaz de usuario, multimedia y documentos.

[26] http://www.aforgenet.com/

Toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Es un lenguaje robusto. La gestión de memoria y punteros es realizada por el propio lenguaje y no por el programador. Otro punto a su favor es que contiene estructuras para la detección de excepciones y obliga al programador a escribir código fiable mediante la declaración de excepciones posibles para una determinada clase reutilizable.

Otras de sus características esenciales, es que incorpora Multi-Threading (ejecución de tareas concurrentes dentro de un mismo programa). Fue diseñado “partiendo de cero”, es decir, no necesitaba ser compatible con versiones anteriores de ningún lenguaje como ocurre con C y C++.

Una de las características más importantes es que los programas “ejecutables”, creados por el compilador de Java, son independientes de la plataforma, es decir, se ejecutan indistintamente en una gran variedad de equipos con diferentes microprocesadores y sistemas operativos. Es un lenguaje de desarrollo público y se puede tener acceso gratis al conjunto de herramientas de desarrollo de aplicaciones de Java (*Java Developer's Kit*, JDK según sus siglas en Inglés). Posee gran conectividad con Bases de Datos, ERP y otros sistemas.

Una ventaja importante que se debe tener presente para el tipo de herramienta que se va a desarrollar, es su trabajo con modelos 3D, aspecto en el que Java tiene un buen desempeño, ya que presenta gran variedad de librerías libres con buena potencia gráfica tales como jGL, JOGL y Java3D, por solo citar algunas. Además, se caracteriza por ser un lenguaje muy utilizado en la implementación de aplicaciones que incorporen técnicas de inteligencia artificial. Ejemplo de ello es la gran variedad de publicaciones y software realizados que emplean librerías como JGAP[27] (usada en la implementación de algoritmos genéticos). Añadir también, la existencia de *frameworks* como JADE[28] (plataforma para el desarrollo de sistemas multi-agentes) que facilitan el desarrollo de este tipo de aplicaciones.

[27] http://jgap.sourceforge.net/
[28] http://jade.tilab.com/

Existe además gran documentación y una amplia comunidad sobre este lenguaje. Sin embargo, resulta importante señalar que los programas realizados en Java suelen ser un poco lentos y para el manejo de código de bajo nivel, se deben utilizar métodos nativos, disminuyendo así su portabilidad. [51, 52]

1.5.5 Selección del lenguaje de programación

Se utilizará Java como lenguaje de implementación de la solución debido a que es un lenguaje de desarrollo de propósito general, para realizar todo tipo de aplicaciones.

Uno de los principales elementos que hacen diferente a Java es su portabilidad, pues permite que la misma aplicación escrita con este lenguaje se pueda ejecutar en cualquier máquina, independientemente del sistema operativo y de la configuración de hardware, gracias a la máquina virtual sobre la que se ejecutan los programas. En el caso de C#, la plataforma Mono permite la ejecución de las aplicaciones en Linux y Mac, diseñadas para Microsoft .NET en entorno Windows. Esta plataforma solo implementa los componentes cubiertos bajo el estándar ECMA[29] aprobado por Microsoft, el cual cubre parte de la librería de clases, el entorno de ejecución y el lenguaje C# [53, 54]. Con C++, sin embargo, si no se utilizan funciones específicas para una determinada plataforma, el mismo código puede ser funcional para todas las arquitecturas de los sistemas operativos.

En cuanto a la potencia gráfica, C++ supera a Java en este sentido, en cambio esto no es una característica determinista pues la aplicación a desarrollar no requiere de grandes prestaciones gráficas.

Al contrario de los demás lenguajes analizados, Java está apoyado por un gran número de librerías que son soportadas por diversas empresas como IBM y Oracle, además de encontrarse otra gran cantidad desarrolladas por terceros, simplificando el desarrollo y la implementación de las aplicaciones.

Java posee un amplio conjunto de librerías para el desarrollo de aplicaciones que incorporan técnicas de Inteligencia Artificial, siendo menor el número de estas en los demás lenguajes estudiados.

Implementa un gran número de clases, incluidas gratuitamente en el JDK, para realizar diversas tareas que permiten al desarrollador centrarse en el negocio y no en la implementación. Se puede desarrollar con IDEs muy robustos como es el caso de NetBeans y Eclipse. Es más solicitado en el mercado de software

[29] Ecma International es una organización internacional basada en membresías de estándares para la comunicación y la información.

por su gran cantidad de *frameworks* (Spring, Struts, Hibernate, etc.) que facilitan el desarrollo de proyectos.

1.6 Plataforma de Cliente Rico

Una Plataforma de Cliente Rico (*Rich Client Platform*, en lo adelante RCP), es un entorno para manejar el ciclo de vida de una aplicación, es la base para la creación de aplicaciones de escritorio.

La mayoría de las aplicaciones de escritorio tienen características similares como menús, barras de herramientas, barras de estado, progreso de visualización, visualización de datos, cargar y guardar datos específicos y configuración, internacionalización y sistemas de ayuda, y mucho más. Por estas y otras características típicas de cada aplicación una RCP provee un marco de trabajo con esas características que pueden ser rápida y sencillamente adicionadas a la aplicación.

Este tipo de plataformas están caracterizadas por poseer una:

- Arquitectura de aplicación flexible y modular.
- Adaptabilidad al usuario final.
- Habilidad para trabajar tanto en línea como desconectada.
- Distribución simplificada al usuario final.

El aspecto más importante de una RCP es su arquitectura. Las aplicaciones basadas en RCP son escritas en forma de módulos, con una lógica coherente entre sus partes independientes. Un módulo es descrito declarativamente y automáticamente cargado por la plataforma. [55]

1.6.1 Plataforma NetBeans

Esta plataforma está completamente basada en la Interfaz de Programación de Aplicación (*Application Programming Interface,* API por sus siglas en inglés) de Java utilizando los componentes AWT y Swing, y utilizando además los conceptos de *Java Standard Edition* (JSE) [55]. Facilita en gran medida el proceso de desarrollo de aplicaciones de escritorio. El ejemplo más fiel del desarrollo de aplicaciones es el caso de NetBeans, el cual ha sido desarrollado a sí mismo en sobre la plataforma antes mencionada.

Provee además disímiles API que reducen el tiempo de desarrollo. Contiene además un módulo que hace mucho más fácil y eficiente instalar o actualizar los módulos de la aplicación del usuario final. Está basada en estándares y componentes reusables, y como resultado las aplicaciones pueden ser desplegadas en

múltiples sistemas, tales como Windows, Linux, Mac, Solaris, etc. Contiene gran variedad de módulos que pueden ser reutilizados en el desarrollo de las aplicaciones; si estos módulos no concuerdan completamente con los requisitos de la aplicación, pues entonces se puede extender de estos componentes a través de sus puntos de extensión. Los componentes de interfaz de usuario como es el caso de los menús, las ventanas, las barras de tarea y otros componentes de la propia plataforma son reutilizables en el desarrollo de aplicaciones. La integración de un marco de trabajo para la creación de *wizards* (asistentes) que ayudan al usuario en pasos tan complejos como la instalación; el soporte de métodos y clases para la internacionalización de las aplicaciones y un excelente sistema de ayuda, son algunas de las principales características que se pueden encontrar en esta plataforma y que la hacen una excelente elección para el proceso de desarrollo. [55]

1.6.2 Plataforma Eclipse

Eclipse comenzó como una aplicación modular integrada a un IDE. En 2004 la versión 3.0 de Eclipse fue liberada, esta versión soportaba la reutilización de la Plataforma Eclipse para construir aplicaciones basadas en la misma tecnología que el IDE Eclipse.

Las aplicaciones basadas en Eclipse son construidas sobre una arquitectura de plugins. Lo cual proporciona crear aplicaciones que hereden de Eclipse con componentes adicionales.

La base de la arquitectura de Eclipse, su sistema modular, es la especificación OSGi. Este estándar altamente adoptado, permite la creación de módulos dinámicos para Java. Los plugins desarrollados se autodescriben usando un archivo de metadatos.

La interfaz de usuario de Eclipse está basada en SWT y JFace. Este maduro grupo de herramientas provee un amplio conjunto de elementos tales como visores, formularios, vinculación de datos, asistentes, etc. Estas herramientas nativas dotan de un excelente rendimiento a las aplicaciones. Existen muchas herramientas para la construcción de interfaces de usuario (GUI) para el IDE Eclipse, pero la mayoría de ellas son para uso comercial.

Provee un conjunto de puntos de extensión y clases para la comunicación entre módulos, así como funcionalidades para la actualización de las aplicaciones y sistemas de ayudas. Esta plataforma está bajo la licencia Eclipse Public License[30] y la ICU4J. [56-58]

1.6.3 Selección de la plataforma de desarrollo

Ambas plataformas proveen un sistema de módulos con administración de dependencias, módulos dinámicos y rutas de clases privadas para los mismos. Facilitan además un excelente servicio de infraestructura y una leve carga de los servicios de infraestructura. Las dos proveen excelentes sistemas con soporte para la actualización e integración de sistemas de ayuda y mucho más.

Sin embargo, se decide hacer uso de la Plataforma NetBeans para el desarrollo de la herramienta producto de las características que se describen a continuación en la Tabla 1.

Tabla 1. Comparación entre las plataformas RCP de NetBeans y Eclipse

	Plataforma NetBeans	**Plataforma Eclipse**
Herramientas de interfaz de usuario	Conjunto de herramientas estándar Swing	SWT
Diseño de interfaz de usuario	Libre y apoyado sobre la herramienta Matisse GUI Builder	Alternativas comerciales para Eclipse
Sistema modular	Sistema modular basado en el estándar OSGi y/o sistema de módulos específico de NetBeans	Sistema de módulos basado en el estándar OSGi
Soporte en JDK	La aplicación VisualVM, una aplicación basada en esta plataforma está incluida en el JDK, por lo tanto, muchas de las librerías de la plataforma NetBeans están incluidas en el JDK	Ninguna de las librerías que requiere para el funcionamiento están incluidas en el JDK, por lo que requiere carga adicional de archivos

[30] http://www.eclipse.org/legal/epl-v10.html

Entrenamiento	Entrenamientos basados en una comunidad libre para organizaciones no comerciales	Sin soporte equivalente

1.7 Entorno de Desarrollo Integrado

Los Entornos de Desarrollo Integrado de software (*Integrated Development Environment*, en lo adelante IDE), son herramientas que ayudan a los programadores a desarrollar software de forma más amigable. Es decir, aquellos en los que el programador puede acceder con el menor esfuerzo a diferentes recursos como editores, compiladores, herramientas de análisis, etc.

Teniendo en cuenta a Java como lenguaje de programación para el desarrollo de la herramienta se hace de gran importancia la selección de un ambiente adecuado para poder explotar todos los beneficios de este y agilizar el desarrollo del software. A continuación se abordan las herramientas a utilizar, teniendo en cuenta que sean libres y exponiendo sus características y ventajas para poder tener un mayor conocimiento de sus prestaciones y valorar su utilidad.

1.7.1 NetBeans

NetBeans es un proyecto de código abierto de gran éxito. Es un producto libre, gratuito y sin restricciones de uso. El soporte de *Java Enterprise Edition* (JEE) es de importancia, en especial para los desarrolladores de Jbuilder[31]. Dado que cuenta con el mejor soporte a estándares industriales de la tecnología Java, ha hecho que el desarrollo de aplicaciones basadas en este lenguaje del tipo empresarial sea más rápido y sencillo; además de que su facilidad de uso, su cumplimiento de regulaciones, sus perfiles de rendimiento y su flexibilidad entre plataformas sean mucho mayor. NetBeans sirve a los programadores para escribir, compilar, depurar y ejecutar programas. Posee un navegador web integrado. Presenta el editor con más colores y un soporte más eficiente para XHTML marcando con diferentes colores las etiquetas de apertura y cierre. Permite mostrar el avance del proyecto y las tareas que se han terminado a través de un Diagrama de Gantt. Tiene además un excelente completamiento de código, pistas de error y ventanas emergentes de documentación.

[31] JBuilder es un entorno de desarrollo para el lenguaje de programación Java de Borland

Está escrito en Java pero puede servir para varios lenguajes de programación. Este IDE contiene las herramientas para que los desarrolladores de software puedan crear aplicaciones de “escritorio”, *Enterprise* (para empresas), Web, y aplicaciones móviles, con el lenguaje Java, así como también C/C++, PHP, JavaScript, Groovy, y Ruby. [59]

1.7.2 Eclipse

Cuando se descarga el Eclipse SDK, se obtiene un equipo de instrumentos para desarrollo en Java o Java Development Toolkit (JDT) para escribir y depurar programas en Java; además se obtiene un ambiente de desarrollo de *plugins* (*Plugin Development Enviroment*) para heredar de Eclipse. Si todo lo que se quiere es un IDE para Java, no se necesita nada además que el JDT. Esto es para lo que la mayoría las personas usan Eclipse. Aunque Eclipse es escrito en Java y su principal uso es como IDE para Java, este es un lenguaje neutral. El soporte para desarrollo en Java es proveído por un componente embebido o *plugin*, pero además están disponibles *plugins* para otros lenguajes, como C/C++, Cobol, C# y ActionScript. En principio permite ejecutar un programa sobre cualquier plataforma. Es una extensible plataforma de código abierto (*open source*) para desarrollar herramientas. Administrado y dirigido por un consorcio de compañías de desarrollo de software con un interés comercial en promover Eclipse como plataforma compartida para herramientas de desarrollo de software. [60]

1.7.3 Selección del entorno de desarrollo

El uso de NetBeans Platform como plataforma para el desarrollo de la aplicación, producto de las ventajas que facilita su uso y su amplia documentación, inducen directamente a la selección de NetBeans como IDE para el desarrollo. A pesar de que ambos IDEs poseen características similares para el trabajo, NetBeans es más fácil e intuitivo de usar, está mejor integrado con Maven[32] y tiene un excelente editor de interfaz de usuario. Mediante su integración con Subversion SVN permitirá el trabajo con el control de versiones durante el desarrollo y la generación de la documentación de todas las interfaces y clases, siendo esta de vital importancia para implementaciones posteriores.

[32] Herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl. [http://maven.apache.org/]

1.8 Gestores de datos

Producto de que la aplicación debe manejar información relevante sobre los dispositivos para el trabajo, es necesario contar con un mecanismo de almacenamiento de información, haciendo que la misma sea persistente en el tiempo. El propósito general es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante, para un buen manejo de datos.

Debido a que la aplicación se ejecutará de forma independiente en una estación de trabajo y que el cúmulo de información que maneja es relativamente pequeño, se usará un gestor de base de datos local como SQLite para el almacenamiento de la información entre el sistema y la base de datos. Por otra parte el uso de archivos en formato XML estará destinado al intercambio de información, ya sea datos de dispositivos o configuración de la herramienta. En los epígrafes a continuación se explican estas tecnologías.

1.8.1 SQLite

SQLite es un proyecto de dominio público creado por D. Richard Hipp que implementa una pequeña librería de aproximadamente 500Kb programada en lenguaje C. Funciona como un sistema de gestión de base de datos relacionales. A diferencia de los motores de base de datos convencionales con la arquitectura cliente-servidor, SQLite es independiente, ya que no se comunica con un motor de base de datos, sino que las librerías de SQLite pasan a integrar la aplicación. La misma utiliza las funcionalidades de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices y los propios datos), son guardados como un solo fichero estándar, en la máquina local. [61, 62]

1.8.2 XML

XML (*eXtensible Markup Language*, Lenguaje de Marca Extensible, en lo adelante XML), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C) en 1998. XML es

una manera de definir lenguajes para diferentes necesidades, de ahí que se le denomine metalenguaje. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML[33].

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. [63]

1.9 Herramienta para el modelado

Las herramientas CASE (del inglés "*Computer Asisted Software Engineering*", Ingeniería de Software Asistida por Computadora) son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases. Estas herramientas brindan ayuda a los analistas, ingenieros de software y desarrolladores. Permiten además el modelado de los sistemas mediante diferentes diagramas y generación de código a partir de éstos, y viceversa. La herramienta CASE seleccionada para el modelado de aplicación es Visual Paradigm for UML.

1.9.1 Visual Paradigm for UML

Es una herramienta profesional multiplataforma que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. Soporta todos los diagramas UML y el Diagrama Entidad-Relación ayudando a una rápida construcción de aplicaciones con calidad y a un menor coste. Su interoperabilidad entre diagramas permite exportar diagramas de un modelo a otro con mucha facilidad. Pueden encontrarse múltiples usuarios trabajando sobre el mismo proyecto gracias a su integración con SVN (software empleado para el control de versiones). Produce documentación del sistema en formato PDF, HTML y MS Word. Genera código en una amplia gama de lenguajes, entre los que se encuentra Java. Posee una interfaz de uso intuitivo y profesional para el trabajo y se puede modelar en varios idiomas.

[33] *Mathematical Markup Language*, es un lenguaje de marcado basado en XML para expresar notaciones matemáticas en combinación con XHTML para que distintas máquinas puedan entenderla. [http://www.w3.org/Math/]

1.10 **Conclusiones**

Teniendo en cuenta el estudio realizado en el presente capítulo y después de haber analizado las herramientas y tecnologías necesarias, se toma la decisión de desarrollar la arquitectura de la herramienta Andrómeda usando el lenguaje Java debido a la inmensa comunidad que posee y a la diversa cantidad de librerías y la Plataforma de Cliente Rico: NetBeans Platform, asegurando de esta forma una gran velocidad en el desarrollo. Todo esto será implementado haciendo uso de herramientas tales como Visual Paradigm for UML para el modelado y NetBeans para la programación. El desarrollo será controlado y dirigido mediante la metodología XP.

Capítulo 2. Exploración y Planificación

En este capítulo se hace alusión a las fases de exploración y planificación, las dos primeras de la metodología de desarrollo XP. El objetivo principal de estas es conocer el alcance del producto a desarrollar y estimar los tiempos de entrega de cada versión. Se exponen, además, los artefactos que se generan a partir de los requerimientos expuestos por el cliente en el capítulo anterior.

2.1 Usuarios del sistema

Los usuarios del sistema son todas aquellas personas o sistemas que interactúan con el mismo con el objetivo de obtener un resultado específico. Como aplicación de escritorio, la herramienta en sí no presenta restricciones de acceso a ciertas funcionalidades para un grupo específico de usuarios y para otros no. Todos tienen los mismos privilegios sobre las funcionalidades internas de la aplicación. Los externos, como es lógico, dependen de las restricciones clásicas que imponen los sistemas operativos, tales como el acceso a documentos de otros usuarios, las preferencias independientes para cada sesión, etc.

Tabla 2. Definición de los usuarios del sistema

Usuarios del sistema	Justificación
Diseñador de WSAN	Puede ser tanto un diseñador de redes con una vasta o con poca experiencia en el diseño de WSAN. En caso del usuario no tener o tener pocos conocimientos sobre el trabajo con la aplicación, puede acceder a los tutoriales de instrucción en línea para adquirir los conocimientos básicos de trabajo con la misma.

2.2 Listas de reservas del producto

Dentro de los objetivos fundamentales que persigue la arquitectura de software desarrollada es que la misma pueda ser extensible mediante *plugins* o módulos que se agreguen en futuras implementaciones.

Otras de las restricciones del sistema que tienen un impacto significativo en la arquitectura son las siguientes:

- Insertar las restricciones para el diseño de la red.
- Gestionar los dispositivos (nodos sensores) mediante una biblioteca
    - Adicionar nuevos dispositivos a la biblioteca.
    - Exportar elementos de la biblioteca.
    - Eliminar los dispositivos creados por el usuario de la biblioteca.
- Permitir la estructura dinámica de las opciones del menú.
- Permitir la incorporación de nuevos modelos de propagación.
- Generar la propuesta de diseño de despliegue.

2.3 Aspectos no funcionales del sistema

Los **aspectos no funcionales** son propiedades o cualidades que el producto debe tener. Son características del mismo que lo hacen atractivo, usable, rápido y confiable.

1. **Software**
    - La herramienta deberá funcionar en los sistemas operativos Windows (XP o superior), MAC, Solaris y las distribuciones de GNU/Linux.
    - Se requiere de la Máquina Virtual de Java versión 1.6 o superior.
    - Se requiere de la librería Java3D.
2. **Hardware**
    - Memoria RAM: 128 MB como mínimo para sistemas Windows y 64 MB para las restantes plataformas.
    - Espacio en disco duro: 98 MB mínimo para Windows y más de 58 MB para los restantes [64].
3. **Diseño y la implementación**
    - La herramienta debe ser implementada usando el lenguaje de programación Java en su versión 1.6.

- La arquitectura seleccionada debe garantizar la extensibilidad y flexibilidad de los diferentes componentes de la herramienta, permitiendo que la evolución de la misma sea de forma fácil y dinámica.
- Hacer uso de la biblioteca Java3D en la visualización de modelos 3D.
- Para el desarrollo de la herramienta se empleará la versión 7.0 o superior del IDE NetBeans.
- Emplear la librería OpenIFCTool para realizar la carga de los modelos IFC.
- Estandarizar el código de la aplicación basándose en la notación camelCase que propone Sun MicroSystems.

4. **Apariencia o interfaz externa**
   - El diseño de la interfaz debe ser sencillo e intuitivo para el diseñador.
   - Debe poseer un área superior al 70% de la pantalla destinada al visor.
   - El tamaño de la fuente debe ser fácil de leer.
5. **Seguridad**
   - **Integridad**: Los plugins desarrollados para el trabajo con la herramienta deben ser archivos binarios que imposibiliten la modificación de la información de los mismos por terceros.
6. **Usabilidad**
   - La herramienta podrá ser usada preferiblemente por diseñadores experimentados y personas de poca experiencia en el diseño de WSAN. Además, puede ser empleada por un usuario común que posea el modelo IFC de una edificación y desee desplegar una red de sensores.
7. **Portabilidad**
   - La herramienta deberá funcionar en los sistemas operativos Windows (XP o superior), Mac, Solaris y las distribuciones de GNU/Linux.
8. **Aspectos legales**
   - La herramienta una vez desarrollada será de Código Abierto y licencia GPL.

2.4 Exploración

La exploración es la etapa del proceso de desarrollo de software que propone XP para comenzar la construcción de un producto. Una vez que los clientes entregan su propuesta al equipo de trabajo, comienza el

análisis en grupo, las horas en los pizarrones, las tormentas de ideas y la conceptualización del software. Al mismo tiempo se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo.

2.4.1 Historias de Usuario

Una de las mejores prácticas adoptadas en el desarrollo de software es la administración de requerimientos. XP propone en este sentido hacer uso de las Historias de Usuario como técnica para especificar las funcionalidades que brindará el sistema y constituye una manera muy dinámica de realizar esta actividad. Como su nombre lo indica son especificadas por los propios usuarios y por tanto redactadas en su lenguaje; de manera sencilla y breve, evitando tecnicismos innecesarios que puedan crear confusión, aunque los programadores pueden contribuir en la tarea. Permiten responder rápidamente a los requerimientos cambiantes y son la base para las pruebas funcionales del sistema.

A continuación se muestran las Historias de Usuario que rigen el desarrollo de la herramienta Andrómeda.

Tabla 3. Estructura de una Historia de Usuario

Historia de Usuario	
Número: Número sucesivo a partir de uno.	**Usuario:** El usuario del sistema que utiliza o protagoniza la historia**.**
Nombre historia: Identifica la Historia de Usuario.	
Prioridad en negocio: Define la relevancia e impacto de la Historia de Usuario para el negocio de acuerdo a las necesidades del usuario.	**Riesgo en desarrollo:** Define la dificultad técnica que supone desarrollar la Historia de Usuario desde el punto de vista del programador.
Puntos estimados: Permiten estimar duración de implementación.	**Iteración asignada:** Precisa la iteración a la que pertenece la historia de usuario.
Descripción: Explica en qué consiste la Historia de Usuario, teniendo en cuenta las acciones realizadas por el usuario y la respuesta brindada por el sistema.	
Observaciones: Información extra que se estime agregar para hacer más comprensible la Historia de Usuario. Por ejemplo: conceptos, pos condiciones, etc.	

Tabla 4. Insertar restricciones del diseño

Historia de Usuario	
Número: 1	**Usuario:** Diseñador de WSAN
Nombre historia: Insertar restricciones del diseño	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Bajo
Puntos estimados: 2 semanas	**Iteración asignada:** 3
Descripción: El primer paso para proceder con la generación de la propuesta de diseño de despliegue de la WSAN consiste en introducir las restricciones del diseñador. Para ello la herramienta debe brindar mecanismos que le permitan al diseñador de la WSAN insertar todas aquellas restricciones (latencia de la red, costos de instalación, costos de mantenimientos, número de nodos sensores, modelo de propagación, eficiencia, tipos de sensores, tipos de actuadores, etc.) que estime conveniente. Al culminar con la inserción de los datos, la herramienta debe visualizar una ventana donde, a modo de resumen, se muestren todas las restricciones que fueron insertadas por el usuario, en caso de existir un error en alguna, el diseñador podrá regresar, editarla y volver a la ventana de resumen (si el diseñador desea regresar a la restricción anterior que seleccionó, deben estar establecidos los mismos parámetros que habían sido seleccionados por él, es decir, no se requerirá volver a seleccionar la restricción).	
Observaciones: En caso que se desee leer o editar las restricciones que fueron seleccionadas estas se podrán visualizar en una ventana que debe aparecer en la parte derecha de la herramienta.	

