UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

TRABAJO DE DIPLOMA

PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

Entorno Integrado para el Desarrollo de Soluciones Dinámicas de Software

Autor: Manfred Ramón Díaz Cabrera

Tutor: Ing. Eddy Sánchez Téllez

Ciudad de la Habana, 26 de mayo de 2010

Agradecimientos

Resumen

[Escriba aquí una descripción breve del documento. Una descripción breve es un resumen corto del contenido del documento. Escriba aquí una descripción breve del documento. Una descripción breve es un resumen corto del contenido del documento.]

Tabla de Contenido

Introducción

La construcción y empleo de Sistemas Distribuidos de Software representan dos de los retos más importantes dentro de la Industria del Software. La naturaleza esparcida de los componentes que forman parte de un Sistema Distribuido complejizan el diseño, desarrollo y su posterior despliegue y mantenimiento en operaciones. La dispersión computacional de los elementos de software en nodos - que pueden incluso estar geográficamente dispersos-, la composición mediante elementos de hardware y software heterogéneos, y otros factores implican que controlar un Sistema Distribuido necesite considerables esfuerzos tanto para las empresas que desarrollan los sistemas como para aquellas que luego los adquieren y deben mantenerlos en explotación.

Más allá de las fases por las que se define transita un sistema a lo largo de su ciclo vida, existe un cambio de contexto importante: desde el marco de la empresa que desarrolla el sistema, hacia la estructura organizacional de su cliente. Este ocurre dentro de la fase de Despliegue. En cada contexto se pueden identificar dos macro problemas para los Sistemas Distribuidos:

1. ***En el lado de la organización desarrolladora***: la complejidad del ciclo de desarrollo.
2. ***En el lado de la organización cliente***: la complejidad de la administración y la operación.

Debido a las proporciones que puede alcanzar un sistema con estas características y que la cantidad de personal involucrado en ambos lados –desarrolladores y clientes- durante el ciclo de vida puede ser cuantiosa en comparación con otros tipos de sistemas, solucionar los problemas enunciados requiere mantener una visión homogénea y completa del sistema por parte de cada uno de estos involucrados –o stakeholders, como comúnmente se les conoce- a lo largo de este ciclo, constituyendo esta una tarea difícil producto de la necesidad de coordinar la perspectiva con la que personas u organizaciones de distintas esferas del conocimiento evalúan e interactúan con un sistema de software.

La tarea de lograr esta visión coherente se le atribuye a la Arquitectura de Software y sus principales responsables: los Arquitectos de Software. Para acometer esta función es necesario que la Arquitectura de Software actúe como vehículo de comunicación y a la vez como plano de construcción del sistema.

Estas características ubican a la Arquitectura como uno de los principales factores de éxito en los sistemas y con mayor relevancia en los Sistemas Distribuidos como consecuencia de lo que ya se ha manifestado anteriormente.

Para utilizar la Arquitectura de Software como vehículo de comunicación y plano de construcción es necesario documentarla. La carencia de un modelo preciso de documentación que permita la concepción de la Arquitectura desde una perspectiva unificada que ataque la solución de los problemas propuestos desde la concepción inicial del sistema, ha contrastado con la calidad final de productos de diferentes escalas.

Un caso de estudio que demuestra cómo pueden manifestarse estos problemas lo constituye el Sistema de Gestión de Emergencias de Seguridad Ciudadana 171 (SIGESC 171), una Solución de Software Distribuida creada con el objetivo de dar atención a las emergencias de seguridad ciudadana en la República Bolivariana de Venezuela. Durante el desarrollo, despliegue y mantenimiento de SIGESC 171 se pudieron recoger algunas experiencias relacionadas con la necesidad de que los Sistemas Distribuidos sean concebidos teniendo en cuenta los problemas manifestados tanto en el desarrollo como luego de su despliegue en la organización de destino. Estos problemas se encuentran reflejados en el *Anexo 1 Caso de Estudio de Sistema Distribuido: Sistema de Gestión de Emergencias de Seguridad Ciudadana (SIGESC 171).*

Los problemas detectados para SIGESC 171 se pueden evaluar como problemas que afectan a varias soluciones en el marco de los proyectos productivos dentro de la Universidad de las Ciencias Informáticas. Es posible hacer referencia también a (Hernández León, 2010) donde se exponen algunos de los problemas planteados para SIGESC 171 como problemas que afectan la gestión de proyectos en la Universidad y que pueden ser resueltos desde la perspectiva de la Arquitectura de Software y su documentación.

Por tanto la problemática que se impone resolver es *la necesidad de modelos precisos de documentación de arquitectura que permitan resolver los problemas de complejidad para el desarrollador y para el cliente de Sistemas Distribuidos desde una perspectiva arquitectónica unificada.*

Siendo el objeto de estudio de esta investigación *la construcción de modelos de documentación de la arquitectura* y el campo de acción *la construcción de modelos de documentación de la arquitectura de Sistemas Distribuidos.*

El objetivo general para dar solución a la problemática planteada es:

- Construir modelos de documentación de Arquitectura de Sistemas Distribuidos para minimizar los problemas de los desarrolladores y los clientes de estos sistemas.

Para el cumplimiento del objetivo general se trazaron un conjunto de objetivos específicos:

- Identificar el conocimiento necesario para describir la Arquitectura de un Sistema Distribuido de manera unificada.
- Definir modelos y metamodelos que permitan representar este conocimiento.

Se proponen para este fin un conjunto de tareas de investigación para apoyar la solución de la temática planteada:

1. Realizar un estudio del arte de las diferentes herramientas, metodologías y tendencias actuales relacionadas con los problemas identificados.
2. Seleccionar las herramientas, metodologías y tendencias que se adapten a la situación.
3. Sobre la base de las herramientas, metodologías y tendencias seleccionadas crear una solución para el problema planteado.

Para dar cumplimiento a estos objetivos se emplearon los siguientes métodos de investigación:

- ***Analítico-sintético***: Durante el proceso de revisiones bibliográficas. También en el estudio del estado del arte de las herramientas, metodologías y durante la recopilación de información referente a los proyectos ejecutados en la Universidad de las Ciencias Informáticas.
- ***Inductivo-Deductivo***: Luego del estudio del estado de las arte de las aplicaciones para generalizar los conceptos y funcionalidades que deben estar presentes en los modelos.

- ***Entrevista***: Para la captura de los principales problemas que ocurren durante el desarrollo y despliegue de las soluciones de software desarrolladas por la Universidad de las Ciencias Informáticas.

El resultado de la investigación propuesta en este documento se estructura en cinco capítulos que pueden analizarse en dos partes fundamentales.

Una primera parte que incluye los Capítulos 1 y 2 donde se analiza la Arquitectura de Sistemas Distribuidos y las tendencias y retos fundamentales en el diseño de estos sistemas y en el Capítulo 3 donde se propone la solución a los problemas de las empresas desarrolladoras durante el ciclo de desarrollo.

La segunda parte –Capítulos 4 y 5- analiza el estado del arte de las tendencias hacia los Sistemas Distribuidos Administrables y concreta una propuesta de solución de modelos que tengan en cuenta la infraestructura y el despliegue de estas soluciones como una primera aproximación al problema de la administración de sistemas.

Capítulo 1 Arquitectura de Software

Introducción a la Arquitectura de Software.

En la década de 1620, Suecia y Polonia se encontraban en guerra. El Rey de Suecia, Gustavus Adolphus, se encontraba determinado a finalizar de manera rápida este conflicto y para ello encargó una nueva nave de guerra que nunca se hubiera visto igual. El *Vasa* se requería que fuera el instrumento de guerra más formidable en el mundo entero: 70 metros de largo, capaz de cargar 300 soldados, y con la increíble cantidad de 64 armas pesadas montadas en dos cubiertas de armas. Intentando adicionar una capacidad de fuego abrumadora a esta nave para asestar el golpe decisivo al enemigo, el Rey insistió en extender las capacidades de fuego del *Vasa* más allá de sus límites. Su arquitecto, Henrik Hybertsson, era un consumado constructor de barcos de origen holandés con una reputación intachable. Incluso con este palmarés la tarea de construir una nave como el *Vasa* se encontraba más allá de su experiencia, incluso la de cualquier otro constructor naval de la época.

Tal y como todas aquellas personas que rompen los límites de su propia experiencia, Hybertsson necesitó balancear muchos aspectos: tiempo de terminación –se encontraba en medio de una guerra-, rendimiento, funcionalidad, seguridad, confiabilidad y por supuesto costo. Además tuvo que balancear las necesidades de un grupo de clientes, el Rey como cliente principal y la tripulación del barco, cuya seguridad dependía de la calidad del trabajo. Para emprender semejante tarea Hybertsson empleó toda su experiencia, experiencia que le indicaba que en la construcción de una embarcación como el *Vasa* debía seguir los principios de la construcción de naves de una sola cubierta de fuego y a partir de ahí extrapolar.

Enrolado en lo imposible de esta tarea, Hybertsson murió un año antes de que se terminara de construir la embarcación. No obstante, el proyecto fue completado siguiendo sus especificaciones y en la mañana del domingo 10 de agosto de 1628 el barco fue inaugurado. De catastrófico se puede calificar el evento: el barco navegó hacia el centro de la bahía de Estocolmo, disparó sus cañones en señal de saludo y acto seguido perdió el balance y se hundió. Decenas de hombres de la tripulación fallecieron ahogados.

Pesquisas subsiguientes concluyeron que la nave había sido construida con las proporciones incorrectas. En otras palabras: **la arquitectura de la nave era defectuosa**.

La relevancia de las implicaciones de las decisiones concernientes a los aspectos arquitectónicos en la construcción, es evidenciada incluso en este relato de casi 400 años. En el desarrollo de soluciones de software, el término *Arquitectura de Software* ha ganado en importancia con el transcurso de los años y el desarrollo manifiesto tanto del sector académico como la industria del software. La incorporación del término de arquitectura asociado al desarrollo de sistemas ha tenido varios momentos importantes.

Desde la propuesta en 1968 por parte de Edsger Dijsktra[1] del diseño de una estructuración correcta de los sistemas antes de comenzar su implementación, pasando por las afirmaciones de P.I. Sharp[2] en una conferencia de la OTAN acerca de la necesidad de especificar el diseño y la forma de un sistema para ingenieros y desarrolladores, la Arquitectura de Software ha ido evolucionando y se han formado concepciones con respecto al significado y las implicaciones de estructurar correctamente un sistema de software.

No fue hasta la década de 1990 cuando la Arquitectura de Software se consolidó. Los trabajos de Perry y Wolf[3] en 1992, Garlan y Shaw[4] en 1996 forman parte de este proceso de consolidación. En estos años surgieron distintas corrientes que alimentaron la concepción sobre la Arquitectura de Software.

Definición de Arquitectura de Software.

Varias definiciones al respecto del término han sido enunciadas, sin prevalecer ninguna sobre otra. En el sitio del Instituto de Ingeniería de Software de la Universidad de Carnegie-Mellon se recogen hoy más de 90 definiciones[5]. Sin embargo, la literatura consultada resalta fundamentalmente las siguientes:

[1] Perteneciente a la Universidad Tecnológica de Eindhoven, Holanda. Ganador del Premio Turing, 1972.
[2] (Comentario en discusión sobre teoría y práctica de la ingeniería de software., 1979)
[3] (Foundations for the study of software architecture, 1992)
[4] (Shaw, y otros, 1996)
[5] Disponibles para consultar en: http://www.sei.cmu.edu/architecture/start/community.cfm

"Una Arquitectura de Software de un programa o sistema informático consiste en la estructura de estructuras de ese sistema, la cual comprende elementos, las propiedades externamente visible de esos elementos, y las relaciones entre ellos." (Bass, et al., 2003)

"La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, la relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución." (IEEE, Septiembre, 2000)[6]

Puede apreciarse la arquitectura de un sistema como el conjunto de decisiones principales concernientes al diseño del propio sistema. El término "*principal*" indica el grado de importancia de la decisión al respecto de las necesidades de los stakeholders del sistema y que le confieren el nivel de "*arquitectónica*", implicando que no todas las decisiones relativas al diseño de un sistema son arquitectónicas y no todas tienen impacto sobre su arquitectura. Estas decisiones de diseño recogen todas las facetas del software a desarrollar: estructura, comportamiento, interacción y propiedades no funcionales.

La Arquitectura de Software define elementos de software mediante la recopilación de la información concerniente a sus responsabilidades y formas de interacción, abstrayéndose de aquellos detalles que no afectan sus relaciones con otros elementos, relaciones de uso, de interacción, y de otras naturalezas. De aquí que cada sistema de software tenga su arquitectura -incluso aquellos compuestos por un único elemento de software- como una propiedad intrínseca, más allá de que se encuentre o no documentada o representada. También la arquitectura compendia el comportamiento de cada uno de los elementos que la conforman, comportamiento que implica la descripción de las interacciones.

Es necesario destacar que todas las definiciones revisadas son indiferentes al respecto de si una arquitectura es o no adecuada para un sistema. No obstante, existen métodos para la evaluación de las arquitecturas que constituyen una forma de identificar si una arquitectura en particular es adecuada para un sistema. Además, es ya una tendencia dentro del diseño de arquitecturas de software la utilización de estilos y patrones arquitectónicos que han probado su eficiencia en la resolución de problemas relacionados con el propósito final de la arquitectura: satisfacer las necesidades de sus stakeholders.

[6] Considerada como la definición oficial de arquitectura de software.

La Arquitectura de Software y sus stakeholders.

Los stakeholders de la arquitectura son una pieza fundamental en su concepción. Según (Clements, et al., 2003) "*un stakeholder de una arquitectura es alguien que tiene un interés personal en ella*". Se puede decir que la arquitectura se construye en función de las necesidades de los stakeholders y en ella deben quedar reflejadas las necesidades que han sido satisfechas. La Tabla 1.1 referencia los grupos de stakeholders involucrados en el proceso de construcción de la arquitectura de un sistema y todos aquellos que de alguna manera se benefician de sus propiedades.

Tabla 1.1 Grupos de stakeholders involucrados con la arquitectura de un sistema.

Grupos	Papel que representan
Adquiridores	Pagan por el producto.
Asesores	Verifican la conformidad del producto con las especificaciones.
Comunicadores	Utilizan la arquitectura en entrenamientos y documentos.
Arquitectos	Desarrollan y utilizan la arquitectura como guía y soporte.
Desarrolladores	Creadores del sistema.
Personal de Mantenimiento	Evolucionan y corrigen el sistema.
Proveedores	Proveen partes integrantes del sistema.
Personal de Soporte	Auxilian en la utilización del sistema.
Administradores del Sistema	Mantienen en ejecución el sistema.
Probadores	Verifican la funcionalidad operativa del sistema.
Usuarios finales	Utilizan el sistema como parte de su trabajo.