Tabla 5. Gestionar dispositivos

Historia de Usuario	
Número: 2	**Usuario:** Diseñador de WSAN
Nombre historia: Gestionar dispositivos	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta.
Puntos estimados: 4 semanas	**Iteración asignada:** 1
Descripción: Para realizar el sensado de una red, la herramienta debe contar con dispositivos	

previamente almacenados en la biblioteca y permitir adicionar nuevos elementos a la misma, para ello es necesario que la herramienta posibilite insertar los datos requeridos para la creación de un dispositivo (modelo, fabricante, potencia de transmisión, frecuencia de transmisión, tipo de sensado, etc.) A demás de poder agregarlos a la biblioteca, el usuario deberá poder eliminar dispositivos de la herramienta. Para esto, el usuario podrá realizar una búsqueda manual entre los elementos de la biblioteca y seleccionar el (los) que desee eliminar, o realizar un filtrado basado en alguna de las características del mismo. El usuario podrá también exportar su biblioteca de dispositivos y actualizar la misma, bien importando nuevos dispositivos de forma manual o desde un sitio de descarga.
Observaciones: La acción de eliminar dispositivos solo será posible con los elementos creados por el propio usuario. La herramienta debe permitir la selección de todos estos dispositivos sin tener que elegirlos individualmente, si el usuario desea filtrar su búsqueda, la herramienta debe ser capaz de mostrar todos los elementos que coincidan con el criterio de búsqueda. Los dispositivos con que contará la herramienta estarán almacenados en una base de datos SQlite, los cuales se podrán exportar en un archivo compactado utilizando una estructura XML.

Tabla 6. Permitir la estructura dinámica del menú

Historia de Usuario	
Número: 3	**Usuario:** Sistema
Nombre historia: Permitir la estructura dinámica del menú.	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Medio
Puntos estimados: 2 semanas	**Iteración asignada:** 2
Descripción: Permite la incorporación de nuevos elementos a la paleta de menú de la herramienta de forma automática una vez que se realice la inserción de algún *plugin*.	
Observaciones: El sistema debe permitir cambios estructurales en la composición de la paleta del menú, permitiendo agregar nuevos elementos al mismo manteniendo la estética de su diseño. El objetivo de esta funcionalidad consiste en permitir que la herramienta sea extensible en el tiempo, en caso que el equipo de desarrollo implemente nuevas funcionalidades que modifiquen el menú principal, se implementaría un *plugin* que le permitiría al usuario poder actualizar su versión de la herramienta.	

Tabla 7. Permitir la incorporación de nuevos modelos de propagación

Historia de Usuario	
Número: 4	**Usuario:** Sistema
Nombre historia: Permitir la incorporación de nuevos modelos de propagación.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio.
Puntos estimados: 2 semanas	**Iteración asignada:** 2
Descripción: Permite la incorporación de nuevos modelos de propagación en forma de *plugins* para su posterior selección y trabajo.	
Observaciones: La herramienta deberá poder incorporar nuevos modelos de propagación de RF, proporcionándole al cliente varias opciones para realizar el diseño de las WSAN y asegurar la escalabilidad de la misma. En caso que el equipo de desarrollo implemente nuevos modelos de propagación de la señal de RF para ser usados en las simulaciones, se implementaría un *plugin* que le permitiría al usuario poder actualizar su versión de la herramienta.	

Tabla 8. Generar la propuesta del diseño de despliegue de la WSAN

Historia de Usuario	
Número: 5	**Usuario:** Diseñador de Red
Nombre historia: Generar la propuesta del diseño de despliegue de la WSAN	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta.
Puntos estimados: 3 semanas	**Iteración asignada:** 4
Descripción: Genera la propuesta del diseño de despliegue de la WSAN una vez terminada la simulación.	
Observaciones: Con esta funcionalidad se visualizaría en el modelo 3D de la edificación la ubicación de los nodos sensores y los actuadores.	

Para comprender la duración de las estimaciones de las historias de usuarios anteriores, es necesario aclarar que 1 semana laboral equivale a 5 días laborales de la misma. Se hace necesario realizar esta aclaración pues en ocasiones se cometen cálculos erróneos que estiman los tiempos de desarrollo en base a los 7 días de la semana.

2.4.2 Modelo de dominio

Un Modelo de Dominio es un artefacto construido con las reglas de UML durante la concepción de un proyecto informático. Dichos modelos pueden utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema. Similares a los mapas mentales utilizados en el aprendizaje, el Modelo de Dominio es utilizado como un medio para comprender el sector industrial o de negocios al cual el sistema va a servir.

Para la herramienta Andrómeda se identificaron conceptos y objetos relacionados con el despliegue de una red de sensores en la actualidad, no logrando definir procesos específicos del sistema, por lo que se propone la realización de un Modelo de Dominio, representado en la Figura 1.

Conceptos del dominio:

- **Diseñador**: Persona encargada de realizar el diseño del despliegue de la WSAN. Es el usuario que se auxiliará de la herramienta Andrómeda para que le ayude con propuestas de diseños de despliegues de dichas redes.
- **Propuesta de Diseño**: Propuesta de plano/modelo que visualiza el diseño de la red que se desea desplegar.
- **Restricciones**: Conjunto de limitantes o características específicas que debe cumplir la WSAN en cuestión para obtener un resultado especifico.
- **Despliegue**: Diseño/modelo donde se visualiza la posición y conectividad de todos los nodos sensores de una WSAN en el área destinada para el sensado.
- **Entorno**: Lugar específico donde se desea realizar el despliegue de la WSAN.
- **Sensor**: Dispositivo que detecta una determinada acción externa, temperatura, presión, etc., y la transmite a otros sensores, actuadores o *gateways*.
- **Actuador**: Dispositivo utilizado por el SCE, para ejecutar una acción determinada en el área de sensado a partir de una información recibida desde un nodo sensor o una estación base (PC).
- **Interior**: Parte interna de una edificación.
- **Exterior**: Parte externa de una edificación.

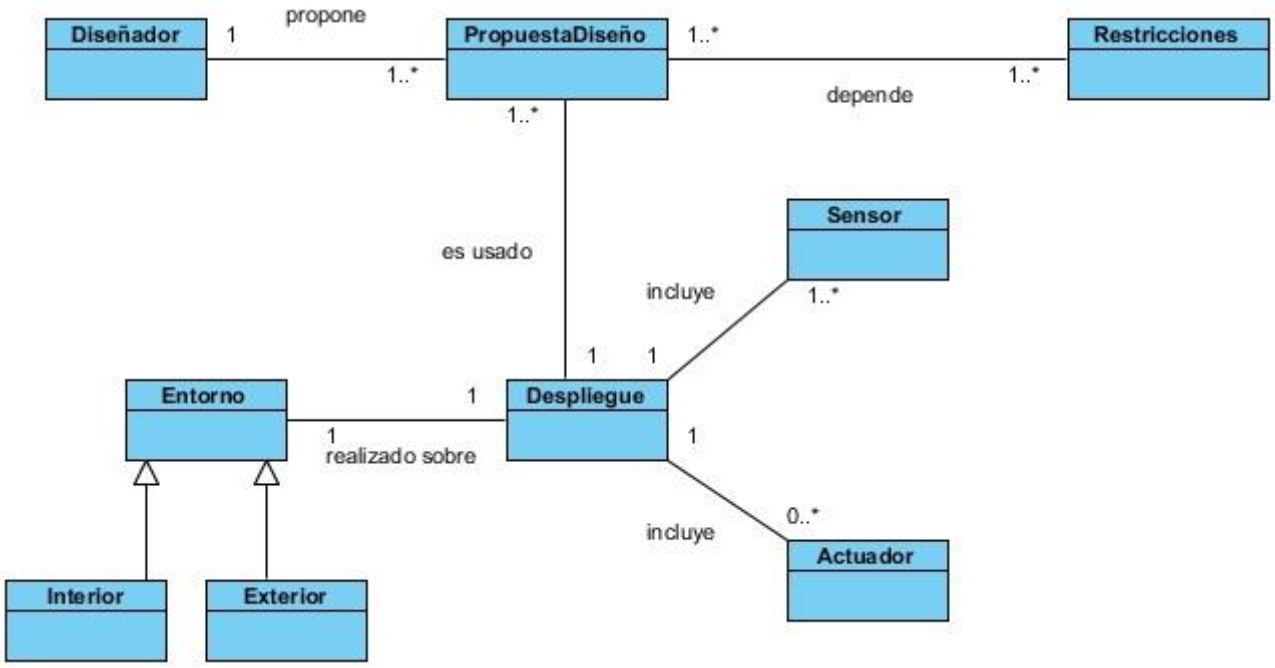

Figura 1. Modelo de Dominio

2.5 **Estimación**

Las estimaciones del esfuerzo para implementar las Historias de Usuario permiten tener una medida bastante real de la velocidad de progreso del proyecto y brindan una guía razonable a la cual ajustarse.

Tabla 9. Estimaciones por Historias de Usuario

No	Historias de Usuario	Puntos de Estimación
1	Insertar restricciones del diseño	2 semanas
2	Gestionar dispositivos	4 semanas
3	Permitir la estructura dinámica del menú	2 semanas
4	Permitir la incorporación de nuevos modelos de propagación	2 semanas
5	Generar la propuesta del diseño de despliegue de la WSAN	3 semanas

2.6 Planificación

Durante la fase de Planificación se realiza una estimación del esfuerzo que costará implementar todas las Historias de Usuario juntas, partiendo de la estimación que a cada una se le asignó en la fase de Exploración. Una vez terminado esto, se procede a organizarlas en las iteraciones correspondientes, en dependencia de la prioridad especificada por el cliente y del tiempo de desarrollo de cada una.

2.6.1 Iteraciones

Una iteración no es más que un mini-proyecto. Al finalizar una, se obtiene un resultado en software con un valor para el cliente. Claro está que este quedará totalmente satisfecho al finalizar la última iteración, ya que es la que concluye y completa el producto acordado inicialmente.

Para organizar las iteraciones no es recomendable extenderlas en más de 1 mes laboral, lo que sería generalmente 20 o 21 días. Los plazos de entrega y retroalimentación largos atentan contra el cumplimiento de los objetivos del cliente, sometidos a pequeños cambios de manera constante. En el plan de liberaciones se establece cuántas iteraciones serán necesarias realizar sobre el sistema para su obtención. Sin embargo, se precisa establecer el contenido de trabajo para todas y cada una de ellas y es aquí donde hace acto de presencia el Plan de Iteraciones, regulando la cantidad de Historias de Usuario a implementar dentro del rango establecido por la estimación efectuada.

A continuación se muestra la duración de cada una de las iteraciones y las respectivas Historias de Usuario a realizar en cada una de ellas junto con sus duraciones estimadas. En el Anexo 10 se muestra de forma más detallada un diagrama de Gantt para cada una de las iteraciones y sus respectivas tareas de ingeniería.

Tabla 10. Historias de Usuario organizadas por iteraciones

Iteraciones	Orden de las Historias de Usuario a implementar	Cantidad de tiempo de trabajo
Iteración 1	Gestionar dispositivos.	4 semanas
Iteración 2	Permitir la incorporación de nuevos modelos de propagación. Permitir la estructura dinámica del menú.	4 semanas
Iteración 3	Crear un nuevo proyecto y especificar en él las restricciones	2 semanas

	del diseño.	
Iteración 4	Generar la propuesta del diseño de despliegue de la WSAN.	3 semanas

2.6.2 Plan de Entregas

El Plan de Entregas es el compromiso final del equipo de desarrollo con los clientes. Es una cuestión de vital importancia para el negocio entre ambas partes, ya que la entrega tardía o temprana de la solución, repercute notablemente en la economía y moral de todos los involucrados. La estimación es uno de los temas más complicados del desarrollo de un proyecto de software y es por ello que resulta de vital importancia tener bien claros los requerimientos del cliente, el estilo de trabajo del equipo de desarrollo y el tiempo con que dispone el cliente para tener en sus manos la solución. En el más extremo de los casos, la honestidad debe prevalecer en lugar de la incertidumbre y saber decir cuándo se puede terminar el proyecto a tiempo o no, es muy importante.

Tabla 11. Plan de Entrega de la herramienta

Entregable	Final 1ra iteración	Final 2da iteración	Final 3ra iteración	Final 4ta iteración
Arquitectura de la herramienta Andrómeda	25/01/2012	13/03/2012	28/04/2012	31/05/2012

2.7 Conclusiones

En el este capítulo se realizó la documentación de la primera etapa del ciclo de vida de la solución propuesta: los artefactos de las historias de usuario, el plan de iteraciones y el de entregas, los cuales fueron detallados de forma clara.

Capítulo 3. Descripción e Implementación de la arquitectura

Teniendo en cuenta la metodología de desarrollo seleccionada, la herramienta de modelado y las diferentes tecnologías y herramientas presentes en la fundamentación teórica, el presente capítulo describirá mediante qué Arquitectura de Software será guiado el desarrollo del sistema. XP no propone concisamente los artefactos a utilizar en la implementación de una solución que utilice dicha metodología. Deja, en manos del equipo de desarrollo, dependiendo de sus capacidades de comunicación, la decisión de utilizar tantos tipos de diagramas de UML como crean posible, y así, facilitar el proceso de desarrollo. En el presente capítulo se describe además el diseño del sistema, los diagramas utilizados y las tareas generadas por cada historia de usuario.

3.1 **Selección de la arquitectura de software**

Para el desarrollo de la herramienta Andrómeda se decidió hacer uso de RCP de la Plataforma NetBeans, la cual ahorra un tiempo considerable y organiza el proceso de desarrollo. En la Figura 2 se puede evidenciar que la misma está formada completamente por módulos, lo cual ofrece una gran ventaja pues las aplicaciones pueden extenderse a través de módulos adicionales que se adapten a las necesidades de la aplicación final. Dichos módulos pueden comunicarse unos con otros sin que esto conlleve necesariamente algún tipo de dependencia entre ellos. [55]

Por tanto el uso de la arquitectura de la plataforma como base para el desarrollo, permite que las aplicaciones sean extensibles mediante *plugins* y módulos que pueden ser agregados, requisito fundamental en las aplicaciones actuales, pues facilita el mantenimiento de la aplicación y el desarrollo en paralelo de otros módulos por parte de los desarrolladores. En el caso de la presente investigación el uso de *plugins* se convierte en un requisito indispensable, ya que a la misma se le pueden adicionar nuevos *plugins* para la visualización de modelos tridimensionales o incluso para calcular la conectividad de la red a través de distintos modelos de propagación que pueden ser agregados.

Por tanto, el diseño modular de esta plataforma y su principio de funcionamiento, así como las ventajas que proporciona, orientan el desarrollo de la aplicación a través del uso de la misma la cual responde al estilo arquitectónico basado en componentes.

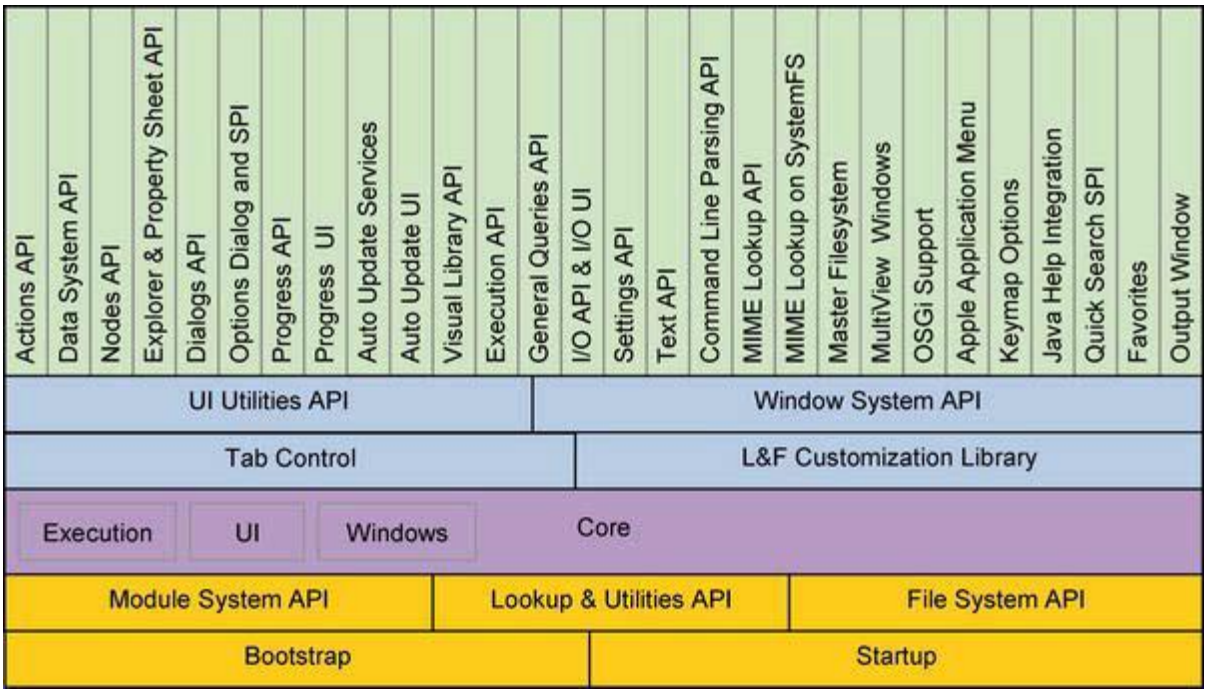

Figura 2. Arquitectura de la Plataforma NetBeans

3.2 **Descripción de la arquitectura**

El uso de las guías para la documentación de las arquitecturas de software (mencionadas en el epígrafe 1.2.7) incluye un gran número de documentos mucho mayor con respecto al sistema a desarrollar y con el tratamiento que propone la metodología XP para la descripción de la arquitectura.

En XP, el objetivo es guiar todos los desarrollos compartiendo una historia simple de cómo el sistema trabaja. Comúnmente el sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Este lenguaje de metáforas se utiliza para describir la arquitectura del sistema, ayudando a la comunicación entre el personal involucrado al proyecto. Martin Fowler [65] explica que la práctica de la metáfora consiste en formar un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema. Este conjunto de nombres ayuda a la nomenclatura de clases y métodos del sistema.

La metodología XP propone además que los modelos sean algo tan simple como un dibujo o bosquejo que se encuentren a disposición de todos los miembros del equipo, visible para el público, porque a pesar

de que las metáforas pueden ser muy eficaces, un modelo arquitectónico a menudo proporciona el mayor detalle que el equipo necesita. [66]

Para el desarrollo de la herramienta Andrómeda se utilizará una arquitectura modular basándose en la Plataforma NetBeans y guiándose por el estilo arquitectónico basado en componentes. Su arquitectura modular favorece la flexibilidad de la solución final al no tener todos los módulos centralizados en un mismo lugar y poder agregar nuevos módulos sin tener que modificar el núcleo de la herramienta.

Para la descripción de la arquitectura de la herramienta Andrómeda se utilizará el modelo de las 4+1 Vistas que propone Philippe B. Kruchten para representar el diseño del software, con la salvedad de que en este caso se usarán solamente las vistas Lógica y Desarrollo, las cuales serán descritas mediante el Diagrama de Clases y el Diagrama de Componentes respectivamente. La investigación considera que las vistas de Proceso, Física y de Casos de Uso no son necesarias ya que no modelan información relevante para arquitectura propuesta, es decir, la Vista de Procesos no describe el tipo de software que se desea desarrollar, la Vista Física carece de sentido producto a que solo existirá un nodo físico donde se ejecutará el producto y por último la Vista de Casos de Uso será descrita mediante las Historias de Usuario que propone la metodología XP.

3.2.1 Vista general de la arquitectura

Como se menciona anteriormente, la arquitectura propuesta será descrita mediante la Vista de Desarrollo y la Vista Lógica. La Figura 3 muestra el Diagrama de Componentes correspondiente a una visión general del sistema. Este es lo suficientemente abstracto como para dar un bosquejo de la solución, y representa los diferentes módulos que comprenden la arquitectura utilizada para dar respuesta a la solución.

A continuación se muestra una breve descripción de los módulos que componen el diseño arquitectónico de la herramienta.

- **NetBeans Platform**: Conjunto de clases, APIs e interfaces que provee la Plataforma NetBeans para el desarrollo de aplicaciones.
- **Andromeda Platform**: Subsistema que hace referencia al núcleo del sistema final a desarrollar. El mismo es una instancia de la Plataforma NetBeans, por lo cual está provisto del conjunto de APIs y clases que proporciona la plataforma. A su vez es el encargado de la gestión de carga de otros módulos.

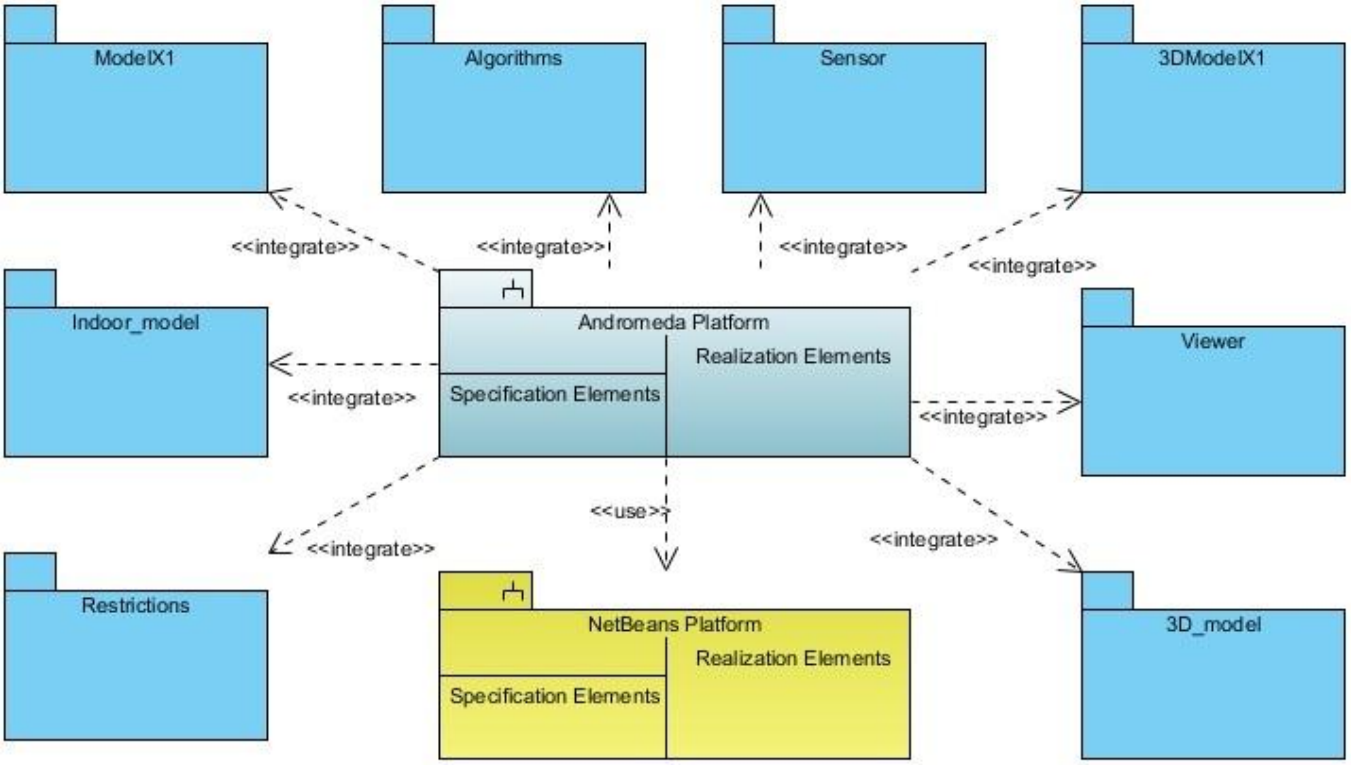

Figura 3. Diagrama de Componentes de la herramienta

- **Sensor**: Es el módulo encargado de obtener toda la información persistente sobre los dispositivos con los que cuenta la herramienta para el trabajo. Además contiene las clases y librerías necesarias para gestionar la información de los dispositivos así como exportar los mismos.
- **Viewer**: Módulo que implementa las funcionalidades necesarias para visualizar los modelos 3D que son cargados en la herramienta. Provee además una barra de herramientas con las funcionalidades necesarias para interactuar sobre los modelos 3D tales como *zoom*, rotación, mover, etc.
- **3D_model**: Módulo que contiene las clases e interfaces encargadas de interpretar los modelos 3D sobre los cuales va a operar el sistema.
- **3DModelX1**: Debido a la arquitectura modular definida para la aplicación, este módulo genérico representa la implementación de las funcionalidades necesarias para procesar un modelo 3D y generar las clases fundamentales para su visualización en la aplicación.
- **Algorithms**: Encargado de contener los algoritmos y clases necesarias para seleccionar las posiciones candidatas de los dispositivos en la red. Además incorpora las clases para determinar, a

partir de las posiciones candidatas seleccionadas, la(s) propuesta(s) de diseño(s) del despliegue de la red.