Estas propiedades contribuyen a la calidad de los servicios prestados e inciden en el diseño y concepción de la arquitectura. Referenciados también en la literatura como requisitos no funcionales o cualidades sistémicas[7], tributan a determinados grupos de stakeholders facilitándoles, en cierto grado, el nivel de complicidad con el sistema. De acuerdo a estos grupos y el rol que desempeñan dentro de la concepción del sistema, los requisitos no funcionales reflejados en la arquitectura se clasifican en: manifiestos, operacionales y evolutivos –ver Tabla 1.2.

[7] (Sun Microsystems, 2001) pp.12.

Tabla 1.2 Requisitos no funcionales y forma de tributo a los stakeholders

Requisitos No Funcionales	Forma de tributo
Manifiestos	Observables por todos los stakeholders que interactúen con el sistema.
Desempeño	Tiempo de respuesta desde el punto de vista del usuario.
Disponibilidad	Por ciento de tiempo que un sistema puede procesar solicitudes.
Confiabilidad	Grado de probabilidad de realizar operaciones correctamente.
Usabilidad	Facilidad de utilización del sistema.
Operacionales	Observables, cuando el sistema se encuentra en operación, por los administradores o personal de soporte del sistema.
Rendimiento	Solicitudes atendidas por unidad de tiempo.
Manejabilidad	Cantidad inversa de esfuerzo para realizar labores administrativas.
Utilidad	Esfuerzo para actualizar el sistema o reparar errores.
Seguridad	Prevención del uso indeseado, inapropiado o abuso del sistema.
Verificabilidad	Esfuerzo invertido para detectar y aislar errores.
Evolutivos	Relacionados con el comportamiento del sistema cuando sufre algún cambio. Dirigidos a los desarrolladores y personal de mantenimiento del sistema.
Escalabilidad	Habilidad para soportar la calidad de servicio requerida conforme la carga aumenta.
Flexibilidad	Esfuerzo ahorrado cuando se hace un cambio de información.
Portabilidad	Esfuerzo ahorrado cuando se migra hacia una infraestructura diferente.
Reusabilidad	Esfuerzo ganado en la utilización de componentes existentes.
Extensibilidad	Esfuerzo ahorrado para adicionar nuevas funcionalidades.
Mantenibilidad	Esfuerzo ahorrado para revisar y corregir errores.

Valor de la Arquitectura de Software.

Desde una perspectiva técnica se puede decir que la Arquitectura es decisiva en el éxito de un proyecto o sistema de software. Existen tres razones fundamentales que exponen la importancia de la Arquitectura de Software:

1. **Comunicación entre stakeholders**: la Arquitectura de Software representa una abstracción común del sistema que todos o casi todos los stakeholders pueden utilizar como la base para el entendimiento, la negociación y el consenso mutuo.
2. **Decisiones tempranas de diseño**: la Arquitectura de Software manifiesta las decisiones tempranas de diseño acerca de un sistema. Constituye la fuente primaria que refleja las decisiones que gobernarán el resto del desarrollo, despliegue y mantenimiento durante el ciclo de vida del software.
    a. Define restricciones en la implementación.
    b. Dicta la estructura organizacional.
    c. Inhibe o permite atributos de calidad del sistema.

   d. Hace más fácil la administración de los cambios en el sistema.
   e. Auxilia en la creación y evolución del prototipo del sistema.
   f. Permite una mayor precisión en la estimación de costos y tiempos.

3. **Abstracción transferible del sistema**: la Arquitectura constituye un modelo intelectual que representa cómo un sistema se encuentra estructurado y cómo sus elementos funcionan de manera cohesionada. Este modelo es transferible más allá de los límites del sistema y puede ser particularmente utilizado en otros sistemas que posean los mismos atributos no funcionales y características funcionales, promoviendo así la reutilización.
   a. Las líneas de productos de software comparten una arquitectura común.
   b. Los sistemas pueden ser construidos utilizando gran cantidad de elementos desarrollados externamente.
   c. Tributa a la restricción de las alternativas de diseño.
   d. Permite el desarrollo basado en plantillas.
   e. Puede constituir una base para el entrenamiento de nuevos miembros del equipo de desarrollo.

La Arquitectura de Software ha emergido como una rama importante de la Ingeniería de Software, particularmente en la construcción de grandes sistemas. Ha permitido a distintos grupos de personas –muchas veces separadas por límites geográficos, organizacionales e incluso temporales- trabajar de manera productiva y cooperada en la solución de problemas de un tamaño mayor a lo que cualquiera de estos podría ser capaz de hacer individualmente.

Estilos y patrones arquitectónicos

Generalmente, cuando los expertos trabajan en la resolución de un problema es usual que una de las primeras ideas que rijan su comportamiento al enfrentarse a este sea la reutilización de las soluciones de problemas que ya hayan resuelto anteriormente, problemas que posean alguna similitud con la situación actual que están resolviendo. Desde el inicio de la Arquitectura de Software se ha observado en la práctica un conjunto de regularidades en la estructura y configuración del diseño e implementación de los

sistemas. Las mismas aparecían una y otra vez en sistemas que cumplían con demandas similares y además no parecían ser tampoco un número elevado de configuraciones diferentes.

Dos conceptos dentro marco teórico-práctico de la Arquitectura de Software recogen este conjunto de regularidades: los estilos y los patrones arquitectónicos.

Estilos arquitectónicos

Según Reinoso en (Reinoso, y otros, 2004)[8] un estilo arquitectónico describe "*un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer un sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo*".

Cuando se habla de arquitecturas multicapas, cliente-servidor, orientadas a servicios u otras se refiere a un grupo de decisiones esenciales sobre los elementos que componen la arquitectura, resaltando las restricciones que cumplen estos elementos y sus relaciones en la estructura de un sistema de software.

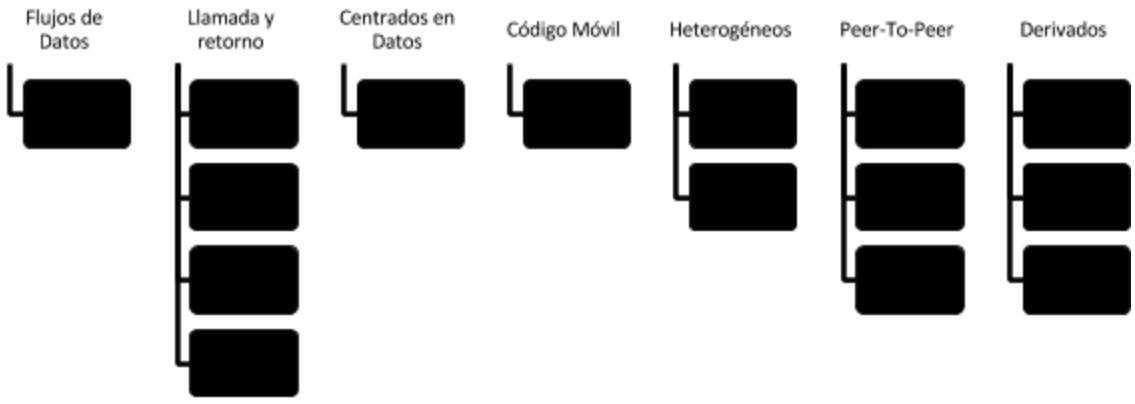
Entre las primeras referencias a estilos arquitectónicos se encuentra (Foundations for the study of software architecture, 1992) donde Perry y Wolf enuncian que: *"La década de 1990, creemos, será la década de la Arquitectura de Software. Usamos el término "arquitectura", en contraste con "diseño", para evocar nociones de codificación, de abstracción, de estándares, de entrenamiento formal (de arquitectos de software) y de estilo".* Aunque sin definir concretamente el término se vislumbra lo que en el futuro sería considerado uno de los pilares fundamentales de la teoría de la Arquitectura de Software.

Las nociones de estilos arquitectónicos encarnan entidades que ocurren al nivel más abstracto del diseño, un nivel puramente arquitectónico. A pesar de que a nivel conceptual es casi unánime la coincidencia de los términos que definen un estilo arquitectónico, no ha sido tal la unanimidad en la catalogación de los mismos en familias. Innumerables autores han dado, sin entrar en detalles acerca de los criterios escogidos, su clasificación taxonómica de los estilos.

[8] (Reinoso, y otros, 2004) pp.6

En la bibliografía de referencia, principalmente en (Reinoso, y otros, 2004) se recogen algunos catálogos considerados primarios y de los que se han derivado varios otros. Para los propósitos de este trabajo se seleccionará la clasificación dada en (Reinoso, 2005), representada en la Figura 1.1. La misma recoge los integrantes fundamentales de las familias y que se encuentran entre los estilos más utilizados en la actualidad: Modelo-Vista-Controlador, Arquitecturas en Capas, Arquitecturas Orientadas a Objetos, a Componentes, a Servicios, entre otras.

Figura 1.1 Catálogo de estilos arquitectónicos



Los beneficios consecuentes del uso de los estilos en la Arquitectura de Software son evidentes. Los estilos arquitectónicos:

1. Promueven la reutilización del diseño, dado que el mismo diseño arquitectónico puede ser reutilizado a través de sistemas con características similares.
2. Conducen a la reutilización de código mediante la replicación de las implementaciones de los aspectos invariantes de los estilos.

3. Permiten un mejor entendimiento de la organización estructural de un sistema mediante la utilización de las convenciones (nombres, propósitos, responsabilidades, restricciones) establecidas para cada estilo.
4. Promueven y soportan la interoperabilidad. Ejemplos: CORBA[9], el modelo OSI[10], o la integración de XML y SOAP para intercambio de mensajes en arquitecturas orientadas a servicios.
5. Al acotar el marco de diseño, permiten el análisis especializado, muchas veces automático para determinar características y propiedades inherentes al estilo.
6. Posibilitan la representación gráfica de los elementos y relaciones que lo componen.

Existen en la actualidad cientos, si no miles, de estilos arquitectónicos en uso. No se pretende hacer de esta sección un tratado sobre los estilos arquitectónicos que existen y sus clasificaciones, pero si es necesario esclarecer que para el resto de la investigación, los estilos arquitectónicos serán vistos desde la óptica de:

1. Elementos Arquitectónicos. (Componentes y Conectores)
2. Relaciones entre los Elementos.
3. Restricciones (Semánticas y Topológicas).

Patrones arquitectónicos

Como se ha referido anteriormente, es poco común el diseño de arquitecturas creando estructuras de componentes que no tengan alguna semejanza con algunas ya creadas. Esta forma de agrupar los problemas y sus soluciones en pares problema-solución es común en varios dominios: arquitectura, economía, ingeniería de software, entre otros.

El concepto de patrón fue introducido a partir del trabajo del arquitecto –en el sentido tradicional de la palabra- Christopher Alexander. En un libro de 1979[11], Alexander define un patrón como "*una regla de tres*

[9] Siglas en Inglés para: Common Object Request Broker Architecture.
[10] Siglas en Inglés para: Open System Interconnection.
[11] Alexander, Christopher. The Timeless Way of Building. Oxford University Press. 1979.

partes, la cual expresa la relación entre cierto contexto, un problema y una solución"[12]. La introducción del concepto de patrón en la Ingeniería de Software proviene de la experiencia acumulada por expertos en la práctica del desarrollo de software. Estas soluciones recurrentes han sido utilizadas para la solución de problemas de manera elegante y efectiva.

Los tres componentes fundamentales de un patrón son el contexto, el problema y la solución. El contexto representa la situación de diseño que da origen al problema de diseño, problema guiado por el conjunto de fuerzas que, de manera repetida, le originan. La solución es la configuración que permite un balance de las fuerzas, ya sea de estructuras de componentes y sus relaciones o comportamientos en tiempo de ejecución.

La cantidad de patrones existentes en la actualidad supera en variedad a los estilos arquitectónicos. Existen un sinnúmero de fuentes que analizan los patrones, en la mayoría de estas fuentes se clasifican los patrones de acuerdo al nivel de abstracción de los problemas de diseño que solucionan. Generalmente las clasificaciones son: patrones arquitectónicos, patrones de diseño y patrones de implementación.

De acuerdo con (Buschmann, y otros, 1996) un patrón arquitectónico *"expresa un esquema de organización estructural fundamental para sistemas de software. Provee un conjunto predefinido de subsistemas, especifica sus responsabilidades, e incluye reglas y normas para organizar las relaciones entre ellos*".[13] Estos patrones representan las tríadas contexto-problema-solución de más alto nivel de abstracción dentro de las soluciones recurrentes a los problemas de diseño. En el propio texto se propone una taxonomía que agrupa a los patrones fundamentales en las familias: del Pantano a la Estructura, Sistemas Distribuidos, Sistemas Interactivos y Sistemas Adaptables, ejemplificadas en la Tabla 1.3.

Tabla 1.3 Familias de patrones arquitectónicos

Grupo	Representantes
Del Pantano a la Estructura	Capas
	Tuberías y filtros
	Pizarra
Sistemas Distribuidos	Broker
Sistemas Interactivos	Modelo-Vista-Controlador

[12] (Buschmann, y otros, 1996) pp.3
[13] (Buschmann, y otros, 1996) pp.12

La taxonomía propuesta por (Buschmann, y otros, 1996) para los patrones arquitectónicos y la nomenclatura de los propios patrones hará recordar en gran medida a los estilos arquitectónicos.

Estilos versus Patrones en el nivel de la Arquitectura de Software.

La comparativa entre patrones y estilos arquitectónicos arroja muchas veces resultados contradictorios. Por una parte son evidentes las similitudes y coincidencias entre ambos, tal y como también son bastante claras sus diferencias.

El primer punto a consensuar en la diferenciación entre estilos y patrones es que ambos se encuentran sustentados y están dirigidos a diferentes grupos de stakeholders dentro del campo de la Arquitectura de Software. Por un lado, los que trabajan con estilos favorecen el tratamiento estructural, teórico, la investigación académica y la arquitectura en el nivel de abstracción más elevada y aquellos que defienden los patrones señalan la reutilización de la experiencia en cuestiones más cercanas a elementos de más bajo grado de abstracción, más cercanas al diseño, la implementación, el código. Cabe señalar que los patrones surgieron primero que los estilos, incluso antes del nacimiento formal de la Arquitectura de Software como campo independiente de la Ingeniería o el Diseño. Los estilos vinieron luego a formalizar las experiencias adquiridas en la práctica.