- **Indoor_model**: Este módulo contendrá las clases e interfaces principales que implementarán los distintos modelos de propagación de señal que serán adicionados a la herramienta. Estos modelos, mediante fórmulas matemáticas propuestas por otros especialistas, determinarán la conectividad entre los sensores de la red.
- **ModelX1**: Al igual que el módulo 3DModelX1, representa un modelo de propagación genérico que puede ser incorporado en forma de complemento a la herramienta para ampliar sus posibilidades de trabajo.
- **Restrictions**: Es el encargado para realizar la creación de un nuevo proyecto e insertar las restricciones del diseño que se propone inicialmente en la confección del proyecto.

3.2.2 Diagrama de clases

El Diagrama de Clases del Sistema describe la arquitectura y las relaciones entre cada una de estas clases. Es la vista más conceptual y detallada de la solución.

A continuación se representan los diagramas de clases que fueron realizados por la arquitectura y que representan la base del sistema.

3.2.2.1 Diagrama de clases del módulo Sensor

El siguiente diagrama de clases representa las clases y relaciones correspondientes al módulo Sensor de la aplicación. En el mismo se muestran las clases del núcleo del módulo y las principales clases correspondientes al manejo de las interfaces de la herramienta. Este diagrama es una versión simplificada del mismo, pues el diagrama íntegro fue colocado en el Anexo 12 debido a su amplitud.

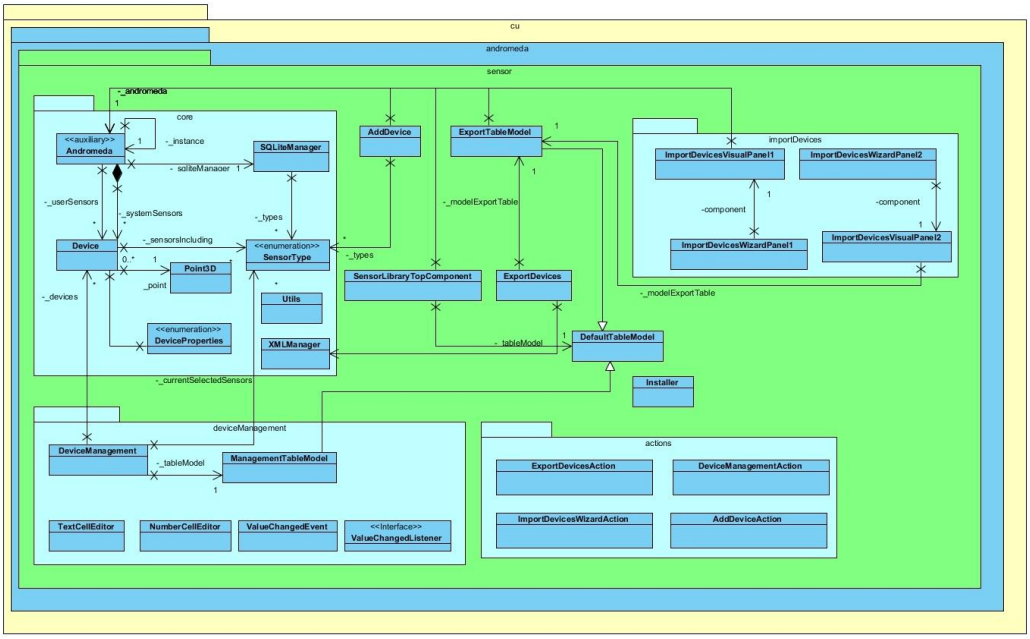

Figura 4. Diagrama de clases simplificado correspondiente al módulo Sensor

3.2.2.2 Diagrama de clases del módulo IndoorModel

El siguiente diagrama por su parte muestra las clases base del módulo IndoorModel, el cual contiene la interfaz que debe ser realizada por cada uno de los modelos de propagación a agregar a la herramienta y la clase Models para la obtención de los mismos.

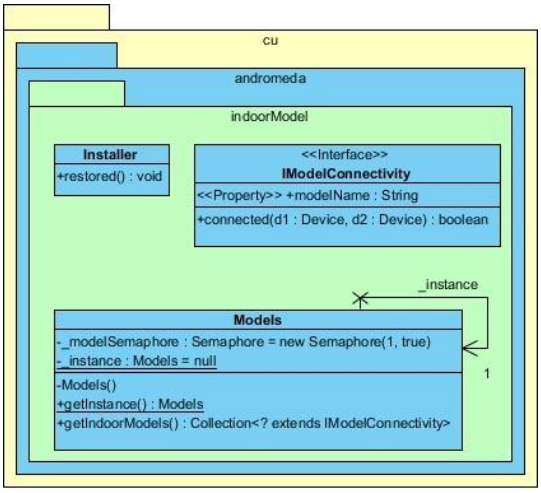

Figura 5. Diagrama de clases correspondiente al módulo IndoorModel

3.2.2.3 Diagrama de clases del módulo Restrictions

Diagrama de clases correspondientes al módulo Restrictions encargado de crear un nuevo proyecto e insertar las restricciones del diseño en la aplicación.

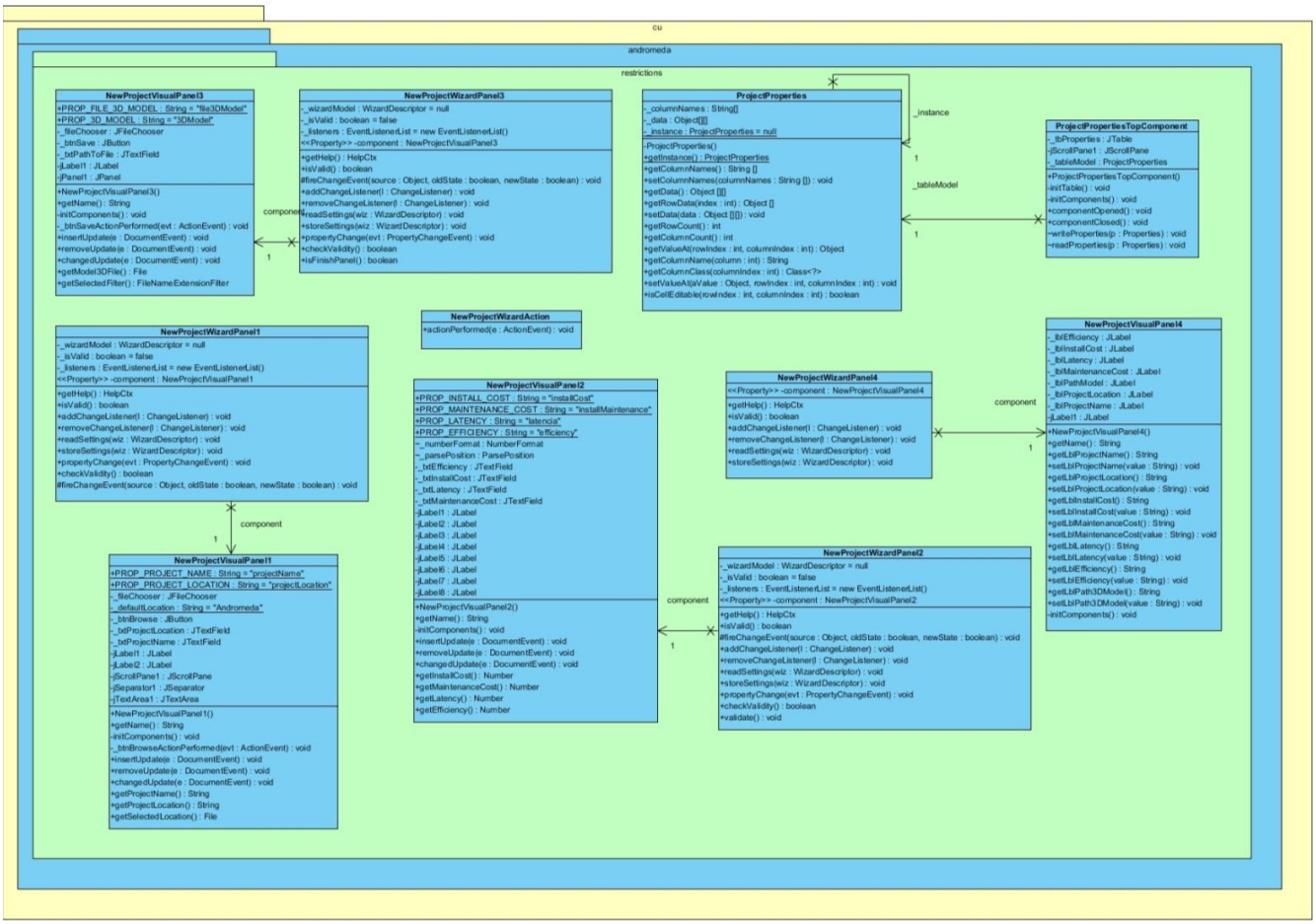

Figura 6. Diagrama de clases correspondiente al módulo Restrictions

3.2.2.4 Diagrama de clases del módulo Algorithms

El siguiente diagrama muestra las clases base correspondiente al módulo Algorithms, las cuales tendrán como objetivo seleccionar las posiciones candidatas que ubicarán los dispositivos en la red y posteriormente optimizar dichas posiciones para generar el diseño del despliegue final.

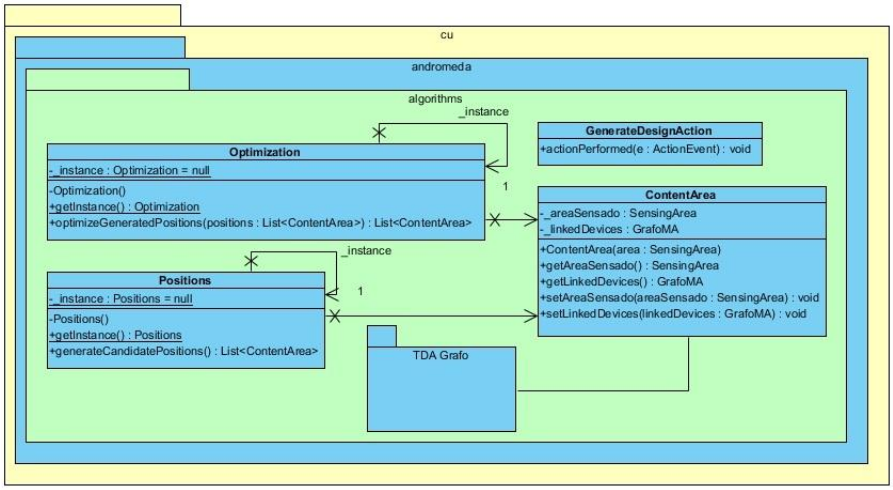

Figura 7. Diagrama de clases correspondiente al módulo Algorithms

3.2.3 Diagrama de Componentes: módulo Sensor

Este módulo basa su funcionamiento a partir de la clase controladora **Andromeda**, la cual contiene la lista de dispositivos que serán usados en la generación de la propuesta de diseño. Los dispositivos exportados e importados se procesarán a través de archivos XML haciendo uso de la librería **jDOM**. Esta librería provee de los métodos necesarios para operar sobre archivos XML de forma rápida y sencilla. Por otra parte, los dispositivos correspondientes al núcleo de la aplicación son cargados mediante la clase **SQLiteManager**, que hace uso a su vez del driver **JDBC-SQLite** para interactuar con bases de datos locales. Los dispositivos son finalmente mostrados mediante la clase **SensorLibrary**. Producto de la estructura modular de la aplicación, las clases estarán disponibles para ser usadas por los demás módulos.

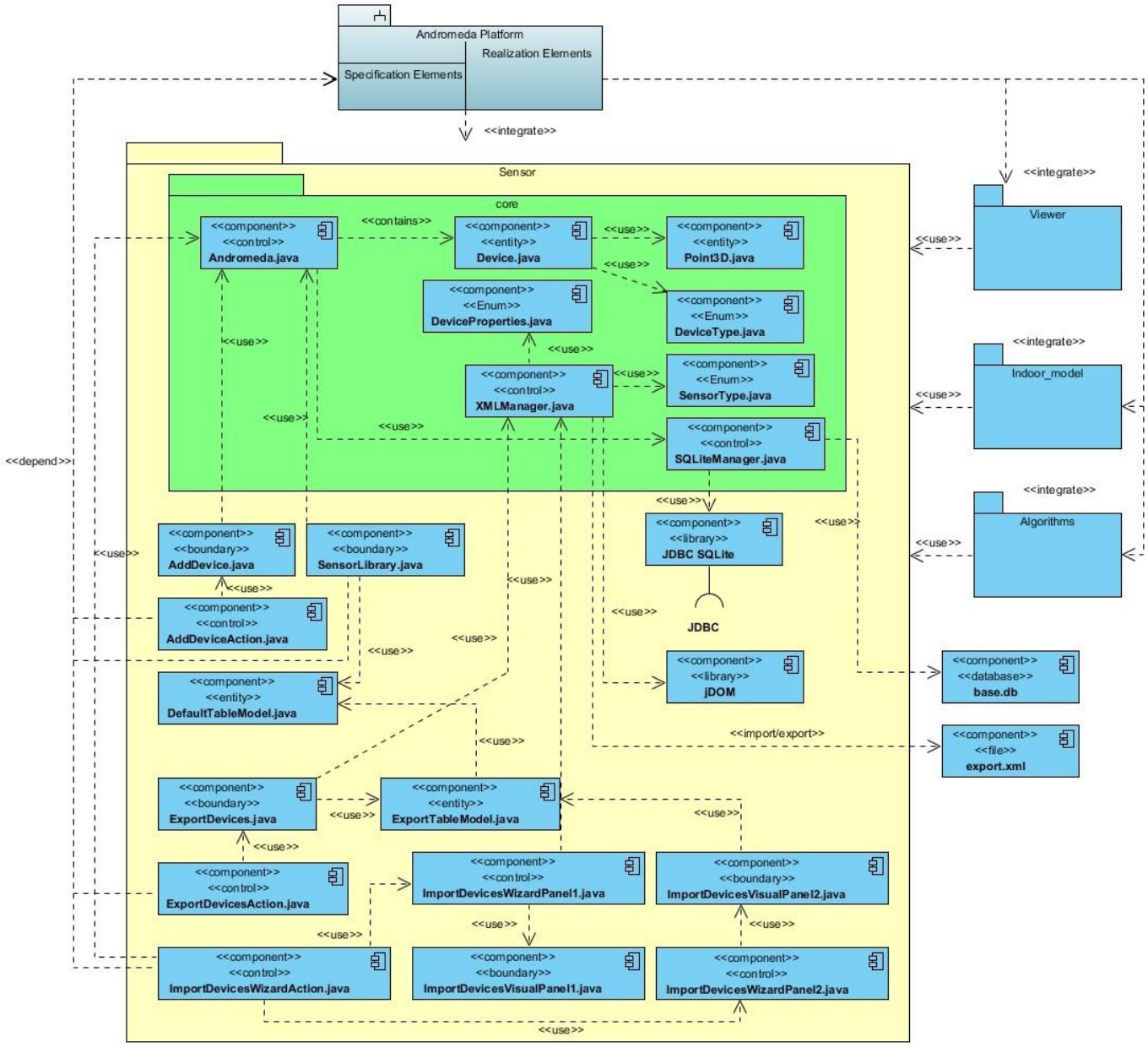

Figura 8. Diagrama de Componentes correspondiente al módulo Sensor

3.2.4 Diagrama de Componentes: módulo Viewer

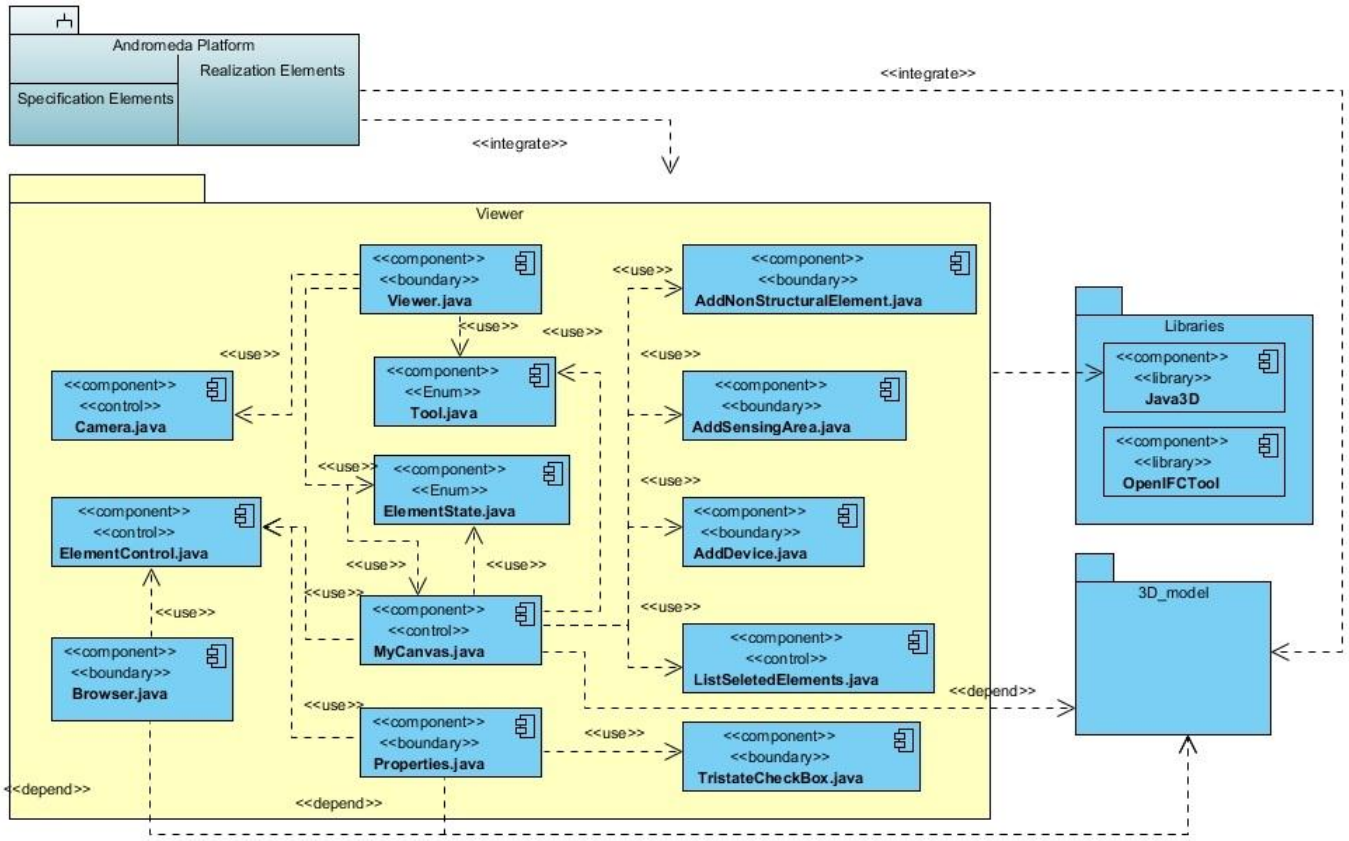

Figura 9. Diagrama de Componentes correspondiente al módulo del Visor

Este módulo, como los demás, se integra a la herramienta implementando un visor que mostrará el modelo 3D cargado tanto en el visor tridimensional como en el árbol jerárquico. Cada formato de archivo especifica su presentación y organización de forma diferente. Para la visualización de dichos modelos se hará uso de la librería de clases **Java3D**. Implementa las operaciones básicas de rotar, mover, etc. que serán accesibles desde la barra de herramientas y que son aplicables sobre los modelos 3D.

3.2.5 Diagrama de Componentes: módulo 3D_model

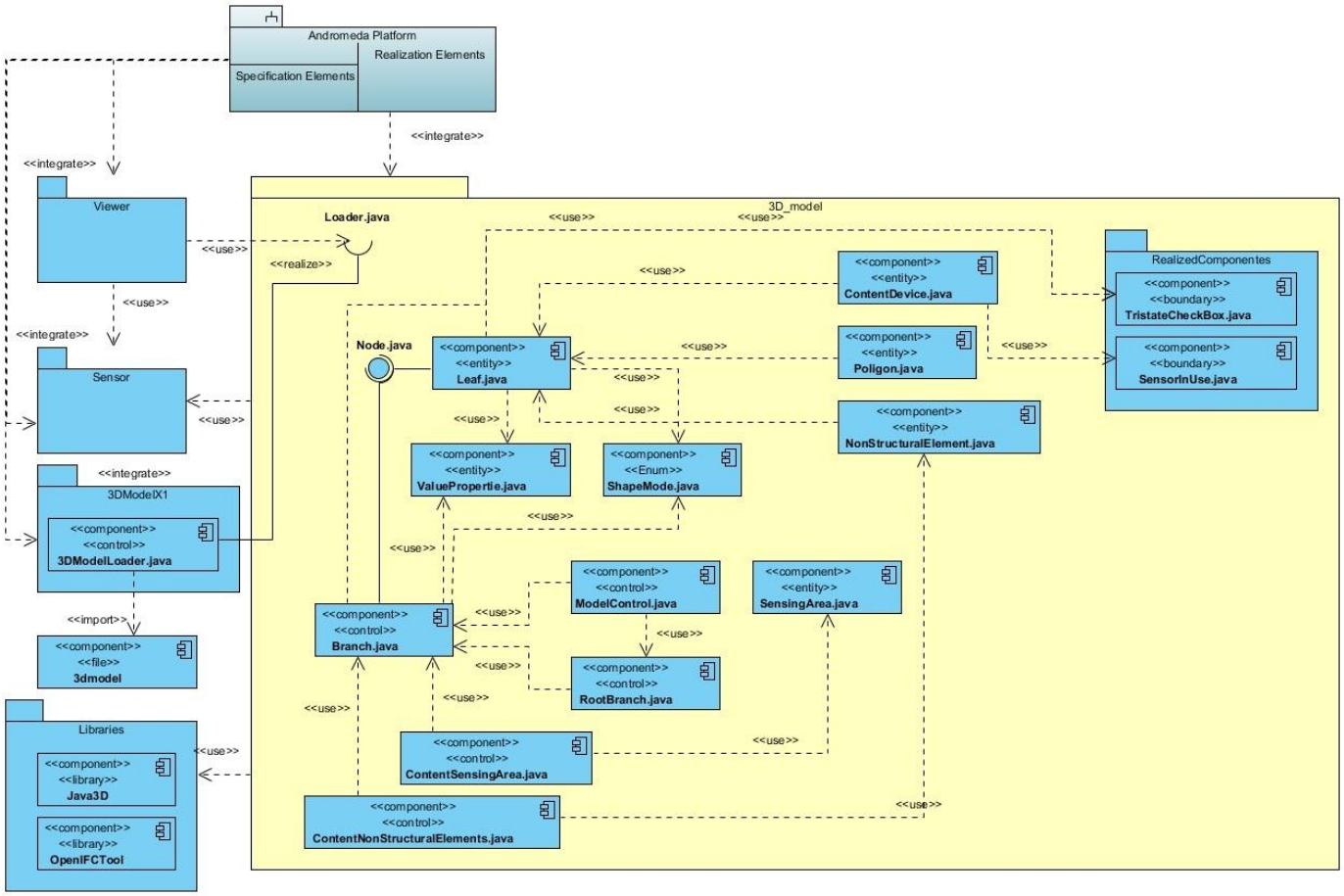

Figura 10. Diagrama de Componentes correspondiente al módulo 3D-model

El módulo contendrá las clases necesarias para cargar los diferentes modelos 3D que vayan a ser utilizados por la herramienta. Está conformado por un conjunto de clases organizadas de forma jerárquica, que son usadas para la visualización en el módulo Viewer.