Cuando ambos términos se encontraban en concepción se pueden hallar referencias a “clases de arquitecturas”, “tipos arquitectónicos”, “arquetipos recurrentes”, entre otros. En sus orígenes entonces podemos decir que ambos se encontraban fusionados. La posterior distinción enfatiza sus manifestaciones dentro de la arquitectura: *“Los estilos solo se manifiestan en arquitectura teórica descriptiva de alto nivel de abstracción; los patrones por todas partes. (…) Los estilos se encuentran en el centro de la arquitectura y son su sustancia. Los patrones de arquitectura están claramente dentro de la*

disciplina arquitectónica, solapándose con los estilos, mientras los patrones de diseño se encuentran más bien en la periferia, si es que no decididamente fuera. "[14]

La base teórica de los estilos arquitectónicos los convierte en una opción fuerte para la automatización del análisis y el diseño de la arquitectura de un sistema de software. Las propiedades establecidas, los roles y responsabilidades de los elementos dentro del marco de definición del estilo, así como las restricciones sobre estos, propician su utilización en herramientas de análisis y diseño arquitectónico que facilitan el trabajo de los arquitectos en el proceso de la toma de decisiones concernientes a las propiedades de la Arquitectura de Software.

Tendencias actuales en la Arquitectura de Software.

Existen un gran número de temas considerados vitales dentro del campo de la Arquitectura de Software. Muchos de estos se encuentran en franco progreso y son la base para varias de las investigaciones que traerán como resultado el cambio definitivo que impulse la industrialización dentro de la producción de sistemas de software. Hasta el día de hoy, la industria del software es considerada una industria artesanal, no se ha logrado aún la consecución de la producción industrial mediante el ensamblaje de componentes, la automatización de los procesos productivos, la optimización de los métodos y herramientas, la estandarización, entre otros factores que son la base para industrias eficientes.

En la Tabla 1.4 se muestran los principales exponentes entre las investigaciones abiertas de la Arquitectura de Software. Con el objetivo de esclarecer algunos puntos dentro del marco de la presente investigación, es necesario hacer una descripción más detallada de dos innovaciones críticas: los Lenguajes Específicos de Dominio y la Arquitectura y el Desarrollo Basados en Modelos.

Tabla 1.4 Investigaciones abiertas en el campo de la Arquitectura de Software

Investigación	Campo de Acción
Modelos y Lenguajes de descripción de Arquitectura	Relativos a la definición de modelos arquitectónicos del software, y al diseño de notaciones específicas para la descripción de la arquitectura de los sistemas de software.
Estilos y Patrones Arquitectónicos	Catalogación y clasificación de estilos arquitectónicos, la búsqueda de patrones estructurales, la determinación de propiedades de los estilos y patrones, la definición de

[14] (Reinoso, y otros, 2004) pp.4

	reglas que ayuden al diseñador (arquitecto) a decidirse por un estilo u otro en función de sus necesidades.
Arquitectura y Lenguajes Específico de Dominio	Adaptación de los estilos arquitectónicos y los lenguajes a las características de un dominio, incluyendo todo lo relacionado a los marcos de trabajo composicionales, el establecimiento de familias y líneas de productos, así como las fábricas de software.
Métodos de Desarrollo	Definición de métodos de desarrollo integrables con los aspectos arquitectónicos.
Herramientas y Entornos de Desarrollo	Editores gráficos y textuales, traductores, generadores de código, repositorios de arquitecturas y componentes, herramientas para el uso de marcos de trabajo, herramientas de análisis, etc.
Ingeniería Inversa y Reingeniería	Desarrollo de técnicas y herramientas de ingeniería inversa para esclarecer las arquitecturas de aplicaciones que no se encuentren previamente documentadas.

Lenguajes Específicos de Dominio

Dentro de la terminología del desarrollo de software, la palabra *lenguaje* es asociada por generalización con los lenguajes textuales de propósito general tales como C#, Visual Basic, Java, entre otros. En la referencia a la Arquitectura y el Desarrollo de aplicaciones, el término *lenguaje* se amplía considerablemente: incluye lenguajes gráficos, diagramas de flujo, diagramas entidad-relación, diagramas de estados, diagramas de Venn, entre muchos otros.

Con el surgimiento del Desarrollo Específico de Dominio se ha incluido una nueva familia de lenguajes: los Lenguajes Específicos de Dominio. Según (Cook, y otros, 2007) un Lenguaje Específico de Dominio (DSL [15]) es: *"un lenguaje personalizado dirigido a un pequeño dominio de problemas, el cual este describe y valida en términos nativos al dominio."*[16] Ejemplos de DSL hay varios, muchos de ellos lenguajes que cubren dominios conocidos como SQL –para Bases de Datos-, HTML –diseño de páginas web-, XML –descripción de datos-, XAML – descripción de interfaces de usuarios-, la Forma Normal de Bakus (BNF) –descripción de lenguajes-, entre otros.

Las clasificaciones dadas a los DSL son varias y dependen del propósito o la intención con que son creados. Por ejemplo Fowler en (Fowler, 2009), propone la clasificación de los DSL en:

[15] Siglas en Inglés para: Domain Specific Language.
[16] (Cook, y otros, 2007) pp.10

1. *Externos*: utilizan un lenguaje diferente al lenguaje en el que se desarrolla la aplicación en la que son utilizados.
2. *Internos*: utilizan el mismo lenguaje de propósito general en el que la aplicación es construida, pero utilizado con un estilo limitado y particular.
3. *Workbenches de Lenguajes*: son Entornos Integrado de Desarrollo diseñados para la construcción de Lenguajes Específicos de Dominio

La utilización de los DSL mejora la productividad del desarrollo, proveen un medio de comunicar de manera clara la intención o las intenciones de una parte del sistema. Brindan una forma de leer y manipular las abstracciones propias del dominio a los que están destinados. Además posibilitan la comunicación con expertos del dominio y ayudan a la comunicación con los clientes y los usuarios dado que proveen un lenguaje para el entendimiento común en términos del dominio que la aplicación automatiza. Desde el punto de vista de la configuración, los DSL permiten cambios en el contexto de ejecución, permiten la descripción de configuraciones y códigos que necesitan ejecutarse de acuerdo a situaciones alternantes en el ambiente de operativo de un sistema.

Arquitectura y Desarrollo Dirigidos por Modelos

La rápida evolución de la tecnología en un ambiente interconectado y distribuido como el imperante en el mundo actual, que cuenta además con diferentes plataformas, hace que integrar varios sistemas dentro de un ambiente organizacional se vuelva complejo. Para resolver el problema de la integración de sistemas y componentes desarrollados con diferentes tecnologías (EJB/J2EE, .NET, XML, SOAP, etc.) y que estos se adapten con facilidad a los cambios producidos en los negocios, la tendencia actual dentro de la industria del software es el llamado Desarrollo Dirigido por Modelos (MDD[17]). En MDD los artefactos principales en el proceso de desarrollo son modelos y las transformaciones a los que estos son susceptibles. La idea detrás de MDD es la generación automática o semiautomática de la implementación a partir de modelos obtenidos que describen un sistema de software en sus partes componentes.

Con la lógica de Desarrollo Dirigido por Modelos expertos del negocio tienen la posibilidad de expresar el conocimiento en un lenguaje de modelado, haciendo posible la abstracción de características de la

[17] Siglas en Inglés de: Model Driven Development.

implementación –como la dependencia de la plataforma- y sirviendo como entrada para la implementación por parte de expertos en las tecnologías de este conocimiento y su transformación a una solución de software.

Existen varias propuestas para concretar MDD dentro de la industria. Entre estas propuestas la más destacable es la iniciativa de Arquitectura Dirigida por Modelos (MDA[18]), diseñada por el Object Management Group (OMG).

Compuesto por empresas líderes dentro de las Tecnologías de la Información como Compaq, Fujitsu, Ericsson, Microsoft, entre otras, desde 1989 el OMG ha venido trabajando en el desarrollo de arquitecturas y herramientas para sistemas, fundamentalmente sistemas distribuidos. Fruto de este trabajo son los estándares CORBA, UML, XMI. Este proceso de estandarización y evolución de las propuestas han resultado en el surgimiento de la Arquitectura Dirigida por Modelos.

La Arquitectura Dirigida por Modelos es la evolución de los estándares con el propósito de mejorar el proceso de Desarrollo de Software Dirigido por Modelos. Las ideas centrales de MDA persiguen los objetivos de separar la especificación de la funcionalidad del sistema de su implementación sobre una plataforma en una tecnología específica y de controlar la evolución desde modelos abstractos a implementaciones tendiendo a aumentar el grado de automatización.

Para alcanzar estas metas OMG propone un proceso de desarrollo que sigue cuando menos las siguientes etapas:

1. Construir un Modelo Independiente de Plataforma (PIM) en un alto nivel de abstracción, e independiente de una tecnología específica.
2. Transformar el modelo PIM en uno o más modelos dependientes de una plataforma específica, denominados Modelos Específicos de Plataforma.
3. Transformar los Modelos Específicos de Plataforma a código ejecutable.

[18] Siglas en Inglés de: Model Driven Architecture.

Con la aplicación de MDA se propicia primeramente la productividad, disminuyendo el tiempo de desarrollo y el esfuerzo en programación. Tributa además a la interoperabilidad y la portabilidad de las aplicaciones, si bien es necesaria la codificación adicional en modelos, promueve la portabilidad dentro de una misma plataforma y entre plataformas distintas. Además se promueve la reusabilidad y la facilidad de mantenimiento.

Capítulo 2 Arquitectura de Sistemas de Software Distribuidos.

Introducción a los Sistemas Distribuidos

Divide y, ¿vencerás? Depende. La teoría de dividir para conquistar empleada siglos atrás por el Imperio Romano ha encontrado su eco en muchos de los aspectos de la vida en la época contemporánea. La Informática como ciencia no ha escapado a los toques pintorescos con los que se ha adoptado esta frase con el transcurso de los años, la manida estrategia se ha reflejado en ideas para desarrollar algoritmos de cómputo –léase, la técnica de solución de problemas "Divide y vencerás"- y a una mayor escala, la distribución de las tareas entre múltiples ordenadores en una red.

En el principio todos los sistemas eran centralizados, en los albores de los centros de procesamientos de datos los ordenadores se encontraban aislados y eran utilizados por personal científico con una elevada preparación profesional. El nacimiento de los miniordenadores en la década de 1970 y el posterior surgimiento de los ordenadores personales una década después, marcó la etapa del surgimiento de los sistemas distribuidos. La distribución de los datos y el procesamiento en distintos ordenadores devino en el incremento de las capacidades de almacenamiento y cómputo de la información.

El desarrollo de las redes de dispositivos durante el transcurso del último decenio ha traído aparejada la evolución de la arquitectura de los sistemas de software, desde la clásica arquitectura cliente/servidor, a las arquitecturas multicapas y, de manera más reciente, a los Servicios Web o la nueva tendencia del Software como Servicio. En estos paradigmas arquitectónicos los componentes de los sistemas pueden estar localizados en diferentes ordenadores y comunicarse a través de redes de computadoras y actuar coordinadamente con el objetivo de satisfacer los requisitos para los que fueron creados. Los sistemas con estas características son conocidos como Sistema Distribuidos.

Un Sistema Distribuido es -según (Coulouris, et al., 2005)- *"aquel en el que los componentes localizados en ordenadores, conectados en red, comunican y coordinan sus acciones únicamente mediante el paso de*

mensajes."[19]Algunas otras definiciones precisan detalles que pueden resultar relevantes cuando se estudian los Sistemas Distribuidos. En (Marquès i Puig, y otros, 2007) se define un sistema distribuido como "*una colección de ordenadores* ***autónomos*** *enlazados por una* ***red*** *de ordenadores y soportados por un* ***software*** *que hace que la colección actúe como un servicio* ***integrado***"[20].

La distribución del procesamiento y almacenamiento de los datos en diferentes nodos ha devenido en una tendencia con gran aceptación. Las Sistemas Distribuidos poseen un conjunto de características inherentes a su arquitectura que los convierten en adaptables a un gran número de dominios de problemas. Primero, la descomposición funcional de los servicios basada en las capacidades, propósitos y características de los diferentes componentes. Además, la distribución inherente de las entidades componentes propicia que la información manipulada por el sistema sea creada y mantenida por diferentes usuarios, además de que pueda ser almacenada, analizada y utilizada por otros sistemas o aplicaciones.

Como resultado de esta distribución es posible afirmar que los Sistemas Distribuidos aumentan la confiabilidad mediante la preservación y el respaldo de la información por largos períodos de tiempo debido a la replicación de los datos en varias localizaciones. También adicionan la posibilidad de escalar los recursos para incrementar el rendimiento y la disponibilidad. La compartimentación entre varias entidades y organizaciones de los datos e infraestructura de soporte de los componentes ayuda también a mitigar el costo total de propiedad.

Arquitecturas de los Sistemas Distribuidos

Es evidente que para obtener los beneficios de la distribución de los componentes y la información, es necesaria una correcta distribución arquitectónica de los mismos. Desde el surgimiento de los Sistemas Distribuidos y su posterior establecimiento como una tendencia válida dentro de la industria del software,

[19] (Coulouris, et al., 2005)
[20] (Marquès i Puig, y otros, 2007) pp.9

la arquitectura de estos ha ido evolucionando a la par de las tecnologías y las necesidades establecidas en cada momento.

Entre los estilos arquitectónicos utilizados en el diseño y construcción de los Sistemas Distribuidos modernos se han destacado fundamentalmente las Arquitecturas multi-estratos[21], las Arquitecturas Orientadas a Componentes, las Arquitecturas de iguales o descentralizadas y más recientemente las Arquitectura Orientada a Servicios (SOA) y las Arquitecturas de publicación/suscripción y las Arquitecturas de código móvil.

Arquitecturas multiestratos.

Quizás son las arquitecturas candidatas para un Sistema Distribuido que han sido empleadas con mayor frecuencia. Dentro de las arquitecturas multiestratos se encuentra el estilo cliente/servidor y las arquitecturas n-estratos[22]. Muchas fuentes consideran la arquitectura n-estratos como una generalización del estilo cliente/servidor. Este último es referido también con frecuencia como una arquitectura 2-estratos haciendo énfasis en su relación de especialización con las arquitecturas multiestratos.