3.2.6 Diagrama de Componentes: módulo Indoor_model

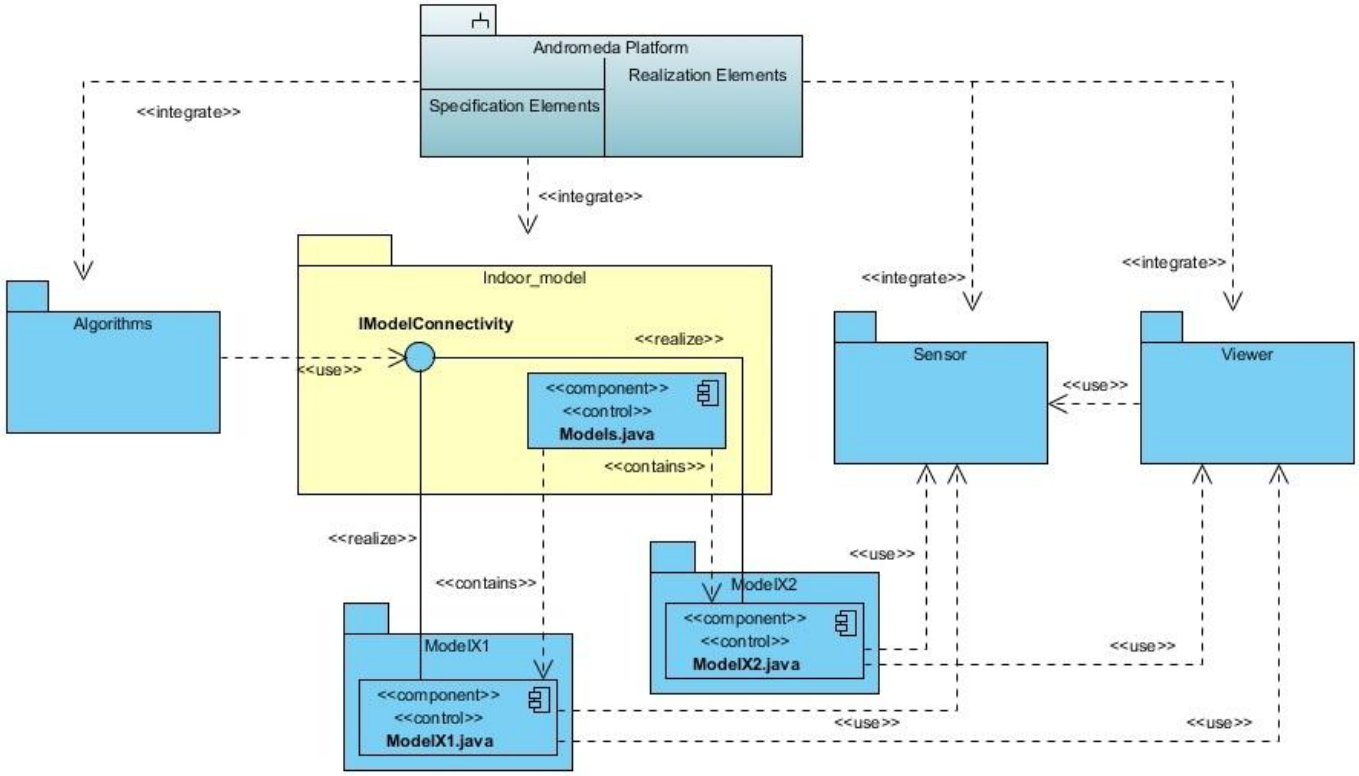

Figura 11. Diagrama de Componentes correspondiente al módulo Indoor_model

El módulo en cuestión contendrá las clases principales que se encargarán de la gestión de los modelos de propagación que empleará aplicación para su funcionamiento. Cada modelo de propagación de señal debe realizar la interfaz **IModelConnectivity**, la cual consta de una función encargada de calcular la conectividad entre los dispositivos a partir de fórmulas matemáticas anteriormente propuestas por otros investigadores para dicho modelo. Esta función será utilizada por el módulo **Algorithms** para comprobar la conectividad entre los diferentes dispositivos a ubicar en el diseño del despliegue. La clase **Models** por su parte, implementará un patrón **Singleton** con el fin de proporcionar en todo momento todos los modelos de propagación con los que cuenta la herramienta para el trabajo.

3.2.7 Diagrama de Componentes: módulo Algorithms

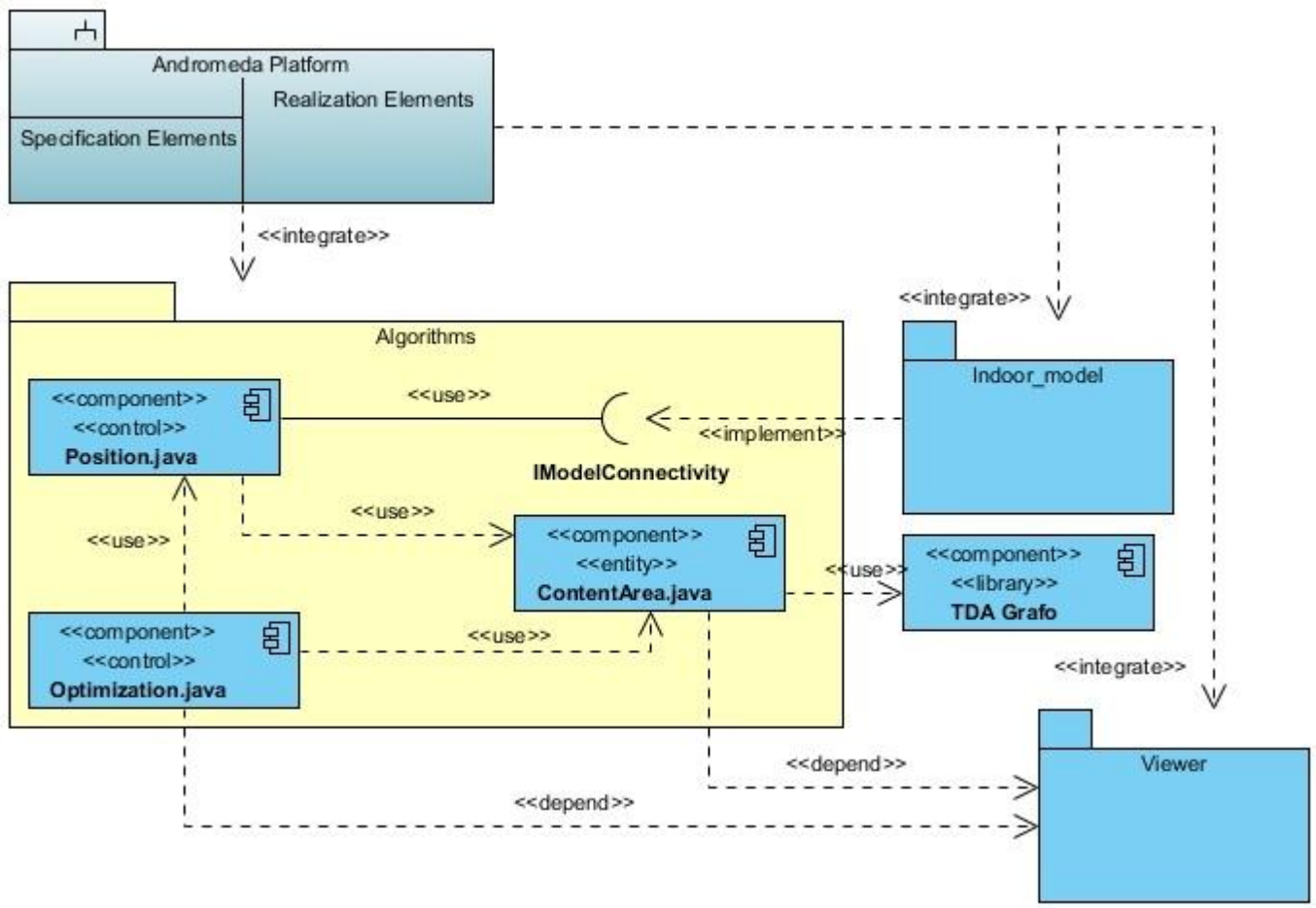

Figura 12. Diagrama de Componentes correspondiente al módulo Algortihms

Este módulo contendrá en esencia dos clases fundamentales para su desempeño. La clase **Position**, mediante un algoritmo de redes neuronales, se encargará de generar todas las posibles ubicaciones de los dispositivos en el diseño de la red. Dichas posiciones son seleccionadas después de haberse comprobado la conectividad a través del uso de la interfaz **IModelConnectivity** realizada por los distintos modelos de propagación de señal de radiofrecuencia que serán empleados.

Por otra parte la clase **Optimization**, depende de dichas posiciones seleccionadas para, mediante un algoritmo de optimización, obtener el diseño más óptimo de la red. Finalizada dicha operación se envían los datos generados al módulo **Viewer** quien se encargará de visualizar el diseño generado por la aplicación.

3.2.7.1 Diagrama de componentes: módulo Restrictions

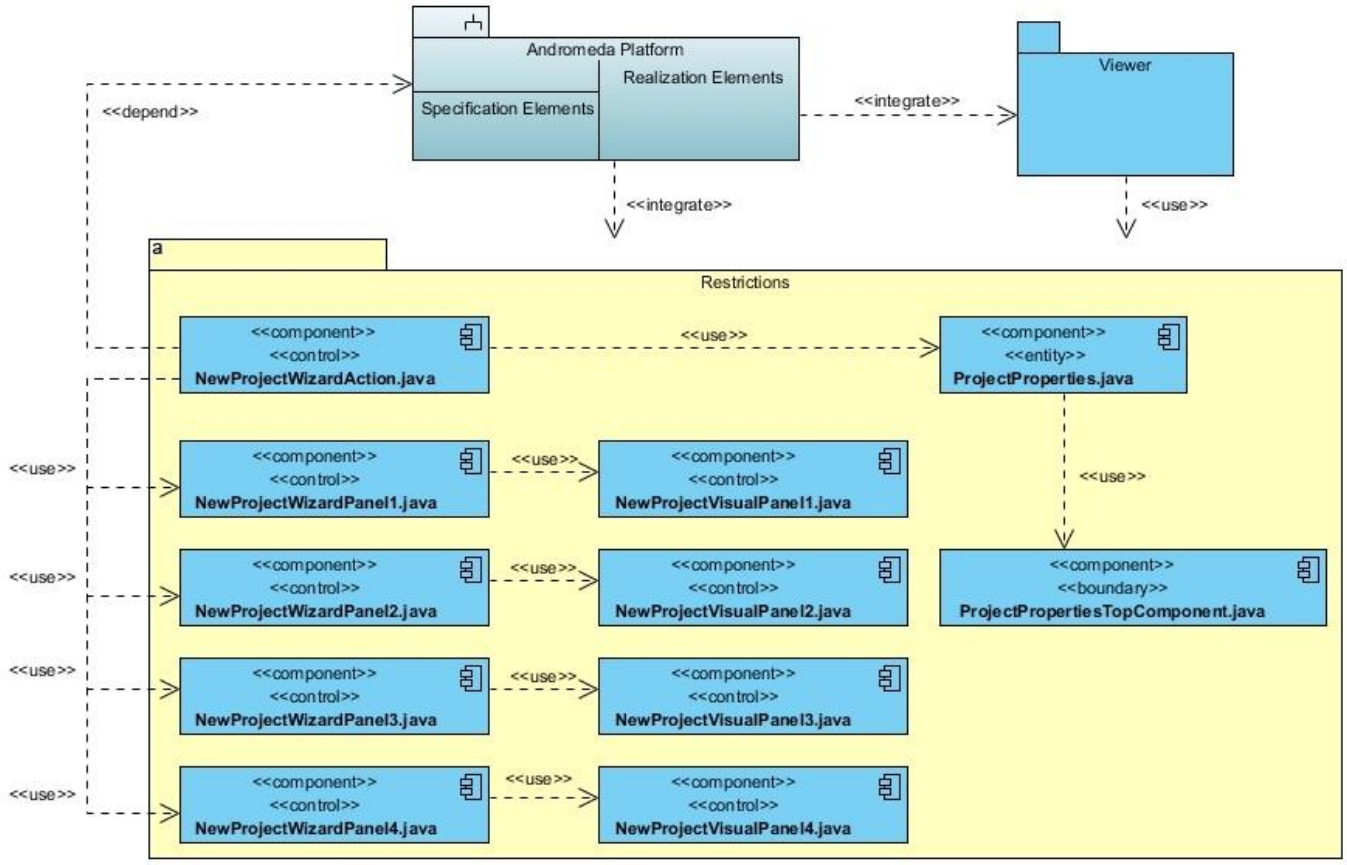

Figura 13. Diagrama de Componentes correspondiente al módulo Restrictions

Este módulo es el encargado de la creación de un nuevo proyecto en la herramienta, insertando para el mismo las restricciones de costo de instalación, mantenimiento, así como la latencia y eficiencia de la red. El modelo 3D, que será empleado en el diseño, es copiado a la carpeta generada por el proyecto y se carga dinámicamente según el tipo de modelo seleccionado en el asistente. Finalmente, todos los datos introducidos en el asisetente conformarán las propiedades que se manejarán en el proyecto.

3.2.8 Diagrama de Paquetes del sistema

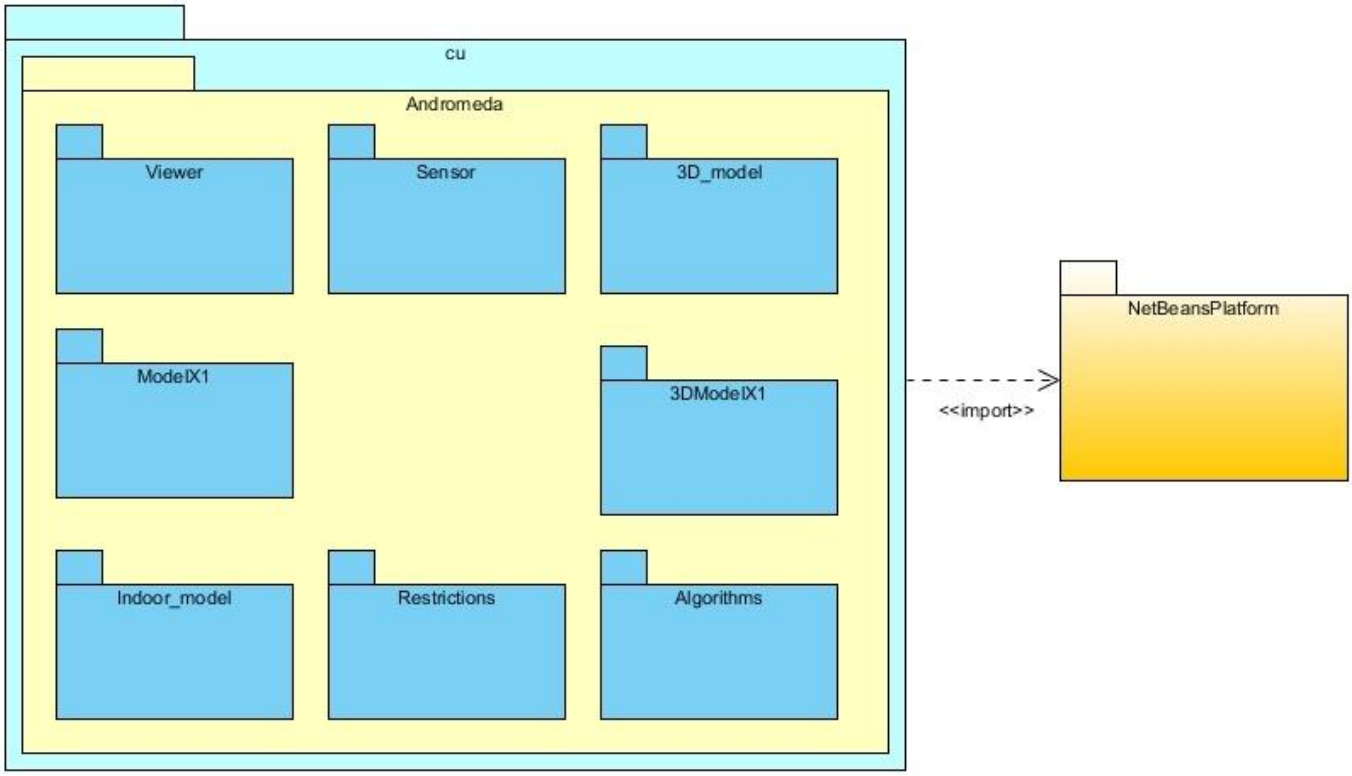

Figura 14. Diagrama de Paquetes del sistema

Los Diagramas de Paquetes se usan para reflejar la organización de paquetes y sus elementos, así como proveer una visualización de los espacios de nombres en las aplicaciones. Uno de los usos más comunes para estos diagramas es organizar los Diagramas de Clase.

Los elementos contenidos en un paquete comparten el mismo espacio de nombres. El hecho de compartir el mismo espacio de nombres garantiza que los elementos contenidos tengan nombres únicos, diferenciando de esta forma las clases y elementos dentro de la herramienta.

3.3 Lineamientos de diseño e implementación

Esta sección contiene las restricciones impuestas por el arquitecto de software para el correcto desarrollo y avance de la solución, tanto desde el punto de vista del diseño, como de la implementación.

3.3.1 Estándares de codificación

Cuando se usa XP como metodología para el desarrollo del software, el código es la forma principal de documentación y comunicación entre los desarrolladores, por lo tanto debe de estar escrito de una manera clara y concisa, que pueda ser identificado fácilmente por cualquier programador de otro equipo. Los estándares de codificación se definen debido a que un estilo de programación homogéneo en un proyecto permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia se le pueda dar mantenimiento de forma eficiente.

En general, un estándar de codificación son reglas que se siguen para la escritura del código fuente. De tal manera que otros programadores se les facilite entender el código (como identificar las variables, las funciones o métodos, etc.).

3.3.1.1 Nomenclatura para la codificación

En la presente solución se usará el estándar de codificación[34] planteado por Sun Microsystems para las aplicaciones escritas en el lenguaje Java [67]. El documento antes mencionado debe ser de absoluto y total conocimiento por parte de los desarrolladores del proyecto. A diferencia del documento estándar, en la aplicación a desarrollar se tomó en consideración que los atributos de cada variable de la clase tuviera como prefijo el caracter "_" y siguiendo la notación camelCase que propone la compañía. El objetivo de esta notación es que los desarrolladores distingan entre las variables locales, las pasadas por parámetro y las propias de la clase. (Ver Anexo 11)

3.3.1.2 Nomenclatura de paquetes de clases

La actual Plataforma NetBeans v7.1 está compuesta por una colección de paquetes escritos en Java. Las especificaciones de los espacios de nombres de los paquetes de la propuesta de solución serán manejadas según las especificaciones de Sun. [68]

El ejemplo a continuación detalla la notación general para la notación de los paquetes perteneciente a la herramienta Andrómeda. El primer nombre del segmento que aparece después de la notación cu.andromeda, será el nombre del sub-proyecto, seguido a continuación del nombre del componente.

[34] Este estándar de codificación está disponible en [http://www.oracle.com/technetwork/java/codeconv-138413.html]

`cu.andromeda.<subproyecto>.<componente>[.*]- Nombres de los paquetes`

3.4 Diseño

XP establece prácticas especializadas que inciden directamente en la realización del diseño para lograr un sistema robusto y reutilizable tratando de mantener su simplicidad, es decir, crear un diseño evolutivo que se va mejorando incrementalmente y que permite hacer entregas pequeñas y frecuentes de valor para el cliente.

3.4.1 Patrones de diseño utilizados

Para el desarrollo de la herramienta Andrómeda se emplearon los patrones de diseño ***Singleton*** y ***Observer***. A continuación se abordará con ejemplos dónde se emplean cada uno de estos.

Patrón Singleton (instancia única):

El uso de este patrón del tipo creacional, garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Una de las clases de la herramienta que implementa dicho patrón es la clase controladora Andromeda como se muestra en la Figura 15. Esta clase contiene la información de los dispositivos a emplear en el diseño, así como la instancia de la clase para gestionar la información de la Base de Datos. El uso de este patrón garantiza que en todo momento se trabaje con la misma instancia de dicha clase en toda la aplicación.

```
/**
 * Devuelve una instancia única de la clase controladora {@link Andromeda}.
 *
 * @return Instancia única de la clase.
 */
public static Andromeda getInstance() {
    if (_instance == null) {
        _instance = new Andromeda();
    }
    return _instance;
}
```

Figura 15. Ejemplo de código que muestra el uso del patrón de diseño Singleton

Patrón Observer (observador):

Este patrón del tipo comportamiento, define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.

Este patrón se ve reflejado en la herramienta de la siguiente forma: La clase Andromeda hereda de la clase Observable y es la encargada de almacenar y notificar a los observadores asociados a ella cada vez que ocurra un cambio en la misma. Ejemplos de los observadores que están asociados a esta clase se encuentran la clase SensorLibraryTopComponent, ExportDevices y DeviceManagement, estas clases visualizan la información de los dispositivos a partir de la clase Observable Andromeda.

3.4.2 Modelo de Datos

Es la descripción de la organización de una base de datos, constituyéndose en una representación gráfica orientada a la obtención de la estructura de datos mediante métodos. En un enfoque más amplio, un Modelo de Datos permite describir los elementos que intervienen en una realidad o en un problema dado y la forma en que se relacionan dichos elementos entre sí. El Modelo de Datos está formado por dos componentes: componente estática, relacionada con el lenguaje de definición de datos (LDD) y dinámica, relacionada con el lenguaje de manipulación de datos (LMD).

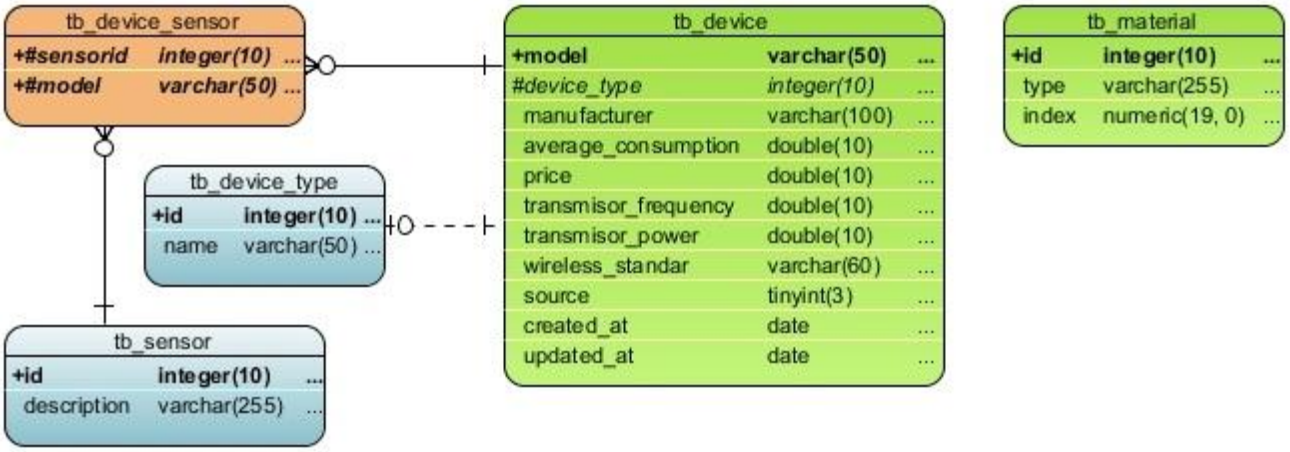

Figura 16. Modelo de Datos de la aplicación

Descripción de las tablas de la Base de Datos de la aplicación

A continuación se muestra la descripción de las tablas que conforman la Base de Datos empleadas por la aplicación para su funcionamiento.

- **tb_device**: Almacena la información persistente de los dispositivos empleados para el trabajo con la herramienta.
- **tb_device_type**: Esta tabla es un nomenclador que almacena los diferentes tipos de dispositivos que son empleados para el trabajo con la herramienta. Los mismos pueden ser *Coordinator, Router y End Device*.
- **tb_sensor**: Nomenclador que almacena los tipos de sensores que pueden estar contenidos dentros de los dispositivos utilizados en el sensado.
- **tb_device_sensor**: Esta tabla representa la relación que existe entre los dispositivos y los sensores correspondientes a cada dispositivo.
- **tb_material**: Almacena la información de los diferentes materiales de los elementos nos estructurales que pueden estar dentro de las edificaciones.

3.5 Implementación

XP propone comenzar las implementación de la solución partiendo de una arquitectura lo más flexible posible para evitar grandes cambios en las próximas iteraciones y en los cambios que generalmente el cliente propone. Como se ha mencionado otras veces en el presente trabajo, la solución en cuestión cuenta con el detalle que es de carácter técnico, por lo que es necesario, desde un inicio, tener bien definida la arquitectura del mismo.

Teniendo en cuenta los análisis y diagramas efectuados anteriormente y una vez trazada la arquitectura del sistema, se procede a realizar la primera iteración.

3.5.1 Iteración 1

El principal objetivo de la primera iteración es desarrollar la Historia de Usuario Nº 1: Gestionar dispositivos mediante una biblioteca. Esta iteración finaliza con varios componentes visuales los cuales se encargan de mostrar los dispositivos que han sido adicionados a la herramienta y los que incluye el proveedor.