Cliente/Servidor

Históricamente la Arquitectura Cliente/Servidor ha indicado que una aplicación de interfaz gráfica de usuario en el escritorio se comunica con un servidor de base de datos que contiene la mayoría de las lógicas de negocio en forma de procedimientos almacenados o en un servidor de aplicaciones dedicado. De manera general el estilo arquitectónico cliente/servidor describe la relación de un cliente con uno o más servidores, donde el cliente inicia una o varias peticiones y espera por las repuestas que luego procesa -Figura 2.1.

Figura 2.1 Arquitectura cliente/servidor

[21] multi-tier en inglés.
[22] N-tier en inglés.

A partir de este modelo básico se han desprendido diferentes variantes. Tal y como se emplea hoy, en la arquitectura cliente/servidor se identifican además relaciones de colaboración entre los servidores para dar tratamiento a una petición del cliente -en este caso existen servidores que se comportan como clientes de otros servidores- y otras relaciones entre servidores como los casos de servidores que proveen los mismos servicios de manera concurrente.

Como se ha mencionado, este estilo arquitectónico se encuentra en un gran número de aplicaciones, incluyendo las aplicaciones basadas en navegadores Web que se ejecutan en Internet o en una intranet, las aplicaciones que acceden almacenes de datos remotos como son los casos de los clientes FTP, los clientes de correo electrónico, las herramientas de consultas de base de datos, entre otras. También herramientas y utilidades que manipulan sistemas remotos como las herramientas de administración de sistemas y de monitorización de redes utilizan la arquitectura cliente/servidor.

La selección de este estilo para la construcción de un Sistema Distribuido provee –entre otros- los siguientes beneficios:

- *Mayor seguridad*: debido a que todos los datos se encuentran almacenados en un servidor, es más fácil controlar la seguridad y las políticas de seguridad que se aplican sobre la información de lo que sería posible controlar en los entornos de ejecución de los clientes.
- *Acceso centralizado a los datos*: la centralización de los datos permite que las operaciones de acceso y actualización sobre estos sean mucho más fáciles de realizar y administrar que en otros estilos.

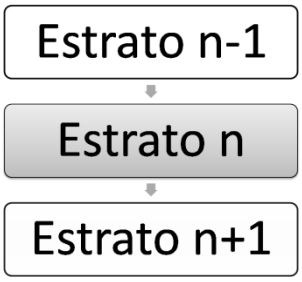
- *Facilidad de mantenimiento*: los roles y las responsabilidades dentro del sistema se encuentran distribuidos entre varios servidores conocidos. Esto asegura que los clientes no adviertan los efectos de la reparación, actualización o relocalización de los servidores.

El lunar negro del estilo cliente/servidor es que en muchos casos existe una tendencia manifiesta a mezclar código de la lógica de datos con la lógica de negocios dentro de un solo servidor. Esta mezcla puede impactar de manera decisiva en la escalabilidad, la extensibilidad y la confiabilidad del sistema.

Arquitectura multiestratos

Como una evolución para la solución a los problemas manifiestos de la arquitectura cliente/servidor surge la arquitectura multiestratos o n-estratos, como también se le conoce. La característica fundamental del estilo multiestratos es la agrupación en niveles o estratos de las aplicaciones y los componentes de servicios de un Sistema Distribuido mediante la descomposición funcional. Cada estrato es completamente independiente del resto excepto para aquel superior o inferior de acuerdo al grado de abstracción con que fueron agrupadas las funcionalidades. El "*n-ésimo*" estrato solo tiene que conocer como manipular las peticiones del "*n-1-ésimo*" y como redirigir estas peticiones al estrato inmediatamente inferior "*n+1-ésimo*" en caso de que este exista -Figura 2.2.

Figura 2.2 Arquitectura multiestratos



No existe una restricción específica sobre la cantidad de estratos que se pueden definir en la descomposición funcional de las aplicaciones y componentes de los Sistemas Distribuidos. El número de

estratos más utilizados es la Arquitectura de tres estratos, sin tener en cuenta que con sólo dos estratos la arquitectura multiestratos se vería reducida a una arquitectura cliente/servidor.

En la Arquitectura de tres estratos generalmente se agrupan las aplicaciones y componentes en un estrato de presentación o cliente -donde se encuentran los que responden a las funcionalidades de presentación de los datos e interacción con el cliente-, un estrato de aplicación -donde residen los componentes que encapsulan la lógica de negocio y otras funciones de soporte- y otro estrato de datos donde se agrupan aquellos elementos del Sistema Distribuido encargados de almacenar y manipular la información.

La utilización del estilo multiestratos elimina los problemas que se habían señalado a la arquitectura cliente/servidor. Debido a la baja dependencia entre los estratos, las operaciones de mantenimiento, actualización u otros cambios que pueden incidir en la escalabilidad, la flexibilidad y la disponibilidad del sistema se pueden llevar a cabo sin necesidad de afectar el funcionamiento general.

Arquitecturas orientadas a componentes

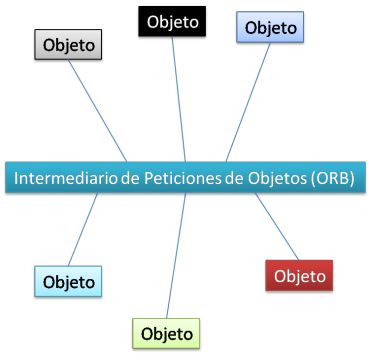
Las Arquitecturas Orientadas a Componentes se concentran en la descomposición del diseño en componentes funcionales o lógicos individuales que exponen interfaces bien definidas que permiten la comunicación entre ellos. Los componentes deben ser reutilizables, reemplazables, diseñados para ejecutarse en diferentes ambientes, extensibles. Deben encapsular todos los detalles del procesamiento interno de los datos excepto aquellos que deban ser públicos a través de sus interfaces y deben cumplir además con la premisa del bajo acoplamiento, dependiendo de la mínima cantidad de componentes posibles.

En los Sistemas Distribuidos, la Arquitectura Orientada a Componentes es la base de las conocidas Arquitecturas de Objetos Distribuidos como la forma más generalizada de implementación de componentes en ambientes distribuidos. En las Arquitecturas de Objetos Distribuidos no existe una distinción de roles o jerarquías como existe en la arquitectura cliente/servidor. Cada entidad distribuida es un objeto que provee servicios a otros objetos y recibe servicios de otros objetos. Esta descentralización

es posible a través de una capa intermedia de comunicación conocida como Intermediario de Peticiones de Objetos (ORB[23]) o simplemente Intermediario (Broker) -Figura 2.3.

La interacción entre los componentes en Arquitecturas de Objetos Distribuidos debe de abstraerse de los mecanismos de infraestructura de comunicaciones en el momento de localización e invocación de los servicios provistos por los componentes. La llamada remota debe lograr ser transparente a la localización del objeto encuestado para el cliente, evitando entremezclar el código correspondiente a la comunicación con el código de negocio del componente. Con esta mezcla podría resultar que un simple cambio en la infraestructura de comunicaciones requiera realizar cambios significativos en los componentes.