Además debe contener un formulario mediante el cual se adicionen nuevos dispositivos que serán almacenados en una Base de Datos. Esta historia de usuario sirve de base para las próximas iteraciones.

Para ello se trazaron las siguientes tareas que se describen a continuación:

- Tarea No. 1: Establecer estándar de código y comentarios.
- Tarea No. 2: Implementar las clases principales del núcleo del módulo.
- Tarea No. 3: Manipular datos persistentes de la Base de Datos.
- Tarea No. 4: Adicionar nuevos dispositivos a la herramienta.
- Tarea No. 5: Modificar o eliminar dispositivos adicionados por el usuario.

Tabla 12. Tarea No. 1: Establecer estándar de código y comentarios

Tarea de Ingeniería	
Número de tarea: 1	Número de Historia de Usuario: 1
Nombre de tarea: Establecer estándar de código y comentarios.	
Tipo de tarea: Configuración	Puntos estimados: 2 días
Fecha inicio: 12 de diciembre de 2011	Fecha fin: 14 de diciembre de 2011
Programador responsable: Henry Rodríguez Estévez	
Descripción: Redactar el estándar de código a utilizar guiándose por los utilizados internacionalmente en proyectos de Java. Consultar documentación en la red para la definición del mismo y los estándares de codificación propuestos por Sun Microsystems.	

Tabla 13. Tarea No. 2: Implementar las clases principales del núcleo del módulo

Tarea de Ingeniería	
Número de tarea: 2	Número de Historia de Usuario: 1
Nombre de tarea: Implementar las clases principales del núcleo del módulo.	
Tipo de tarea: Implementación	Puntos estimados: 3 días.

Fecha inicio: 14 de diciembre de 2011	Fecha fin: 17 de diciembre de 2011
Programador responsable: Pablo Molina Toledo	
Descripción: Implementar las clases principales del núcleo de la herramienta, las cuales serán empleadas para la posterior implementación de las demás funcionalidades. Paquete: cu.andromeda.sensor.core Clases: Andromeda, Device, SensorProperties, SensorType, DeviceType	

Tabla 14. Tarea No. 3: Manipular datos persistentes de la Base de Datos

Tarea de Ingeniería	
Número de tarea: 3	Número de Historia de Usuario: 1
Nombre de tarea: Manipular datos persistentes de la Base de Datos.	
Tipo de tarea: Implementación	Puntos estimados: 6 días.
Fecha inicio: 2 de enero de 2012	Fecha fin: 7 de enero de 2012
Programador responsable: Henry Rodríguez Estévez	
Descripción: Implementar la clase SQLiteManager, necesaria para la manipulación de datos persistentes en la base de datos. La misma debe permitir la inserción, actualización y eliminación de los dispositivos. Confeccionar además, las funcionalidades para visualizar los datos obtenidos en la ventana principal del módulo SensorLibraryTopComponent, a través de la clase controladora Andromeda. Paquete: cu.andromeda.sensor.core, cu.andromeda.sensor Clases: SQLiteManager, Andromeda, SensorLibraryTopComponent, DefaultTableModel, DeviceType, SensorType	

Tabla 15. Tarea No. 4: Adicionar nuevos dispositivos a la herramienta

Tarea de Ingeniería	
Número de tarea: 4	Número de Historia de Usuario: 1
Nombre de tarea: Adicionar nuevos dispositivos a la herramienta.	
Tipo de tarea: Implementación	Puntos estimados: 5 días.
Fecha inicio: 9 de enero de 2012	Fecha fin: 13 enero de 2012
Programador responsable: Pablo Molina Toledo	
Descripción: Confeccionar la interfaz de usuario y las funcionalidades para adicionar nuevos dispositivos a la herramienta y almacenar los mismos en la base de datos. Paquete: cu.andromeda.sensor.core, cu.andromeda.sensor Clases: SQLiteManager, Andromeda, AddDevice, DeviceType, SensorType	

Tabla 16. Tarea No. 5: Modificar o eliminar dispositivos adicionados por el usuario

Tarea de Ingeniería	
Número de tarea: 5	Número de Historia de Usuario: 1
Nombre de tarea: Modificar o eliminar dispositivos adicionados por el usuario.	
Tipo de tarea: Implementación	Puntos estimados: 3 días.
Fecha inicio: 16 enero de 2012	Fecha fin: 19 enero de 2012
Programador responsable: Pablo Molina Toledo	
Descripción: Confeccionar la interfaz de usuario e implementar las funcionalidades para modificar o eliminar solo los dispositivos adicionados por el diseñador a la herramienta. Los cambios deben ser reflejados de forma permanente en la base de datos de la aplicación. Paquete: cu.andromeda.sensor.core, cu.andromeda.sensor, cu.andromeda.sensor.deviceManagement Clases: SQLiteManager, Andromeda, DeviceManagement, ManagementTableModel, DeviceType, SensorType	

3.5.2 Iteración 2

El principal objetivo de esta iteración es realizar las funcionalidades de exportar e importar dispositivos de la Historia de Usuario No. 1. Posteriormente dar cumplimiento a las Historias de Usuario No. 2: Permitir la estructura dinámica del menú; y No. 3: Permitir la incorporación de nuevos modelos de propagación. Esta iteración finaliza con las clases necesarias para incorporar nuevos elementos al menú de la herramienta y la selección de los diferentes modelos de propagación adicionados. Estos modelos de propagación podrán ser empleados posteriormente a la hora de crear las áreas de sensado.

Para ello se trazaron las siguientes tareas que se describen a continuación:

- Tarea No. 6: Implementar la funcionalidad de exportar e importar dispositivos.
- Tarea No. 7: Estudio de la arquitectura de la Plataforma NetBeans.
- Tarea No. 8: Implementar las clases de las acciones del menú de la herramienta.
- Tarea No. 9: Confeccionar las clases e interfaces para la adición de nuevos modelos de propagación.

Tabla 17. Tarea No. 6: Implementar la funcionalidad de exportar e importar dispositivos.

Tarea de Ingeniería	
Número de tarea: 6	Número de Historia de Usuario: 1
Nombre de tarea: Exportar e importar dispositivos.	
Tipo de tarea: Implementación	Puntos estimados: 5 días.
Fecha inicio: 20 enero de 2012	Fecha fin: 27 enero de 2012
Programador responsable: Pablo Molina Toledo	
Descripción: Realizar las clases e interfaces de usuario para llevar a cabo la funcionalidad de exportar los dispositivos que se encuentran en la Base de Datos de la aplicación a un archivo XML. Posteriormente desarrollar un asistente para importar los dispositivos que han sido exportados. Los dispositivos importados deben estar disponibles para su utilización después de haberse adicionados a la herramienta. Paquete: cu.andromeda.sensor.core, cu.andromeda.sensor, cu.andromeda.sensor.importDevices Clases: ExportDevices, ExportTableModel, Andromeda, XMLManager, Device, ImportDevicesWizardAc-	

tion, ImportDevicesVisualPanel1, ImportDevicesVisualPanel2, ImportDevicesWizardPanel1, ImportDevicesWizardPanel2, SQLiteManager, DeviceType, SensorType, Utils

Tabla 18. Tarea No. 7: Estudio de la arquitectura de la Plataforma NetBeans

Tarea de Ingeniería	
Número de tarea: 7	Número de Historia de Usuario: 2, 3
Nombre de tarea: Estudio de la arquitectura de la Plataforma NetBeans.	
Tipo de tarea: Investigación	Puntos estimados: 10 días.
Fecha inicio: 30 enero de 2012	Fecha fin: 11 febrero de 2012
Programador responsable: Henry Rodríguez Estévez	
Descripción: Estudio de la arquitectura de la Plataforma NetBeans para comprender el funcionamiento de la misma y cómo implementar los puntos de extensión de los módulos que provee la plataforma para la agregación dinámica de elementos al menú de la aplicación.	

Tabla 19. Tarea No. 8: Implementar las clases de las acciones del menú de la herramienta

Tarea de Ingeniería	
Número de tarea: 8	Número de Historia de Usuario: 2, 3
Nombre de tarea: Implementar las clases de las acciones del menú de la herramienta.	
Tipo de tarea: Implementación	Puntos estimados: 4 días.
Fecha inicio: 14 febrero de 2012	Fecha fin: 17 de febrero de 2012
Programador responsable: Pablo Molina Toledo	
Descripción: Implementar las clases para los elementos que se incorporarán al menú de la herramienta, correspondientes a cada módulo. Cada clase representa una acción que será incorporada al menú y por tanto debe implementar la interfaz ActionListener del lenguaje.	

Paquete: cu.andromeda.sensor.actions Clases: ExportDevicesAction, AddDeviceAction, ImportDeviceAction, DeviceManagementAction,

Tabla 20. Tarea No. 9: Confeccionar las clases e interfaces para la adición de nuevos modelos de propagación

Tarea de Ingeniería	
Número de tarea: 9	Número de Historia de Usuario: 3
Nombre de tarea: Confeccionar las clases e interfaces para la adición de nuevos modelos de propagación.	
Tipo de tarea: Implementación	Puntos estimados: 2 días.
Fecha inicio: 20 de febrero 2012	Fecha fin: 22 de febrero 2012
Programador responsable: Henry Rodríguez Estévez	
Descripción: Confeccionar la clase controladora que se encargará de la gestión y manipulación de los modelos de propagación, así como crear la interfaz de comunicación que deberá ser implementada por cada uno de los modelos de propagación que se incorporen a la herramienta. El uso de esta interfaz por parte de cada uno de los modelos permitirá de forma dinámica la manipulación y uso de los modelos a la hora de generar una nueva propuesta de diseño. Paquete: cu.andromeda.indoorModel Clases: IModelConnectivity, Models	

3.5.3 Iteración 3

En la presente iteración se le dará cumplimiento a la Historia de Usuario No. 4: Crear un nuevo proyecto y especificar en él las restricciones del diseño. Esta iteración finaliza con la confección de un asistente para la creación de un nuevo proyecto, en el cual se definen los parámetros que el diseñador estima de la red (restricciones) y por último selecciona el modelo 3D que será empleado en el proyecto. Al concluir el asis-

tente se debe visualizar el modelo 3D y las propiedades del proyecto tales como el nombre, ubicación del proyecto en la estación de trabajo y las restricciones antes insertadas.

Para ello se trazaron las siguientes tareas que se describen a continuación:

- Tarea No. 10: Confección de las interfaces de usuario de los paneles del asistente de configuración del proyecto.
- Tarea No. 11: Desarrollar las clases para la manipulación y validación de los paneles del asistente.
- Tarea No. 12: Desarrollar las clases para la gestión y visualización de las propiedades del proyecto.

Tabla 21. Tarea No. 10: Confección de las interfaces de usuario de los paneles del asistente de configuración del proyecto

Tarea de Ingeniería	
Número de tarea: 10	Número de Historia de Usuario: 4
Nombre de tarea: Confección de las interfaces de usuario de los paneles del asistente de configuración del proyecto.	
Tipo de tarea: Implementación	Puntos estimados: 2 días.
Fecha inicio: 23 de febrero 2012	Fecha fin: 25 de febrero 2012
Programador responsable: Pablo Molina Toledo	
Descripción: Confeccionar las interfaces de usuario de los paneles que conforman el asistente para la creación de un nuevo proyecto. En los mismos debe contemplarse un espacio para la inserción de los datos del proyecto, la inserción de las restricciones de la red y la selección del modelo 3D que va a ser usado en el diseño de despliegue. Paquete: cu.andromeda.restrictions Clases: NewProjectVisualPanel1, NewProjectVisualPanel2, NewProjectVisualPanel3, NewProjectVisualPanel4	

Tabla 22. Tarea No. 11: Desarrollar las clases para la manipulación y validación de los paneles del asistente

Tarea de Ingeniería	
Número de tarea: 11	Número de Historia de Usuario: 4
Nombre de tarea: Desarrollar las clases para la manipulación y validación de los paneles del asistente.	
Tipo de tarea: Implementación	Puntos estimados: 4 días.
Fecha inicio: 27 de febrero 2012	Fecha fin: 2 de marzo 2012
Programador responsable: Pablo Molina Toledo	
Descripción: Confeccionar las clases para la gestión de la lógica de negocio de los paneles de visuales del asistente. En los mismos debe estar contemplado el manejo de los datos introducidos por el usuario, así como la validación de los mismos. Paquete: cu.andromeda.restrictions Clases: NewProjectWizardPanel1, NewProjectWizardPanel2, NewProjectWizardPanel3, NewProjectWizardPanel4, NewProjectWizardAction	

Tabla 23. Tarea No. 12: Desarrollar las clases para la gestión y visualización de las propiedades del proyecto

Tarea de Ingeniería	
Número de tarea: 12	Número de Historia de Usuario: 4
Nombre de tarea: Desarrollar las clases para la gestión y visualización de las propiedades del proyecto.	
Tipo de tarea: Implementación	Puntos estimados: 3 días.
Fecha inicio: 5 de marzo 2012	Fecha fin: 8 de marzo 2012
Programador responsable: Henry Rodríguez Estévez	
Descripción: Desarrollar las clases destinadas a visualizar la información procesada al crearse un nuevo	

proyecto e insertarse las restricciones sobre las cuales está acotado el proyecto. Paquete: cu.andromeda.restrictions Clases: ProjectProperties, ProjectPropertiesTopComponent

3.5.4 Iteración 4

Con la conclusión de la presente iteración quedarán sentadas las bases para la realización de la Historia de Usuario No. 5 la cual debe permitir generar la propuesta del diseño de despliegue de la WSAN. La misma concluye con el diagrama de componentes que oriente a los posteriores desarrolladores de la funcionalidad. Además se dejan plasmadas las clases principales para dar respuesta a la generación del diseño de la red a partir de su integración con los demás módulos.

Para dar cumplimiento a esta iteración se definieron las siguientes tareas:

- Tarea No. 13: Definir la interacción del módulo Algorithms con los restantes módulos de la aplicación.
- Tarea No. 14: Confeccionar las clases principales del módulo Algorithms.

Tabla 24. Tarea No. 13: Definir la interacción del módulo Algorithms con los restantes módulos de la aplicación

Tarea de Ingeniería	
Número de tarea: 13	Número de Historia de Usuario: 5
Nombre de tarea: Definir la interacción del módulo Algorithms con los restantes módulos de la aplicación.	
Tipo de tarea: Implementación	Puntos estimados: 5 días.
Fecha inicio: 2 de abril 2012	Fecha fin: 6 de abril 2012
Programador responsable: Henry Rodríguez Estévez	
Descripción: Definir junto al cliente y el equipo de desarrollo cómo se producirá la interacción entre el módulo Algorithms y los restantes módulos de la aplicación para dar respuesta a la funcionalidad a desa-	

rrollar. Módulos: Algorithms, Sensor, Viewer

Tabla 25. Tarea No. 14: Confeccionar las clases principales del módulo Algorithms

Tarea de Ingeniería	
Número de tarea: 14	Número de Historia de Usuario: 5
Nombre de tarea: Confeccionar las clases principales del módulo Algorithms.	
Tipo de tarea: Implementación	Puntos estimados: 5 días.
Fecha inicio: 16 de abril 2012	Fecha fin: 20 de abril 2012
Programador responsable: Pablo Molina Toledo	
Descripción: Desarrollar las clases arquitectónicamente significativas que darán respuesta a la funcionalidad para generar la propuesta de diseño de red y mostrarla al diseñador. Emplear el TDA (Tipo de Dato Abstracto) Grafo para la manipulación de los dispositivos y sus conexiones entre sí, de cada una de las áreas de sensado. Paquete: cu.andromeda.algorithms, cu.andromeda.viewer Clases: GenerateDesignAction, Positions, ContentArea, SensingArea, Optimization	

3.6 **Conclusiones**

Una vez finalizado el presente capítulo, se han cumplido satisfactoriamente la mayoría de las tareas de la investigación planteadas para la realización del trabajo. Se seleccionó el estilo arquitectónico a utilizar en el desarrollo de la herramienta y la tarea más importante en este capítulo, la descripción de la arquitectura, donde a través de los diagramas de UML se muestra una panorámica de la arquitectura.

Capítulo 4. Evaluación y validación de la arquitectura propuesta

La evaluación de la arquitectura es un proceso fundamental para el desarrollo de cualquier software. En el presente capítulo se describe el método de evaluación propuesto para llevar a cabo este proceso, definiendo elementos de vital importancia para el desarrollo de dicha evaluación, entre los que se encuentran las diferentes técnicas e instrumentos utilizados para la evaluación de la arquitectura.

4.1 Evaluación de la arquitectura candidata

Luego de haber investigado sobre los métodos y técnicas existentes para la evaluación de la arquitectura, se tomó la decisión de utilizar el método MECABIC (ver epígrafe 1.2.8.1), utilizando la técnica de validación cualitativa basada en escenarios que propone el mismo. Esta técnica sugiere que se utilice el instrumento Árbol de Utilidades para representar los atributos de calidad relevantes. Es necesario aclarar que para la evaluación de la arquitectura propuesta de la herramienta Andrómeda se modificó el árbol de utilidades que propone el método MECABIC, el cual está basado en el modelo de calidad ISO-9126 instanciado para la arquitectura de software, utilizando como modelo de calidad el ISO/IEC 9126 adaptado para arquitecturas de software que propone Losavio para identificar los atributos de calidad (ver epígrafe 1.2.8.3) que serán reflejados en los niveles del Árbol de Utilidades, debido a que es el mismo estándar que utiliza este método de evaluación.

Siguiendo los pasos del método seleccionado, se presentó el método de validación a utilizar al equipo de desarrollo, donde se explicó el funcionamiento del mismo, las funcionalidades más importantes del sistema, los objetivos del negocio, el contexto relacionado con el proyecto, la arquitectura inicial a ser evaluada y las directrices arquitectónicas (requerimientos de calidad a ser satisfechos por la arquitectura).

Posteriormente, se identificaron y analizaron los elementos del diseño para ver cómo corresponden a los requerimientos de calidad y para determinar cómo influyen sobre la realización de los escenarios de calidad presentes en el Árbol de Utilidades, el cual fue generado y revisado una vez identificados estos requerimientos.

4.1.1 Árbol de Utilidades

El siguiente Árbol de Utilidades representa los atributos de calidad, refinados a los escenarios más importantes para la validación de la arquitectura.

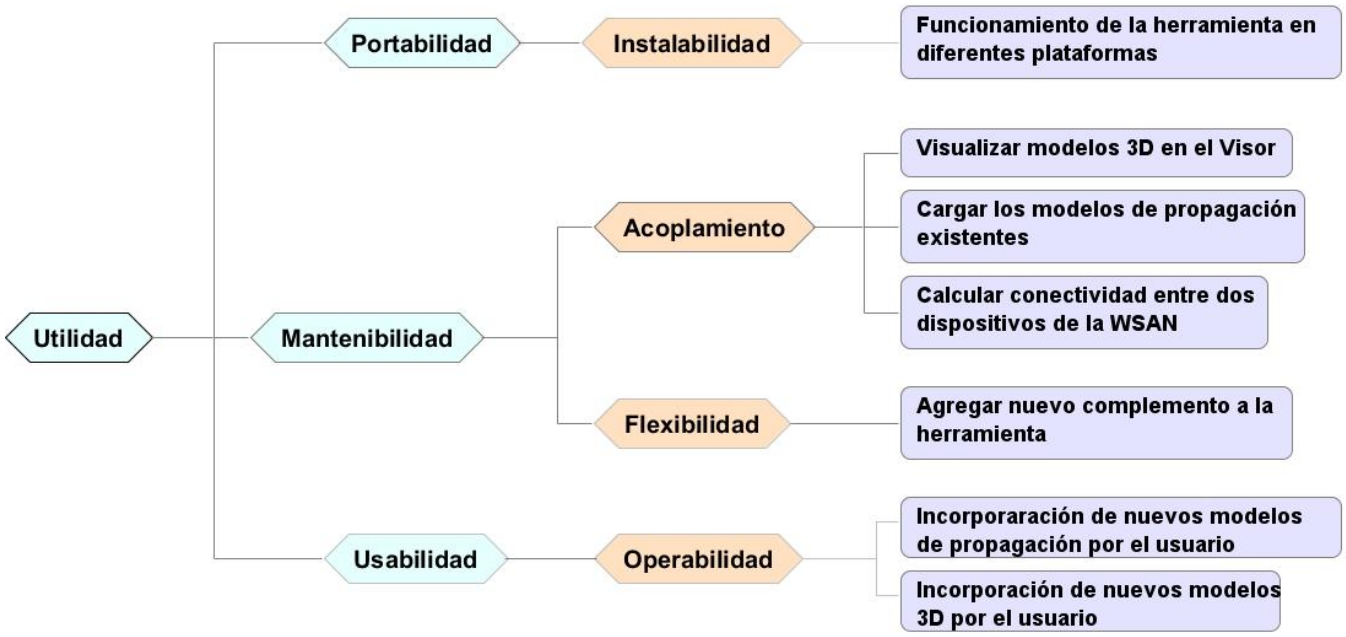

Figura 17. Árbol de Utilidad de la arquitectura de la herramienta Andrómeda

4.1.2 Análisis de los Escenarios

A continuación se realizará el análisis de cada uno de los escenarios correspondientes a cada atributo de calidad empleando la siguiente plantilla utilizada para realizar un análisis desde un enfoque arquitectónico

Tabla 26. Plantilla para el análisis de un enfoque arquitectónico [22]

Escenario #: Número del escenario	Nombre del escenario del Árbol de Utilidades
Atributo	Atributo de calidad a que hace referencia.
Ambiente	Descripción de lo que está pasando en el momento del estímulo.
Estímulo	Parte del escenario que explica o describe lo que el

	interesados hace para iniciar la interacción con el sistema
Respuesta	Indicador de cómo el sistema, a través de su arquitectura, debe responder al estímulo

Decisiones de la Arquitectura	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Decisiones arquitectónicas tomadas en el escenario en correspondencia al atributo de calidad al que hace referencia.	Decisiones arquitecturales que podrán afectar o causar problemas sobre los atributos de calidad ya existentes.	Puntos o componentes que resultan críticos para lograr los atributos de calidad deseados	Decisiones que se toman a nivel arquitectural, que favorecen ciertos atributos de calidad, afectando directamente a otros	Decisiones arquitecturales que son apropiadas en términos de los atributos de calidad a los cuales están "afectando".

Tabla 27. Escenario #1. Visualizar modelos 3D en el Visor

Escenario #1	Visualizar modelos 3D en el Visor
Atributo	Acoplamiento
Ambiente	Carga del modelo a visualizar
Estímulo	Al finalizar el asistente para la creación de un nuevo proyecto
Respuesta	El módulo 3D destinado para el procesamiento del archivo 3D a emplear en el diseño de la WSAN se carga dinámicamente dependiendo del archivo seleccionado. Se construyen las clases para Java 3D y se envían al módulo Visor para su visualización.

Decisiones de la Arquitectura	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Implementar un módulo por cada modelo 3D a		S1		NR1

ser cargado				
Implementar interfaz de comunicación entre el modelo 3D y el módulo Visor		S2		NR2
Utilizar el módulo Visor para visualizar los modelos 3D en el escenario		S3		NR3

Un punto de sensibilidad arquitectónica lo constituye la implementación de un módulo para realizar la carga de cada uno de los modelos 3D con los que cuenta la herramienta para el trabajo. A través de cada uno de ellos se podrán visualizar estos modelos en el Visor, siendo necesario que exista al menos un módulo en la herramienta para el correcto funcionamiento de misma. Esta decisión arquitectónica no constituye un riesgo en la arquitectura de la herramienta debido a que a partir de los módulos incorporados en la misma, es que podrán ser cargados estos modelos 3D, permitiendo así el trabajo con los mismos. Otro punto de sensibilidad se manifiesta a la hora de establecer la comunicación entre el módulo Visor y los modelos de propagación, siendo necesario la implementación de una interfaz que permitiera al Visor el acceso a cada uno de los elementos del modelo 3D para poder visualizarlos. La utilización del módulo Visor se considera otro punto de sensibilidad debido a que este es quien permite la visualización de estos modelos 3D en el escenario. A su vez es catalogado como un no riesgo porque este módulo garantiza la visualización de los elementos de los modelos 3D en el escenario, la generación de las áreas de sensado y la incorporación de los dispositivos de una WSAN en la misma, siendo imprescindible su empleo en la herramienta.

Tabla 28. Escenario #2. Cargar los modelos de propagación existentes

Escenario #2	Cargar los modelos de propagación existentes
Atributo	Acoplamiento
Ambiente	Ejecución de la herramienta
Estímulo	Selección del área de sensado
Respuesta	El uso del componente *Lookup* empleado por la Plataforma NetBeans realiza la búsqueda de los servicios ofrecidos por la interfaz IModelConnectivity realizada por los

	modelos de propagación incorporados en la herramienta y son visualizados a la hora de crear un área de sensado.

Decisiones de la Arquitectura	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Confeccionar una interfaz de comunicación entre la clase Control, que implementa el patrón Singleton, del modelo de propagación y cada uno de los modelos específicos.		S4		NR4

Un punto de sensibilidad encontrado en la arquitectura candidata es la implementación de la interfaz de comunicación entre la clase Control y cada uno de los modelos de propagación específicos, debido a que si se produjera algún cambio en esta interfaz, el módulo Visor no sería capaz de conocer los modelos de propagación existentes en la herramienta. Al mismo tiempo esta decisión arquitectónica es considerada un no riesgo porque el correcto funcionamiento de la misma garantiza que el módulo Visor tenga dominio total de cada modelo 3D con los que cuenta la herramienta para la simulación de la WSAN.

Tabla 29. Escenario #3. Calcular la conectividad entre dos dispositivos de la WSAN

Escenario #3	Calcular la conectividad entre dos dispositivos de la WSAN
Atributo	Acoplamiento
Ambiente	Comunicación entre módulos
Estímulo	Generar la propuesta de diseño de la WSAN
Respuesta	La clase Positions del módulo Algorithms calcula la conectividad entre los dispositivos a partir del modelo de propagación definido en cada área de sensado.

Decisiones de la Arquitectura	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Implementar interfaz de comunicación que debe desarrollar cada modelo de propagación para realizar el cálculo de la conectividad entre los dispositivos pertenecientes al módulo Sensor		S5		NR5

que contienen la información necesaria para realizar dicha operación				

Esta decisión arquitectónica es considerada un punto de sensibilidad debido a que un ligero cambio en esta interfaz pudiera interferir en el cálculo de la conectividad entre dispositivos debido a que esta interfaz permite conocer información como: la distancia entre dispositivos, tipo de material presente en la edificación, potencia de transmisión de los dispositivos, etc. Esta información es suministrada por los módulos Visor, IndoorModel y Sensor. La implementación de esta interfaz asegura el acceso a la información de cada uno de estos módulos, es por eso que también se considera un no riesgo para la arquitectura candidata.

Tabla 30. Escenario #4. Funcionamiento de la herramienta en diferentes plataformas

Escenario #4	Funcionamiento de la herramienta en diferentes plataformas			
Atributo	Instalabilidad			
Ambiente	Instalación			
Estímulo	Instalación de la herramienta			
Respuesta	Se procede a realizar el uso de la herramienta			
Decisiones de la Arquitectura	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Utilizar Java como lenguaje de programación			TR1	NR6
Utilizar NetBeans como IDE de programación				NR7

La utilización de Java como lenguaje de programación constituye un no riesgo debido a que las aplicaciones construidas bajo este lenguaje son independientes de la plataforma del sistema operativo. Esta decisión arquitectónica favorece la portabilidad de la aplicación, sin embargo para la generación de gráficos 3D, este lenguaje necesita la utilización de librerías con un gran número de clases implementadas, ralentizando así el funcionamiento de la misma. El uso de NetBeans como IDE para la programación facilita el proceso de desarrollo de la herramienta pues posee un excelente completamiento de código, detección de errores y ventanas emergentes de documentación, por lo que esta decisión es considerada un no riesgo.

Tabla 31. Escenario #5. Agregar nuevo complemento a la herramienta

Escenario #5	Agregar nuevo complemento a la herramienta			
Atributo	Flexibilidad			
Ambiente	Funcionamiento normal de la herramienta			
Estímulo	Al instalar/desinstalar o activar/ desactivar un módulo			
Respuesta	Se ejecuta la acción que provoco el estimulo			
Decisiones de la Arquitectura	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Hacer uso de la funcionalidad que brinda la Plataforma NetBeans para la gestión de módulos.				NR8

La utilización de la Plataforma NetBeans para la gestión de módulos es considerada un no riesgo porque esta plataforma provee ciertas facilidades para la comunicación entre los componentes y contiene además un módulo que hace mucho más fácil y eficiente instalar o actualizar los módulos, de la aplicación del usuario final.

Tabla 32. Escenario #6. Incorporar nuevo modelo de propagación por el usuario

Escenario #6	Incorporar nuevo modelo de propagación por el usuario			
Atributo	Operabilidad			
Ambiente	Funcionamiento normal de la herramienta			
Estímulo	Agregar un nuevo modelo de propagación			
Respuesta	El Lookup del sistema reconoce dinámicamente el nuevo módulo agregado y lo incorpora a la herramienta			
Decisiones de la Arquitectura	Riesgo	Sensibilidad	Tradeoff	No Riesgo
Utilizar el diseño modular que propone la Plata-			TR2	NR9

forma NetBeans la cual permite exportar e importar nuevos modelos de propagación en forma de *plugin*.				

En el Escenario 6 existe un *tradeoff* pues la Plataforma NetBeans contiene un gran número de clases que ralentizan el proceso de inicio de la herramienta, sin embargo, estas clases facilitan en gran medida el acoplamiento entre módulos. La utilización de la Plataforma NetBeans constituye a su vez un no riesgo pues este brinda todas las clases necesarias para que exista la comunicación entre los módulos.

Tabla 33. Escenario #7. Incorporar nuevo modelo 3D por el usuario

Escenario #7	Incorporar nuevo modelo 3D por el usuario			
Atributo	Operabilidad			
Ambiente	Funcionamiento normal de la herramienta			
Estímulo	Agregar nuevo modelo 3D			
Respuesta	El Lookup del sistema reconoce dinámicamente el nuevo módulo agregado y lo incorpora a la herramienta			
Decisiones de la Arquitectura		Riesgo	Sensibilidad	Tradeoff	No Riesgo
Realizar una clase Interfaz para establecer la comunicación entre el sistema y los modelos de propagación que serán incorporados.			S6		NR10

La utilización de esta interfaz es considerada un punto de sensibilidad porque su incorrecta modificación o ausencia significaría un impacto negativo en la comunicación entre la herramienta y los modelos de propagación existentes en esta. Por otra parte la utilización de esta interfaz constituye un no riesgo porque asegura la interacción entre la herramienta y los modelos de propagación.

4.2 Resultados de la evaluación

Al culminar el proceso de evaluación de la arquitectura se realizó la recapitulación de los pasos del MECABIC y toda la información recogida en cada paso, incluyendo el contexto del negocio, los requerimientos guías y la arquitectura. Se realizó además la documentación de los escenarios, riesgos, no riesgos, puntos de sensibilidad y los *tradeoff* generados por las decisiones arquitectónicas tomadas en cada escenario. La evaluación de la arquitectura demostró que la utilización de la Plataforma NetBeans da cumplimiento a atributos de calidad como: portabilidad, mantenibilidad y la usabilidad de la aplicación, logrando así, uno de los principales objetivos de la investigación. La utilización de esta plataforma posibilitó el desarrollo modular de la arquitectura, la cual permite un bajo acoplamiento entre los componentes del sistema. La selección de Java como lenguaje de programación dotó a la herramienta de portabilidad, debido a las características de este lenguaje, facilitando además el trabajo con los Modelos 3D, ya que presenta librerías libres con buena potencia gráfica.

4.3 Conclusiones

En este capítulo se realizó la evaluación de la arquitectura propuesta a través del método MECABIC, utilizando como instrumento el Árbol de Utilidades y la técnica de validación basada en escenarios. Para la realización de la misma se identificaron los principales atributos de calidad a evaluar junto con sus correspondientes escenarios y los posibles puntos de sensibilidad, riesgos, puntos de negociación y no riesgos, posibilitando el perfeccionamiento de las decisiones arquitectónicas tomadas con anterioridad, comprobando que realmente la aplicación cumple con los atributos de calidad seleccionados.

Conclusiones

Los conocimientos previamente adquiridos sobre Arquitectura de Software y el desarrollo de aplicaciones, sumado a los adquiridos en la presente investigación, hicieron posible realizar:

- La selección de las herramientas y tecnologías a emplear en el proceso de desarrollo de la arquitectura.
- La definición de la arquitectura para la herramienta multiplataforma Andrómeda, haciendo uso de las herramientas y tecnologías estudiadas para su elaboración.
- La capacidad de extensión de la solución, para integrar a la herramienta nuevos modelos de propagación para el cálculo de conectividad en la red, así como la incorporación de nuevos módulos para procesar modelos 3D, permitiéndole ser una sólida herramienta, altamente flexible, lo cual ha sido el principio fundamental durante el proceso de desarrollo de la misma.
- La evaluación de la arquitectura a través del método MECABIC, lo que ayudó a identificar los posibles puntos de riesgo así como constatar que dicha arquitectura fuera óptima y cumpliera con las metas trazadas, contribuyendo a la organización y a tener un mayor grado de confiabilidad en el sistema que se espera obtener como resultado.

Recomendaciones

A partir del trabajo realizado y después de haber analizado los resultados obtenidos se sugiere a los futuros desarrolladores de la herramienta las siguientes recomendaciones que aportarían un mayor valor de uso en la misma.

- Incorporación de una biblioteca para los dispositivos actuadores que puedan ser utilizados en el diseño de la red.
- Desarrollo de un módulo para la gestión de actualizaciones automáticas de la base de datos de los dispositivos de la herramienta. La misma pudiera hacerse a través de un servicio web proporcionado por el sitio oficial de la aplicación en la cual se ofrezcan actualizaciones de los dispositivos.
- Refinar la arquitectura de forma tal que gane en flexibilidad.

Bibliografía referenciada

1. Electrónica, O.I.d.S.d.l. and T.d.l.I.y. Telecomunicaciones (2010) *Redes de sensores. Aplicaciones para control automático de edificios.*, 62.
2. Giannopoulos, N., C. Goumopoulos, and A. Kameas, *Design Guidelines for Building a Wireless Sensor Network for Environmental Monitoring.* Informatics, Panhellenic Conference on, 2009. **0**: p. 148-152.
3. Nourizadeh, S., et al. *Medical and Home Automation Sensor Networks for Senior Citizens Telehomecare*. in *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*. 2009. LORIA Res. Lab., INPL, France.
4. Ma, X., et al. *Supervisory and Energy Management System of large public buildings*. in *Mechatronics and Automation (ICMA), 2010 International Conference on*. 2010.
5. Papadopoulos, N., et al. *A Connected Home Platform and Development Framework for smart home control applications*. in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*. 2009.
6. Reynoso, C.B., *Introducción a la Arquitectura de Software*. 2004: Buenos Aires, Argentina.
7. Bass, L., P. Clements, and K. Bass, *Software Architecture in Practice*. Second Edition ed, ed. S.E. Institute. 2003, Pittsburgh, USA: Addison-Wesley.
8. Garlan, D. and M. Shaw, *An Introduction to Software Architecture*. Vol. 2. 1994, Singapore: World Scientific Publishing Company.
9. Reynoso, C.B. and N. Kiccillof, *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*. 2004: Buenos Aires, Argentina.
10. Camacho, E., F. Cardeso, and G. Nuñez, *Arquitecturas de software. Guía de estudio.* 2004.
11. Bengtsson, P., *Design and Evaluation of Software Architecture*. 1999, Department of Software Engineering and Computer Science: University of Karlskrona/Ronneby.
12. Buschmann, F., et al., *Pattern – Oriented Software Architecture. A System of Patterns*, J.W. Sons, Editor. 1996: Inglaterra.

13. Shaw, M. and P. Clements, *A field guide to Boxology: Preliminary classification of architectural styles for software systems.*, ed. C.S.D.a.S.E. Institute. 1997, Carnegie Mellon University.
14. Burbeck, S. *Application programming in Smalltalk-80: How to use Model-View-Controller (MVC)*. 1992 4/03/1997 [cited 2012 20/02]; Available from: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.
15. Craig, L., *UML y patrones*. 2004: Prentice Hall.
16. Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994: Addison-Wesley Professional. 416.
17. Kruchten, P.B., *The 4+1 View Model of architecture.* Software, IEEE, 1995. **12**(6): p. 42-50.
18. Soni, D., R.L. Nord, and C. Hofmeister, *Software architecture in industrial applications.* ICSE '95: Proceedings of the 17th international conference on Software Engineering, New York, NY, USA, 1995: p. 196-207.
19. Clements, P., et al., *Documenting Software Architectures: Views and Beyond.* Addison Wesley Professional, 2002.
20. Group, I.A.W., *EEE Standard 1471-2000, Recommended practice for Architectural Description of Software-Intensive Systems.* IEEE, 2000: p. 1-23.
21. Carrascoso, Y.A.P., E.C. Gómez, and A.C. Vega. *Procedimiento para la evaluación de arquitecturas de software basadas en componentes.* 2009 [cited 2012 7/05]; Available from: http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.
22. Clements, P., R. Kazman, and M. Klein, *Evaluating Software Architectures*. 2002: Addison-Wesley.
23. González, A., et al., *Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC)*. 2009, Venezuela.
24. Losavio, F., et al., *Quality Characteristics for Software Architecture*, ed. JOT. 2003: JOT.
25. Barrenetxea, G., et al. *The hitchhiker's guide to successful wireless sensor network deployments*. in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. 2008: ACM.
26. Zhang, H. and J.C. Hou, *Maintaining sensing coverage and connectivity in large sensor networks.* NSF International Workshop on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, 2004. **-**: p. .

27. Kar, K. and S. Banerjee. *Node placement for connected coverage in sensor networks*. in *Proceedings of WiOpt*. 2003.
28. Zou, Y. and K. Chakrabarty. *Sensor deployment and target localization based on virtual forces*. in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. 2003.
29. Wang, Y.-C., C.-C. Hu, and Y.-C. Tseng. *Efficient deployment algorithms for ensuring coverage and connectivity of wireless sensor networks*. in *Wireless Internet, 2005. Proceedings. First International Conference on*. 2005.
30. Chang, J.-J., P.-C. Hsiu, and W. Tei. *Search-Oriented Deployment Strategies for Wireless Sensor Networks*. in *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07. 10th IEEE International Symposium on*. 2007. Dept. of Comput. Sci. & Inf. Eng., National Taiwan Univ., Taipei.
31. Guinard, A., A. McGibney, and D. Pesch. *A wireless sensor network design tool to support building energy management*. in *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*. 2009: ACM.
32. Huang, Y.-K., et al. *An Integrated Deployment Tool for ZigBee-Based Wireless Sensor Networks*. in *Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on*. 2008.
33. Jamali, M.a., et al. *An energy-efficient algorithm for connected target coverage problem in wireless sensor networks*. in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*. 2010. Dept. of Comput. Eng., Isalamic Azad Univ., Shabestar, Iran.
34. Mc Gibney, A., M. Klepal, and J.T. Odonnell. *Design of underlying network infrastructure of smart buildings*. in *Intelligent Environments, 2008 IET 4th International Conference on*. 2008: Centre of Adaptive Wireless Systems, Cork Institute of Technology, Ireland.
35. Daintree, *Daintree Networks*. 2010.
36. Ma, Y.-S., et al. *RealSSim: a simulator for indoor sensor network systems*. in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*. 2012: ACM.
37. Fritzke, B. *A Growing Neural Gas Network Learns Topologies*. in *Advances in Neural Information Processing Systems 7*. 1995: MIT Press.
38. Zigbee, *Zigbee Alliance*. 2010.

39. Pinto, A., et al. *Synthesis of embedded networks for building automation and control*. in *American Control Conference, 2008*. 2008.
40. Kohtamaki, T., et al. *PiccSIM Toolchain - design, simulation and automatic implementation of wireless networked control systems*. in *Networking, Sensing and Control, 2009. ICNSC '09. International Conference on*. 2009. Helsinki Univ. of Technol., Helsinki.
41. Shakkottai, S., R. Srikant, and N. Shroff. *Unreliable sensor grids: coverage, connectivity and diameter*. in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. 2005.
42. NetQuest, *NetQuest Software*. 2010.
43. Dhillon, S.S., K. Chakrabarty, and S.S. Iyengar. *Sensor placement for grid coverage under imprecise detections*. in *Information Fusion, 2002. Proceedings of the Fifth International Conference on*. 202. Dept. of Electr. & Comput. Eng., Duke Univ., Durham, NC
44. Española, R.A. *Metodología*. [cited 2012; Available from: http://buscon.rae.es/draeI/SrvltGUIBusUsual?TIPO_HTML=2&TIPO_BUS=3&LEMA=metodolog%C3%ADa.
45. Penadés, M.C. and P.O.L. Torres (2006) *Métodologías ágiles para el desarrollo de software. eXtreme Programming (XP)*. **05**.
46. Serrano, J. *Explicando Scrum a mi abuela*. 2007 9/5/2007 [cited 2012 19/06]; Available from: http://geeks.ms/blogs/jorge/archive/2007/05/09/explicando-scrum-a-mi-abuela.aspx.
47. *Lenguajes Orientado a Objetos - C++*. 2002 [cited 2011 23 octubre]; Available from: http://www.ciao.es/Lenguajes_Orientado_a_Objetos_C__Opinion_524009.
48. Foundation, P.S. *The Python Language Reference*. 2012 12/06/2012 [cited 2011 28/11]; Available from: http://docs.python.org/reference/.
49. Knowlton, J., *Python*. 1 ed. 2009.
50. Software, A. *IMSL C# Numerical Library*. 2005 [cited 2011 16/11]; Available from: http://www.aertia.com/productos.asp?pid=237.
51. Pawlan, M., *Essentials of the Java Programming Language*. 1999.
52. Arnold, K., J. Gosling, and D. Holmes, *Java™ Programming Language*. 4 ed. 2005.
53. Project, M. *Cross platform, open source .NET development framework*. 2012 [cited 2012 8/06]; Available from: http://www.mono-project.com/Main_Page.

54. Project, M. *ECMA Standards*. 2012 [cited 2012 8/06]; Available from: http://www.mono-project.com/ECMA.
55. Bock, H., *The Definitive Guide to NetBeans Platform 7*. 2011, Apress.
56. Vogel, L. *Eclipse RCP Tutorial*. 2012 22/02/2012 [cited 2012 28/02]; Available from: http://www.vogella.com/articles/EclipseRCP/article.html.
57. Tödter, K. and G. Wielenga, *NetBeans Platform Compared with Eclipse Rich Client Platform*, JavaOne, Editor. 2008, Sun Microsystem.
58. Corporation, O. *What's the Difference between NetBeans Platform and Eclipse RCP?* 2008 [cited 2012 22/02]; Available from: http://netbeans.org/features/platform/compare.html.
59. Walls, C., *Spring in Action*. Third Edition ed. 2011.
60. Gallardo, D. and E. Burnette, *Eclipse in Action*. Seven Edition ed, ed. Manning. 2003.
61. SQLite.org. *About SQLite*. 2011 [cited 2011 22/02]; Available from: http://www.sqlite.org/about.html.
62. Tihuilo. *SQLite - Seudo Motor de Base de Datos*. 2011 26/01/2011 [cited 2012 22/02]; Available from: http://www.e-coffeetech.com/articulos/base-de-datos/107-sqlite-seudo-motor-de-base-de-datos.html.
63. Benz, B. and J.R. Durant, *XML Programming Bible*. 2003: Wiley Publishing, Inc.
64. Oracle. *What are the system requirements for Java 6?* . 2011 [cited 2012 12 enero]; Available from: http://www.java.com/en/download/help/sysreq.xml.
65. Fowler, M. *Is Design Dead?* 2001; Available from: www.martinfowler.com/articles/designDead.html.
66. Ambler, S.W. *Agile Architecture: Strategies for Scaling Agile Development*. 2011 [cited 2012 28/02]; Available from: http://www.agilemodeling.com/essays/agileArchitecture.htm.
67. Sun Microsystems, I., *Code Conventions for the Java Programming Language*. 1997, Sun Microsystems, Inc: California, U.S.A.
68. Whiteman, D. and M. Keller. *Naming Conventions*. 2012 24/04/2011 [cited 2012 21/02]; Available from: http://wiki.eclipse.org/Naming_Conventions.
69. Graphisoft. *Graphisoft*. [cited 2012 22/03]; Available from: http://www.graphisoft.es/producto/archicad/.
70. Alliance, O. *The OSGi Architecture*. 2012 [cited 2012 12/06]; Available from: http://www.osgi.org/Technology/WhatIsOSGi.

71. Gutiérrez, J.M. (1999) *Sistemas Expertos Basados en Reglas*.

72. Fielding, R.T. and R. Taylor, *Principled design of the modern Web architecture*, A.T.o.I. Technologies, Editor. 2002.

73. Mitaritonna, A.D., *CAPA-3: Una innovadora metodología para el desarrollo de software en ambientes de trabajo virtuales.*, in *Ingeniería en Sistemas de Información*. 2010, Universidad Tecnológica Nacional Facultad Regional Buenos Aires: Buenos Aires.

74. Didiergeorges, P. *Scrum: La reunión de planificación de Sprint*. 2009 2010 [cited 2012 20/01]; Available from: http://www.didiergeorges.com/blog/scrum-la-reunion-de-planning-de-sprint-1ere-partie.

Bibliografía consultada

1. Abruzzese, D., et al. Long life monitoring of historical monuments via Wireless Sensors Network. in Wireless Communication Systems, 2009. ISWCS 2009. 6th International Symposium on. 2009.

2. Abruzzese, D., et al. Monitoring and vibration risk assessment in cultural heritage via Wireless Sensors Network. in Human System Interactions, 2009. HSI '09. 2nd Conference on. 2009.

3. Akyildiz, I.F. and I.H. Kasimoglu, Wireless sensor and actor networks: research challenges. Ad Hoc Networks, 2004. 2(4): p. 351-367.

4. Akyildiz, I.F., X. Wang, and W. Wang, Wireless mesh networks: a survey. Computer Networks, 2005. 47(4): p. 445-487.

5. Alliance, O. The OSGi Architecture. 2012 [cited 2012 12/06]; Available from: http://www.osgi.org/Technology/WhatIsOSGi.

6. Ambler, S.W. Agile Architecture: Strategies for Scaling Agile Development. 2011 [cited 2012 28/02]; Available from: http://www.agilemodeling.com/essays/agileArchitecture.htm.

7. Ambler, S.W. Artifacts for Agile Modeling: The UML and Beyond. 2011 [cited 2012 28 febrero]; Available from: http://www.agilemodeling.com/essays/modelingTechniques.htm.

8. Anastasi, G., G.L. Re, and M. Ortolani. WSNs for structural health monitoring of historical buildings. in Human System Interactions, 2009. HSI '09. 2nd Conference on. 2009.

9. Arnold, K., J. Gosling, and D. Holmes, Java™ Programming Language. 4 ed. 2005.

10. Bacnet, BACnet Alliance. 2010.

11. Barrenetxea, G., et al. The hitchhiker's guide to successful wireless sensor network deployments. in Proceedings of the 6th ACM conference on Embedded network sensor systems. 2008: ACM.

12. Bass, L., P. Clements, and K. Bass, Software Architecture in Practice. Second Edition ed, ed. S.E. Institute. 2003, Pittsburgh, USA: Addison-Wesley.

13. Bengtsson, P., Design and Evaluation of Software Architecture. 1999, Department of Software Engineering and Computer Science: University of Karlskrona/Ronneby.

14. Benz, B. and J.R. Durant, XML Programming Bible. 2003: Wiley Publishing, Inc.

15. Bock, H., The Definitive Guide to NetBeans Platform 7. 2011, Apress.

16. Booch, G., J. Rumbaugh, and I. Jacobson, El Proceso Unificado de Desarrollo de Software. 2000, Madrid: Editorial Pearson Educación.

17. Braun, J.E. Intelligent Building Systems - Past, Present, and Future. in American Control Conference, 2007. ACC '07. 2007. Purdue Univ., Lafayette.

18. Brennan, R. and D. O'Sullivan, Open Framework Middleware for Intelligent WSN Topology Adaption in Smart Buildings. 2009: IEEE.

19. Brennan, R., et al. Open Framework Middleware for intelligent WSN topology adaption in smart buildings. in Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on. 2009.

20. Burbeck, S. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). 1992 4/03/1997 [cited 2012 20/02]; Available from: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

21. Buschmann, F., et al., Pattern – Oriented Software Architecture. A System of Patterns, J.W. Sons, Editor. 1996: Inglaterra.

22. Camacho, E., F. Cardeso, and G. Nuñez, Arquitecturas de software. Guía de estudio. 2004.

23. Canós, J.H., P. Letelier, and M.d.C. Penadés, Métodologías Ágiles en el Desarrollo de Software, Valencia.

24. Carrascoso, Y.A.P., E.C. Gómez, and A.C. Vega. Procedimiento para la evaluación de arquitecturas de software basadas en componentes. 2009 [cited 2012 7/05]; Available from: http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.

25. Ceriotti, M., et al. Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment. in Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on. 2009.

26. Chang, H.-J. and G.-T. Park. Coordinator assignment scheme and routing algorithm for wireless sensor and actuator networks. in Education Technology and Computer (ICETC), 2010 2nd International Conference on. 2010.

27. Chang, J.-J., P.-C. Hsiu, and W. Tei. Search-Oriented Deployment Strategies for Wireless Sensor Networks. in Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC '07. 10th IEEE International Symposium on. 2007. Dept. of Comput. Sci. & Inf. Eng., National Taiwan Univ., Taipei.

28. Chen, C.W. and Y. Wang, Chain-Type Wireless Sensor Network for Monitoring Long Range Infrastructures: Architecture and Protocols. Int. J. Distrib. Sen. Netw., 2008. 4: p. 287-314.

29. Chen, G.Z. and Z.C. Zhu. Sensor deployment strategy for chain-type wireless underground mine sensor network. in China Univ Mining&Technol. 2008.

30. Chen, W., Y. Sun, and H. Xu. Clustering chain-type topology for wireless underground sensor networks. in Intelligent Control and Automation (WCICA), 2010 8th World Congress on. 2010.

31. Chintalapudi, K., et al., Monitoring civil structures with a wireless sensor network. Internet Computing, IEEE, 2006. 10(2): p. 26-34.

32. Chou, Y. A Seamless City: The Case Study Of Taipei´s WIFI project. in 16th Euro. Regional Conf. 2005.

33. Clarke, J.A., et al., Simulation-assisted control in building energy management systems. Energy and Buildings, 2002. 34(9): p. 933-940.

34. Clements, P., et al., Documenting Software Architectures: Views and Beyond. Addison Wesley Professional, 2002.

35. Clements, P., R. Kazman, and M. Klein, Evaluating Software Architectures. 2002: Addison-Wesley.

36. Corporation, O. What's the Difference between NetBeans Platform and Eclipse RCP? 2008 [cited 2012 22/02]; Available from: http://netbeans.org/features/platform/compare.html.

37. Craig, L., UML y patrones. 2004: Prentice Hall.

38. Daintree, Daintree Networks. 2010.

39. Dhillon, S.S., K. Chakrabarty, and S.S. Iyengar. Sensor placement for grid coverage under imprecise detections. in Information Fusion, 2002. Proceedings of the Fifth International Conference on. 202. Dept. of Electr. & Comput. Eng., Duke Univ., Durham, NC

40. Didiergeorges, P. Scrum: La reunión de planificación de Sprint. 2009 2010 [cited 2012 20/01]; Available from: http://www.didiergeorges.com/blog/scrum-la-reunion-de-planning-de-sprint-1ere-partie.

41. Electrónica, O.I.d.S.d.l. and T.d.l.I.y. Telecomunicaciones (2010) Redes de sensores. Aplicaciones para control automático de edificios., 62.

42. Española, R.A. Metodología. [cited 2012; Available from: http://buscon.rae.es/draeI/SrvltGUIBusUsual?TIPO_HTML=2&TIPO_BUS=3&LEMA=metodolog%C3%ADa.

43. Fairbanks, G., Just Enough Software Architecture. A Risk-Driven Approach. 2010, Marshall & Brainerd.

44. Fielding, R.T. and R. Taylor, Principled design of the modern Web architecture, A.T.o.I. Technologies, Editor. 2002.

45. Foundation, P.S. The Python Language Reference. 2012 12/06/2012 [cited 2011 28/11]; Available from: http://docs.python.org/reference/.

46. Fowler, M. Is Design Dead? 2001; Available from: www.martinfowler.com/articles/designDead.html.

47. Fowler, M., et al., Refactoring: Improving the Design of Existing Code. 1999: Addison-Wesley.

48. Fritzke, B. A Growing Neural Gas Network Learns Topologies. in Advances in Neural Information Processing Systems 7. 1995: MIT Press.

49. Gallardo, D. and E. Burnette, Eclipse in Action. Seven Edition ed, ed. Manning. 2003.

50. Gamma, E., et al., Design Patterns: Elements of Reusable Object-Oriented Software. 1994: Addison-Wesley Professional. 416.

51. Gandomi, A., V.A. Tafti, and H. Ghasemzadeh. Wireless Sensor Networks Modeling and Simulation in Visualsense. in Computer Research and Development, 2010 Second International Conference on. 2010.

52. Gao, G. and K. Whitehouse. The self-programming thermostat: optimizing setback schedules based on home occupancy patterns. in Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings. 2009: ACM.

53. Garey, M.R. and D.S. Johnson, Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences. 1979: WH Freeman and Company, San Francisco, Calif.

54. Garlan, D. and M. Shaw, An Introduction to Software Architecture. Vol. 2. 1994, Singapore: World Scientific Publishing Company.

55. Gascón, D., et al. Experimental Evaluation of Radio Transceivers for Sensor Networks in Harsh Environments. in European Wireless Sensor Network Conference (EWSN2010), Coimbra, Portugal. 2010.

56. Giannopoulos, N., C. Goumopoulos, and A. Kameas, Design Guidelines for Building a Wireless Sensor Network for Environmental Monitoring. 2009, Washington, DC, USA: IEEE Computer Society.

57. Giannopoulos, N., C. Goumopoulos, and A. Kameas, Design Guidelines for Building a Wireless Sensor Network for Environmental Monitoring. Informatics, Panhellenic Conference on, 2009. 0: p. 148-152.

58. Gómez, O.S., Evaluando Arquitecturas de Software. Parte 1. Panorama General. Vol. Vol 1. 2007: Brainworx S.A.

59. González, A., et al., Método de Evaluación de Arquitecturas de Software Basadas en Componentes (MECABIC). 2009, Venezuela.

60. González, A.A., A.M. Núñez, and J.P. García, Diseño de un simulador para redes de sensores, in Sistemas Informáticos. 2009, Universidad Complutense de Madrid: España. p. 69.

61. Gorton, I., Essential Software Architecture. Second Edition ed, ed. Springer. 2011, New York. 260.

62. Graphisoft. Graphisoft. [cited 2012 22/03]; Available from: http://www.graphisoft.es/producto/archicad/.

63. Group, I.A.W., EEE Standard 1471-2000, Recommended practice for Architectural Description of Software-Intensive Systems. IEEE, 2000: p. 1-23.

64. Guinard, A., A. McGibney, and D. Pesch. A wireless sensor network design tool to support building energy management. in Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings. 2009: ACM.

65. Guo, W., W.M. Healy, and M. Zhou. ZigBee-wireless mesh networks for building automation and control. in Networking, Sensing and Control (ICNSC), 2010 International Conference on. 2010.

66. Guo, W. and M. Zhou. An emerging technology for improved building automation control. in Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on. 2009.

67. Gutiérrez, J.M. (1999) Sistemas Expertos Basados en Reglas.

68. Halgamuge, M.N., T.-K. Chan, and P. Mendis. Experiences of Deploying an Indoor Building Sensor Network. in Sensor Technologies and Applications, 2009. SENSORCOMM '09. Third International Conference on. 2009.

69. Huang, Y.-K., et al. An Integrated Deployment Tool for ZigBee-Based Wireless Sensor Networks. in Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on. 2008.

70. IEEE-802. IEEE 802.15 WPAN™ Task Group 4 (TG4). 2012 23 enero 2012 [cited 2012 23 enero]; Available from: http://www.ieee802.org/15/pub/TG4.html.

71. Ieee, IEEE 802.15 $ WPAN^TM$ Task Group 4 (TG4). 2010.

72. Intille, S.S., Designing a home of the future. Pervasive Computing, IEEE, 2002. 1(2): p. 76-82.

73. Jamali, M.a., et al. An energy-efficient algorithm for connected target coverage problem in wireless sensor networks. in Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on. 2010. Dept. of Comput. Eng., Isalamic Azad Univ., Shabestar, Iran.

74. Jiang, X., et al. Experiences with a high-fidelity wireless building energy auditing network. in Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems. 2009: ACM.

75. Johnstone, I., et al. Experiences from a wireless sensor network deployment in a petroleum environment. in Proceedings of the 2007 international conference on Wireless communications and mobile computing. 2007: ACM.

76. Jurdak, R., et al., Octopus: monitoring, visualization, and control of sensor networks. Wireless Communications and Mobile Computing, 2009. -: p. (Print).

77. Kar, K. and S. Banerjee. Node placement for connected coverage in sensor networks. in Proceedings of WiOpt. 2003.

78. Kastner, W., et al., Communication Systems for Building Automation and Control. Proceedings of the IEEE, 2005. 93(6): p. 1178-1203.

79. Kohtamaki, T., et al. PiccSIM Toolchain - design, simulation and automatic implementation of wireless networked control systems. in Networking, Sensing and Control, 2009. ICNSC '09. International Conference on. 2009. Helsinki Univ. of Technol., Helsinki.

80. Kruchten, P.B., The 4+1 View Model of architecture. Software, IEEE, 1995. 12(6): p. 42-50.

81. Lai, Z., et al. On the use of an Intelligent Ray Launching for indoor scenarios. in Antennas and Propagation (EuCAP), 2010 Proceedings of the Fourth European Conference on. 2010.

82. Lee, M.J., et al., Emerging standards for wireless mesh technology. Wireless Communications, IEEE, 2006. 13(2): p. 56-63.

83. Levis, P., et al. TOSSIM: accurate and scalable simulation of entire TinyOS applications. in Proceedings of the 1st international conference on Embedded networked sensor systems. 2003: ACM.

84. Libelium, fabricante de los sensores Waspmote. 2010.

85. Losavio, F., et al., Quality Characteristics for Software Architecture, ed. JOT. 2003: JOT.

86. Lynch, J.P. and K.G. Loh, A Summary Review of Wireless Sensors and Sensor Networks for Structural Health Monitoring. The Shock and Vibration Digest, 2006. 0: p. 91-128.

87. Ma, X., et al. Supervisory and Energy Management System of large public buildings. in Mechatronics and Automation (ICMA), 2010 International Conference on. 2010.

88. Ma, Y.-S., et al. RealSSim: a simulator for indoor sensor network systems. in Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems. 2012: ACM.

89. Martínez, J.A.M., Aplicación de modelado específico de dominio a las redes de sensores inalámbricos, in Tecnologías de la Información y las Comunicaciones. 2007, Universidad Politécnica de Cartagena: Colombia. p. 149.

90. Mascarenas, D., et al., A Mobile Host Approach for Wireless Powering and Interrogation of Structural Health Monitoring Sensor Networks. Sensors Journal, IEEE, 2009. 9(12): p. 1719-1726.

91. Mc Gibney, A., M. Klepal, and J.T. Odonnell. Design of underlying network infrastructure of smart buildings. in Intelligent Environments, 2008 IET 4th International Conference on. 2008: Centre of Adaptive Wireless Systems, Cork Institute of Technology, Ireland.

92. Meijer, E., R. Wa, and J. Gough. Technical Overview of the Common Language Runtime. 2000 [cited 2011 10 de Octubre]; Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.3489.

93. Meza, J.S.M. and J.F.C. Franco, Desarrollo de un protoripo de simulador de un sistema domótico para hogares, basado en redes de protocolo X10, in Ingeniería de Sistemas y Computación. 2010, Universidad Tecnológica de Pereira. p. 199.

94. Microsystems, S. How to Write Doc Comments for the Javadoc Tool. 2000 2004 [cited 2012 21 febrero]; Available from: www.oracle.com/technetwork/java/javase/documentation/index-137868.html.

95. Mitaritonna, A.D., CAPA-3: Una innovadora metodología para el desarrollo de software en ambientes de trabajo virtuales., in Ingeniería en Sistemas de Información. 2010, Universidad Tecnológica Nacional Facultad Regional Buenos Aires: Buenos Aires.

96. Molina, P.F., Diseño de nodos inteligentes para instalaciones domóticas basadas en bus., in Sistemas de Telecomunicación. 2008, Universidad Politécnica de Cataluña: España. p. 89.

97. Montes, T.O. (2012) Redes de sensores inalámbricas.

98. MoteWorks, MoteWorks TM Brochure. 2010.

99. Mukhopadhyay, S.C., A. Gaddam, and G.S. Gupta. Wireless Sensors for Home Monitoring - A Review. in Recent Patents on Electrical Engineering. 2008.

100. Neri, A., et al. Environmental monitoring of heritage buildings. in Environmental, Energy, and Structural Monitoring Systems, 2009. EESMS 2009. IEEE Workshop on. 2009.

101. NetQuest, NetQuest Software. 2010.

102. Nourizadeh, S., et al. Medical and Home Automation Sensor Networks for Senior Citizens Telehomecare. in Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on. 2009. LORIA Res. Lab., INPL, France.

103. Nourizadeh, S., et al., Medical and Home Automation Sensor Networks for Senior Citizens Telehomecare 2009, France: LORIA Res. Lab.

104. Oracle. What are the system requirements for Java 6? . 2011 [cited 2012 12 enero]; Available from: http://www.java.com/en/download/help/sysreq.xml.

105. Ortiz, R. Arquitecturas basadas en eventos para la ayuda a la decisión en sistemas de tráfico rodado. 2008 Mié, 10/09/2008 [cited 2011 20/Octubre]; Available from: http://www.ia.urjc.es/cms/es/node/576.

106. Osterlind, F., et al. Integrating building automation systems and wireless sensor networks. in Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on. 2007.

107. Papadopoulos, N., et al. A Connected Home Platform and Development Framework for smart home control applications. in Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on. 2009.

108. Parbat, B., A.K. Dwivedi, and O.P. Vyas, Article: Data Visualization Tools for WSNs: A Glimpse. International Journal of Computer Applications, 2010. 2(1): p. 14-20.

109. Park, S.C., et al. Implementation of a BACnet-ZigBee gateway. in Industrial Informatics (INDIN), 2010 8th IEEE International Conference on. 2010.

110. Pawlan, M., Essentials of the Java Programming Language. 1999.

111. Penadés, M.C. and P.O.L. Torres (2006) Métodologías ágiles para el desarrollo de software. eXtreme Programming (XP). 05.

112. Pinto, A., et al. Synthesis of embedded networks for building automation and control. in American Control Conference, 2008. 2008.

113. Pinto, J., et al., MonSense--Application for Deployment, Monitoring and Control of Wireless Sensor Networks. ACM Real WSN, 2006. -: p. 1.

114. Project, M. Cross platform, open source .NET development framework. 2012 [cited 2012 8/06]; Available from: http://www.mono-project.com/Main_Page.

115. Project, M. ECMA Standards. 2012 [cited 2012 8/06]; Available from: http://www.mono-project.com/ECMA.

116. Rashid, R.A., et al. Home healthcare via wireless biomedical sensor network. in RF and Microwave Conference, 2008. RFM 2008. IEEE International. 2008.

117. Reynoso, C.B., Introducción a la Arquitectura de Software. 2004: Buenos Aires, Argentina.

118. Reynoso, C.B. and N. Kiccillof, Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. 2004: Buenos Aires, Argentina.

119. Rodríguez, J.I.P., Desarrollo de plugins distribuidos como servicios Web REST, in Ciencia de la Computación. 2010, PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE: Santiago de Chile. p. 57.

120. Serrano, J. Explicando Scrum a mi abuela. 2007 9/5/2007 [cited 2012 19/06]; Available from: http://geeks.ms/blogs/jorge/archive/2007/05/09/explicando-scrum-a-mi-abuela.aspx.

121. Shakkottai, S., R. Srikant, and N. Shroff. Unreliable sensor grids: coverage, connectivity and diameter. in INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies. 2005.

122. Shaw, M. and P. Clements, A field guide to Boxology: Preliminary classification of architectural styles for software systems., ed. C.S.D.a.S.E. Institute. 1997, Carnegie Mellon University.

123. Shu, L., et al. NetTopo: Beyond Simulator and Visualizer for Wireless Sensor Networks. in Future Generation Communication and Networking, 2008. FGCN '08. Second International Conference on. 2008.

124. Smith, D.K. and M. Tardif, Building Information Modeling: a strategic implementation guide for architects, engineers, constructors, and real estate asset managers, ed. amp and J.W. Sons Inc. 2009: John Wiley & Sons Inc.

125. Software, A. IMSL C# Numerical Library. 2005 [cited 2011 16/11]; Available from: http://www.aertia.com/productos.asp?pid=237.

126. Soni, D., R.L. Nord, and C. Hofmeister, Software architecture in industrial applications. ICSE '95: Proceedings of the 17th international conference on Software Engineering, New York, NY, USA, 1995: p. 196-207.

127. SQLite.org. About SQLite. 2011 [cited 2011 22/02]; Available from: http://www.sqlite.org/about.html.

128. Sun Microsystems, I., Code Conventions for the Java Programming Language. 1997, Sun Microsystems, Inc: California, U.S.A.

129. Tac, Wireless Controller Networks for Building Automation. Benefits and opportunities for facility owners. 2010.

130. Tihuilo. SQLite - Seudo Motor de Base de Datos. 2011 26/01/2011 [cited 2012 22/02]; Available from: http://www.e-coffeetech.com/articulos/base-de-datos/107-sqlite-seudo-motor-de-base-de-datos.html.

131. Tödter, K. and G. Wielenga, NetBeans Platform Compared with Eclipse Rich Client Platform, JavaOne, Editor. 2008, Sun Microsystem.

132. Turon, M. MOTE-VIEW: A Sensor Network Monitoring and Management Tool. in Embedded Networked Sensors, 2005. EmNetS-II. The Second IEEE Workshop on. 2005.

133. Vogel, L. Eclipse RCP Tutorial. 2012 22/02/2012 [cited 2012 28/02]; Available from: http://www.vogella.com/articles/EclipseRCP/article.html.

134. Walls, C., Spring in Action. Third Edition ed. 2011.

135. Wang, Y.-C., C.-C. Hu, and Y.-C. Tseng. Efficient deployment algorithms for ensuring coverage and connectivity of wireless sensor networks. in Wireless Internet, 2005. Proceedings. First International Conference on. 2005.

136. Whiteman, D. and M. Keller. Naming Conventions. 2012 24/04/2011 [cited 2012 21/02]; Available from: http://wiki.eclipse.org/Naming_Conventions.

137. WiFi, WiFi Alliance. 2010.

138. Xu, N., et al. A wireless sensor network For structural monitoring. in Proceedings of the 2nd international conference on Embedded networked sensor systems. 2004: ACM.

139. Yamazaki, T. Beyond the Smart Home. in Hybrid Information Technology, 2006. ICHIT '06. International Conference on. 2006.

140. Yeh, L., et al., iPower&58; an energy conservation system for intelligent buildings by wireless sensor networks. Int. J. Sen. Netw., 2009. 5: p. 1-10.

141. Zhang, H. and J.C. Hou, Maintaining sensing coverage and connectivity in large sensor networks. NSF International Workshop on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks, 2004. -: p. .

142. Zigbee, Zigbee Alliance. 2010.

143. Zou, Y. and K. Chakrabarty. Sensor deployment and target localization based on virtual forces. in INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies. 2003.

144. Zuniga, M. and B. Krishnamachari. Analyzing the transitional region in low power wireless links. in Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on. 2004.

Glosario de términos

- **Actuador**: Un actuador es un dispositivo capaz de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre un proceso automatizado.
- **API**: *Application Program Interface* o Interfaz de Programación de Aplicaciones. Es un conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación. Cuando se intenta estandarizar una plataforma, se estipulan unas APIs comunes a los que deben ajustarse todos los desarrolladores de aplicaciones.
- **AWT**: *Abstract Window Toolkit* (en español Kit de Herramientas de Ventana Abstracta) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.
- **BIM:** El concepto de BIM (*Building Information Modeling* o *Modelado de Información para la Edificación,* en su traducción al español) o Edificio Virtual™ de Graphisoft[35], abarca todo el proceso de diseño y gestión de toda la información a lo largo del ciclo de vida del edificio. No sólo es un simple modelo 3D en un ordenador, el Edificio Virtual contiene además con gran detalle información adicional sobre los materiales del edificio y sus características. Es una base de datos tridimensional que hace un seguimiento de todos los elementos que componen el edificio. Esta información puede incluir área y volumen de superficies, propiedades térmicas, descripciones de las habitaciones, precios, información sobre especificaciones del producto, ventanas, puertas y acabados. [69]
- **ECMA**: Organización internacional basada en membresías de estándares para la comunicación y la información. El objetivo de Ecma es desarrollar, en cooperación con las organizaciones de estándares nacionales, europeas e internacionales, estándares y reportes técnicos con el fin de facilitar y estandarizar el uso de las Tecnologías de Información y Comunicación y Dispositivos Electrónicos.
- **Framework**: En el desarrollo de software, un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software

[35]Graphisoft es pionero y líder en el desarrollo de soluciones de Virtual ™ Building.

concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas, un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio, además, provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio.

- **IDE**: Es un Entorno de Desarrollo Integrado (IDE, por sus siglas en Inglés) de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.
- **IFC**: *Industry Foundation Classes* (IFC) es un formato que incluye elementos constructivos como ventanas, puertas, muros, losas, entre otras. Está basado en la idea de objetos que describen de forma total el Edificio Virtual. Estos objetos están definidos para dar continuidad al ciclo de vida del proyecto desde su concepción hasta las etapas de diseño, documentación y construcción. Este archivo está disponible a todos los participantes de la industria de la construcción, para su uso en todos los países, aprovechando este lenguaje común para compartir elementos inteligentes del Edificio Virtual con otras disciplinas. Es un formato de uso general para BIM, siendo una especificación abierta y no está controlada por un único proveedor.
- **JFace**: es un conjunto de widgets para realizar interfaces de usuario construido sobre SWT. Proporciona una serie de construcciones muy frecuentes a la hora de desarrollar interfaces gráficas de usuario, tales como cuadros de diálogo, evitando al programador la tediosa tarea de lidiar manualmente con los widgets de SWT.
- **Modelo de Propagación**: Un modelo de propagación es un conjunto de expresiones matemáticas, diagramas y algoritmos utilizados para representar las características de la propagación de la RF en un determinado entorno.
- **OSGi**: La tecnología OSGi es un conjunto de especificaciones que definen un sistema de componentes dinámicos para Java. Estas especificaciones permiten un modelo de desarrollo donde las aplicaciones son dinámicas, integrado por diferentes componentes reutilizables. Estas especificaciones reducen la complejidad del software, proporcionando una arquitectura modular para grandes sistemas distribuidos, así como aplicaciones pequeñas y embebidas. [70]

- **Sensor**: dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas.
- **SWT**: *Standard Widget Toolkit*, es conjunto de componentes para construir interfaces gráficas en Java. Recupera la idea original de la biblioteca AWT de utilizar componentes nativos, con lo que adopta un estilo más consistente en todas las plataformas, pero evita caer en las limitaciones de ésta.
- **Swing**: Biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

Anexos

Anexo 1. Relación de abstracción entre estilos y patrones

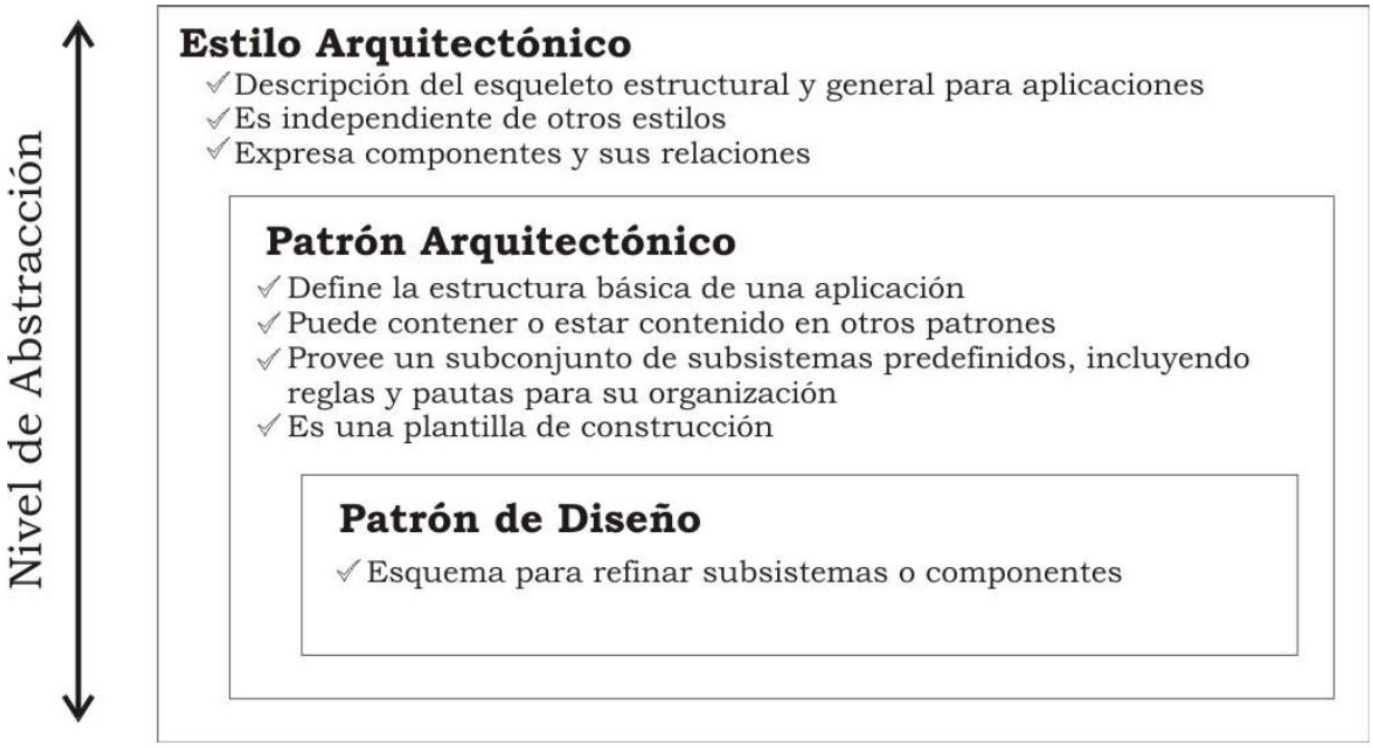

Figura 18. Relación de abstracción entre estilos y patrones

Anexo 2. Estilos y patrones arquitectónicos

1. Sistemas de flujo de datos

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. [9]

Tuberías y Filtros

Históricamente se relaciona con las redes de proceso descritas por Kahn hacia 1974. Su nombre ha prevalecido a lo largo del tiempo pues, si bien dichos estilos no realizan tareas de filtrado, como la eliminación

de campos o registros; si ejecutan formas variables de transformación, una de las cuales puede ser el filtrado.

Este estilo enfatiza en la transformación incremental de los datos por sucesivos componentes. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas.

Dicho estilo se debe usar cuando:

- Se puede especificar la secuencia de un número conocido de pasos.
- No se requiere esperar la respuesta asincrónica de cada paso.
- Los componentes situados posteriores al componente actual deben ser capaces de interactuar con los datos que vienen de los componentes superiores.

Entre sus ventajas destacan su fácil entendimiento e implementación y que fuerza un procesamiento secuencial del programa. Sin embargo no maneja con demasiada eficiencia las construcciones condicionales, bucles y otras lógicas de control de flujo, pueden llegar a requerirse buffers de tamaño indefinido; y no es apto para manejar situaciones interactivas, sobre todo cuando se requieren actualizaciones incrementales en la representación de la pantalla.

2. Estilos centrados en datos

Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Los estilos de bases de datos, sistemas de hipertexto y pizarras o repositorios forman parte de esta categoría de estilos.

Arquitecturas de Pizarra o Repositorio

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él [8]. A este estilo a su vez se le han definido dos sub-categorías:

- Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional.
- Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control. [9]

Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, de habla, etc.), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados. Además, recientemente está siendo utilizado en exploraciones de inteligencia artificial distribuida o cooperativa, en robótica, en modelos multi-agentes y en programación evolutiva.

3. Estilos de código móvil

Este grupo de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico.

Sistemas Basados en Reglas

Son uno de los modelos de representación del conocimiento más ampliamente utilizados. Esto es debido a que resultan muy apropiados en situaciones en las que el conocimiento que se desea representar surge de forma natural con estructura de reglas. Uno de los paradigmas de programación tradicionalmente asociado a los sistemas basados en reglas es el de los sistemas de producción.

Estos sistemas se usan cuando el dominio del problema es estrecho, es decir, es un dominio en el que se comprende bien la teoría. Además, el conocimiento debe poder representarse mediante hechos y reglas. [71]

Estos estilos son muy usados en aplicaciones del campo de la Inteligencia Artificial, donde la representación del conocimiento juega un papel primordial.

4. Estilos Heterogéneos

Es una de las familias de estilos a la que más se le ha hecho referencia en los últimos tiempos. En este apartado podrían agregarse formas que aparecen esporádicamente en los censos de estilos, como los sistemas de control de procesos industriales, sistemas de transición de estados, arquitecturas específicas de dominios [8] o estilos derivados de otros estilos, como es el caso de REST.

Sistemas de Control de Procesos

Estos sistemas se caracterizan no sólo por los tipos de componentes, sino por las relaciones que mantienen entre ellos. El objetivo de un sistema de esta clase es mantener ciertos valores dentro de ciertos ran-

gos especificados, llamados puntos fijos o valores de calibración; el caso más clásico es el de los termostatos. [9]

La ventaja fundamental de este estilo radica en su elasticidad ante perturbaciones externas.

Arquitecturas Basadas en Atributos

Además de especificar los habituales componentes y conectores, los estilos basados en atributos incluyen atributos de calidad específicos que declaran el comportamiento de los componentes en interacción. En realidad no tipifica como un estilo en estado puro, sino como una asociación entre la idea de estilo con análisis arquitectónico y atributos de calidad.

En resumen el estilo pretende lograr que la Arquitectura de Software estuviera más cerca de ser una disciplina de ingeniería, aportando el beneficio esencial de la ingeniería (la predictibilidad) al diseño arquitectónico.

5. Estilos Peer – to – Peer

También llamado estilos de componentes independientes. En resumen enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante difusión (*broadcast*).

Arquitecturas Basadas en Eventos

Estas arquitecturas se vinculan con sistemas basados en actores y redes de conmutación de paquetes (publicación-suscripción). Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa (como se haría en un estilo orientado a objetos) un componente puede emitir mediante difusión uno o más eventos. [9]

Este modelo de eventos puede utilizarse cuando: se desea manejar independientemente y de forma aislada diversas implementaciones de una función específica; y cuando las respuestas de una implementación no afectan la forma en que trabajan otras implementaciones.

Las ventajas que proveen el uso de este estilo señalan que se optimiza el mantenimiento de los procesos de negocios que no están relacionados (son independientes). Permite el desarrollo en paralelo, lo que puede resultar en mejoras de performance. Por último sus implementaciones pueden ser sincrónica o asincrónicamente porque no siempre se espera una respuesta.

Desafortunadamente este estilo no permite construir respuestas complejas a funciones de negocios y debido a que su comunicación suele ser asincrónica un componente no puede utilizar los datos o el estado de otro componente para efectuar su tarea, debido a que no se debe mantener una dependencia entre los mismos. [9]

Arquitecturas Orientadas a Servicios

Estas arquitecturas en los últimos años han recibido un tratamiento intensivo en el campo de exploración de los estilos. Al mismo tiempo se percibe una tendencia a promoverlas de un sub-estilo propio de las configuraciones distribuidas. Esto es debido al compás de las predicciones y pronósticos de convertirse en la tendencia que habrá de ser dominante en este milenio. Este predominio no se funda en la idea de servicios en general, comunicados de cualquier manera, sino que más específicamente va de la mano de la expansión de los *Web Services* basados en XML, en los cuales los formatos de intercambio se basan en XML 1.0 *Namespaces* y el protocolo de elección es SOAP [9] comunicándose siempre sobre un transporte que por defecto es HTTP.

Un *Web Service* es un sistema de software diseñado para soportar interacción máquina – máquina sobre una red. Posee una interfaz descrita en un formato procesable por máquina (específicamente WSDL).

Desde el punto de vista arquitectónico los servicios son entidades de software que encapsulan funcionalidades de negocios y proporcionan dicha funcionalidades a otras entidades a través de interfaces públicas bien definidas. Por su parte los componentes del estilo (los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen, mientras que la funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran. Los servicios utilizan una UDDI[36], mediante la cual se puede requerir un servicio para descubrirlo y/o utilizarlo dinámicamente.

[36] Siglas del catálogo de negocios de Internet denominado *Universal Description, Discovery and Integration*.

Arquitecturas Basadas en Recursos

La caracterización más detallada del estilo denominado *Representational State Transfer* (en lo adelante REST). La literatura especializada tiende a considerar a REST una variante menor de las Arquitecturas basadas en Servicios, Fielding considera que REST resulta de la composición de varios estilos más básicos, incluyendo repositorio replicado, cache, cliente-servidor, sistema en capas, sistema sin estado, máquina virtual, código a demanda e interfaz uniforme. [72]

En síntesis, podría decirse que REST define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El ejemplo de referencia es nada menos que la Web, donde los URIs[37] identifican los recursos y HTTP es el protocolo de acceso para el intercambio.

Anexo 3. Patrones de responsabilidad

Experto

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen.

Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debe conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento.

[37] Cadena de caracteres corta que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente estos recursos son accesibles en una red o sistema.

Bajo acoplamiento

Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases.

Alta cohesión

Mantiene la complejidad dentro de límites manejables, lo que significa que asigna una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

Polimorfismo

El polimorfismo tiene varios sentidos. Dentro de este contexto significa asignar el mismo nombre a servicios en varios objetos, cuando los servicios se parecen o estén relacionados entre ellos. Los tipos de objetos suelen estar relacionados en una jerarquía con una superclase común. [15]

Anexo 4. Descripción de atributos de calidad observables vía ejecución

Tabla 34. Descripción de atributos de calidad observables vía ejecución. [57]

Atributo de calidad	Descripción
Disponibilidad *(Availability)*	Es la medida de disponibilidad del sistema para el uso.
Confidencialidad *(Confidentiality)*	Es la ausencia de acceso no autorizado a la información.
Funcionalidad *(Functionality)*	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño *(Performance)*	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones

	dadas, como velocidad, exactitud o uso de memoria. Se refiere a aspectos temporales del comportamiento del mismo. Se refiere a capacidad de respuesta, ya sea el tiempo requerido para responder a aspectos específicos o el número de eventos procesados en un intervalo de tiempo.
Confiabilidad *(Reliability)*	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
Seguridad externa *(Safety)*	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad interna *(Security)*	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

Anexo 5. Descripción de atributos de calidad no observables vía ejecución

Tabla 35. Descripción de atributos de calidad no observables vía ejecución [57]

Atributo de Calidad	**Descripción**
Configurabilidad *(Configurability)*	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
Integrabilidad *(Integrability)*	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad *(Integrity)*	Es la ausencia de alteraciones inapropiadas de la información.

Interoperabilidad *(Interoperability)*	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de *integrabilidad.*
Modificabilidad *(Modifiability)*	Es la habilidad de realizar cambios futuros al sistema.
Mantenibilidad *(Maintainability)*	Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
Portabilidad *(Portability)*	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
Reusabilidad *(Reusability)*	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
Escalabilidad *(Scalability)*	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
Capacidad de Prueba *(Testability)*	Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

Anexo 6. Atributos de calidad y sus sub-características del modelo ISO/IEC 9126 adaptado

Tabla 36. Atributos de calidad y sus subcaracterísticas del modelo ISO/IEC 9126 adaptado

Características	Subcaracterísticas	Elementos arquitectónicos
Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia.
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos.
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos.
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea.
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de software para manejar excepciones.
	Recuperabilidad	Existencia de mecanismos o dispositivos de software para restablecer el nivel de desempeño y recuperar datos.
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución. Eficiencia para una funcionalidad.
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo.
Mantenibilidad	Acoplamiento	Interacciones entre componentes.
	Modularidad	Número de componentes que dependen de un componente.

Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación.
	Instalabilidad	Presencia de mecanismos de instalación.
	Coexistencia	Presencia de mecanismos que faciliten la co-existencia.
	Reemplazabilidad	Lista de componentes reemplazables para cada componente.

Anexo 7. Fases de desarrollo de RUP

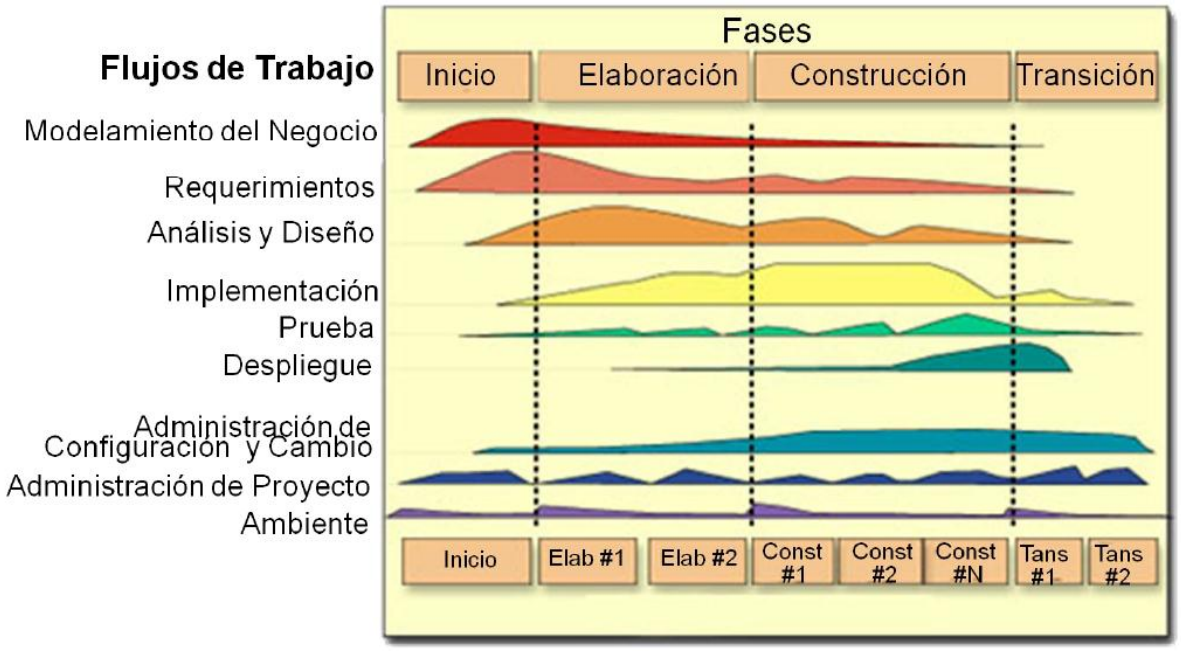

Figura 19. Fases de desarrollo de RUP [73]

Anexo 8. Ciclo de desarrollo de XP

Figura 20. Ciclo de desarrollo de XP

Anexo 9. Ciclo de desarrollo de SCRUM

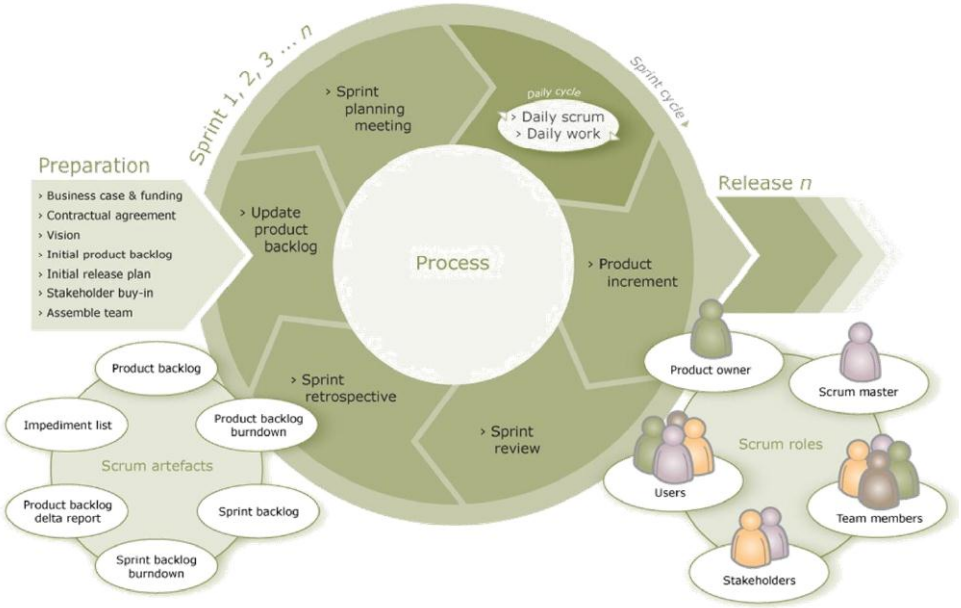

Figura 21. Ciclo de desarrollo de SCRUM [74]

Anexo 10. Diagrama de Gantt para cada una de las iteraciones

Task Name	Start	Finish
Primera Iteración	**12/12/2011**	**19/01/2012**
Establecer estándar de código y comentarios.	12/12/2011	14/12/2011
Implementar las clases principales del núcleo del módulo.	14/12/2011	17/12/2011
Manipular datos persistentes de la Base de Datos.	02/01/2012	07/01/2012
Adicionar nuevos dispositivos a la herramienta.	09/01/2012	13/01/2012
Modificar o eliminar dispositivos adicionados por el usuario.	16/01/2012	19/01/2012

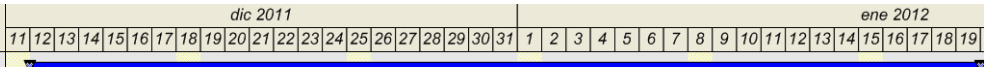

Figura 22. Diagrama de Gantt de la primera iteración

Task Name	Start	Finish
Segunda Iteración	**20/01/2012**	**22/02/2012**
Exportar e importar dispositivos.	20/01/2012	27/01/2012
Estudio de la arquitectura de la plataforma NetBeans.	30/01/2012	11/02/2012
Implementar las clases de las acciones del menú de la herramienta.	14/02/2012	17/02/2012
Confeccionar las clases e interfaces para la adición de nuevos modelos de propagación.	20/02/2012	22/02/2012

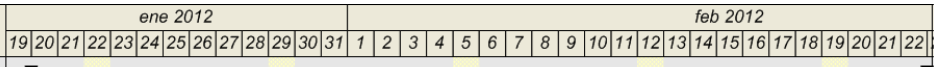

Figura 23. Diagrama de Gantt de la segunda iteración

Task Name	Start	Finish
Tercera Iteración	**23/02/2012**	**08/03/2012**
Confección de las interfaces de usuario de los paneles del asistente de configuración del proyecto.	23/02/2012	25/02/2012
Desarrollar las clases para la manipulación y validación de los paneles del asistente	27/02/2012	02/03/2012
Desarrollar las clases para la gestión y visualización de las propiedades del proyecto	05/03/2012	08/03/2012

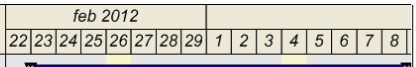

Figura 24. Diagrama de Gantt de la tercera iteración

Task Name	Start	Finish
Cuarta Iteración	**02/04/2012**	**20/04/2012**
Definir la interacción del módulo Algorithms con los restantes módulos de la aplicación.	02/04/2012	06/04/2012
Confeccionar las clases principales del módulo Algorithms	16/04/2012	20/04/2012

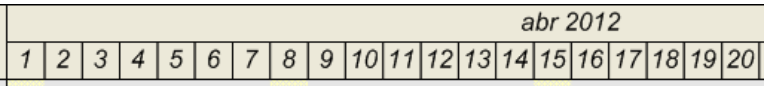

Figura 25. Diagrama de Gantt de la cuarta iteración

Anexo 11. Ejemplo de codificación empleada en el desarrollo

```
package cu.andromeda.sensor.core;

import java.io.File;
import java.sql.Date;
import java.sql.*;
import java.util.*;

/**
 * Clase que se encarga del manejo de la base de datos SQLite que contiene la
 * información de los sensores administrados por la applicación.
 *
 * @author Pablo
 * @version 1.0
 * @since 1.0
 */
public class SQLiteManager {

    /**
     * Variable que contiene la ruta relativa de la Base de Datos.
     */
    public static final String DB_PATH = System.getProperty("user.dir") + File.separatorChar + "base.db";
    /**
     * Variable encargada de manejar la información y conexión a la Base de
     * Datos.
     */
    private static Connection _connection = null;
    /**
     * Variable utilizada para realizar el flujo de información hacia el archivo
     * de la Base de Datos.
     */
    private static File _file;
    /**
     * Arreglo con los sensores del sistema.
     */
    private static SensorType[] _types;
    private static DeviceType[] _deviceTypes;

    /**
     * Crea un nuevo objeto de la clase para manejar el flujo de datos hacia la
     * base de datos.
     */
    public SQLiteManager() {
```

```java
/**
 * Método para insertar los sensores para un dispositivo determinado en la
 * tabla.
 *
 * @param d
 * @throws ClassNotFoundException
 * @throws SQLException
 */
private void insertSensorsDivice(Device d) throws ClassNotFoundException, SQLException {

    getConnection().setAutoCommit(false);

    String sql = "INSERT INTO tb_device_sensor (sensorid, model) VALUES (?, ?);";
    PreparedStatement statementInsert = getConnection().prepareStatement(sql);

    int cant = d.getSensorsIncluding().size();
    for (int i = 0; i < cant; i++) {
        statementInsert.setInt(1, d.getSensorsIncluding().get(i).ordinal());
        statementInsert.setString(2, d.getModel());

        //Adicionar la consulta a la lista
        statementInsert.addBatch();
    }

    //ejecutar la consulta
    statementInsert.executeBatch();
    statementInsert.close();

    getConnection().setAutoCommit(true);
}
```

Figura 26. Ejemplos de la codificación utilizada en el desarrollo

Anexo 12. Diagrama de clases ampliado del módulo Sensor

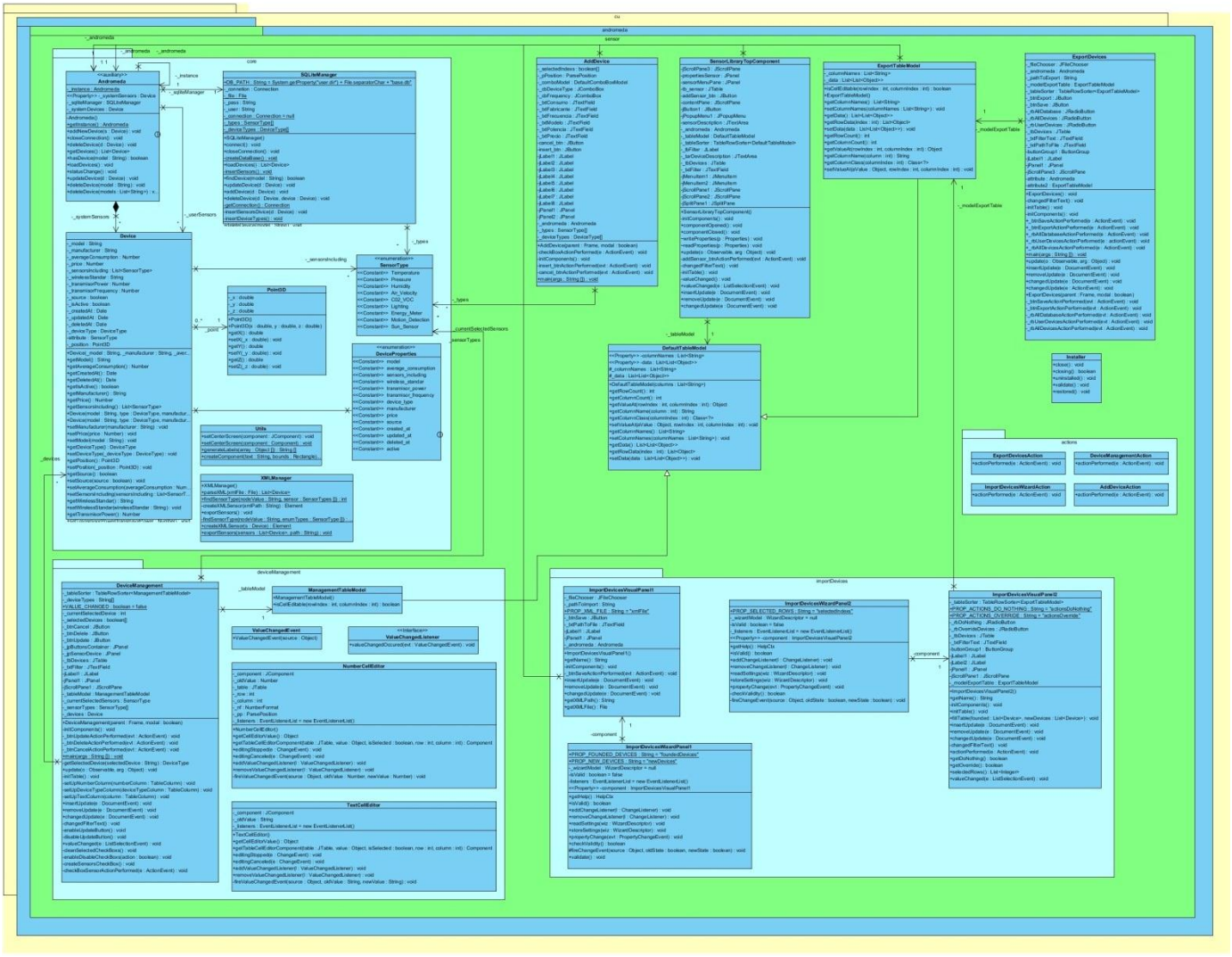

Figura 27. Versión ampliada del diagrama de clases correspondiente al módulo sensor