Figura 2.3 Arquitectura de Objetos Distribuidos



Numerosas plataformas proveen la posibilidad de implementar Sistemas Distribuidos con Arquitecturas de Objetos Distribuidos: Microsoft ha implementado COM[24] y su variante distribuida DCOM[25] dentro de su plataforma Microsoft Windows, además de garantizar una tecnología con enfoque similar para usuarios de .NET Framework conocida como .NET Remoting, OMG ha creado CORBA[26] como una infraestructura estándar para la implementación de Sistemas Distribuidos basados en ORB y dentro de la plataforma Java

[23] Siglas en inglés de: Object Request Broker.
[24] Siglas en inglés de: Common Object Model.
[25] Siglas en inglés de: Distributed Common Object Model.
[26] Siglas en inglés de: Common Object Request Broker Architecture.

de Sun Microsystems es posible encontrar EJB[27] como otra propuesta de implementación de objetos distribuidos.

La utilización de las Arquitecturas Orientadas a Componentes posibilita:

- *Facilidad en el despliegue:* cuando se creen nuevas versiones de los componentes estas pueden ser puestas en funcionamiento sin afectar otros componentes o el sistema en general.
- *Costo reducido:* El uso de componentes creados por terceros ayuda a la reducción de los costos de desarrollo y la implementación.
- *Facilidad de desarrollo:* La implementación de interfaces conocidas en los componentes para proveer funcionalidades definidas permite desarrollar sin causar impactos significativos en el sistema.
- *Reutilización:* Los componentes desarrollados pueden ser reutilizados en distintas partes del sistema.

Arquitecturas de igual a igual[28]

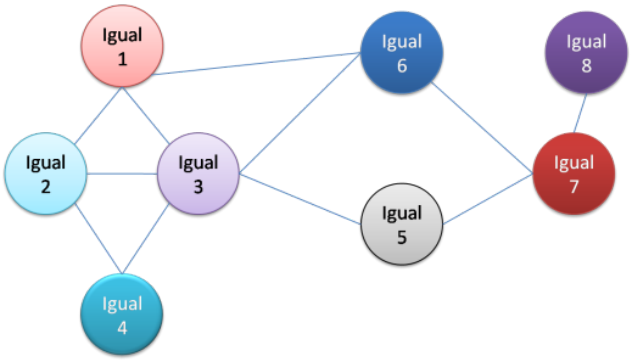
A diferencia de las arquitecturas multiestratos, las arquitecturas de igual a igual se concentran en la distribución horizontal de las funcionalidades de un cliente o un servidor en partes lógicamente equivalentes, de manera que cada parte tenga todas las funcionalidades permitiendo el balance de la carga sobre el sistema. En una Arquitectura de igual a igual la interacción entre componentes ocurre de manera simétrica, al actuar estos con un carácter dual –clientes y servidores a la vez-. Esta característica permite tomar ventaja del poder computacional y de almacenamiento de un gran número de ordenadores conectados a la red del Sistema Distribuido.

De acuerdo con la estructura y localización de los componentes, las Arquitecturas de igual a igual se catalogan en no estructuradas (descentralizadas), estructuradas (semi-centralizadas) e híbridas -Figura 2.4 y Figura 2.5. En el caso de la arquitectura no estructurada los componentes se conectan al sistema sin conocer la topología del sistema ni la distribución física del resto de los componentes. En los sistemas de

[27] Siglas en inglés de: Enterprise JavaBeans.
[28] En inglés: Peer-to-Peer (P2P) Architectures.

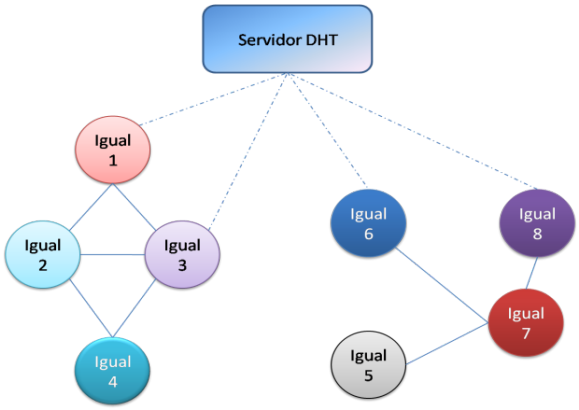
iguales estructurados la organización topológica de los componentes es fuertemente controlada y la información no va a cualquier lugar sino a una localización específica.

Figura 2.4 Arquitectura de igual a igual no estructurada



Para controlar estas localizaciones los sistemas de iguales estructurados utilizan tablas hash distribuidas (DHT[29]) permitiendo que las operaciones de ubicación e incorporación de componentes al Sistema Distribuido se haga de manera determinista.

Figura 2.5 Arquitectura de igual a igual estructurada



Los sistemas de iguales considerados híbridos utilizan una combinación de técnicas no estructuradas con estructuradas. En las arquitecturas de igual a igual híbridas se plantea la existencia de un componente de

[29] Siglas en inglés: Distributed Hash Table.

indización o gestión de localización de recursos, dedicado a mantener información sobre la ubicación de los datos en los iguales. El caso híbrido es tal vez la Arquitectura de igual a igual más familiar debido a la implementación de esta en protocolos de compartimentación de información como BitTorrent, eDonkey, o el caso de la aplicación de comunicación Skype, entre otros, reconocidos por los servicios provistos dentro de Internet.

Arquitecturas orientadas a servicios

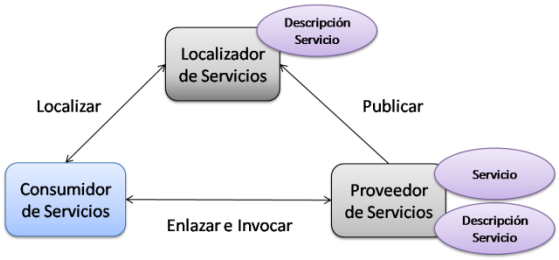
Las Arquitecturas Orientadas a Servicios (SOA[30]) permiten que las funcionalidades de una aplicación sean provistas como un grupo de servicios que serán consumidos por otras aplicaciones. Estos servicios se encuentran débilmente acoplados debido a la utilización de interfaces -basadas en estándares- que pueden ser invocadas, publicadas y descubiertas por sus clientes. De acuerdo con CDBI la Arquitectura Orientada a Servicios es un "*estilo resultante de políticas, prácticas y frameworks que permiten que la funcionalidad de una aplicación se pueda proveer y consumir como un conjunto de servicios, con una granularidad relevante para el consumidor. Los servicios pueden invocarse, publicarse y descubrirse y están abstraídos de su implementación utilizando una sola forma estándar de interfaces.*"[31]

Entre los componentes fundamentales de SOA -reflejados en la Figura 2.6- se encuentran:

- *Servicio*: entidad lógica sin estado, auto-contenida, que acepta una o varias llamadas y devuelve respuestas mediante una interfaz bien definida. Es además autónoma, distribuida y débilmente acoplada.
- *Descripción de Servicio*: contrato definido por una o más interfaces públicas.
- *Proveedor de servicio*: entidad de software que implementa una especificación de servicio.
- *Consumidor de servicio*: entidad de software que llama a un proveedor de servicios. Esta entidad puede ser un cliente u otro servicio.
- *Localizador de servicio*: tipo específico de proveedor de servicios que actúa como registro y permite buscar interfaces de proveedores y sus ubicaciones.

[30] Siglas en inglés de: Service Oriented Architecture.
[31] (Reinoso, 2005) min. 28:17s.

Figura 2.6 Arquitectura Orientada a Servicios



Las Arquitecturas Orientadas a Servicios han probado ser beneficiosas cuando se necesita:

- *Alineamiento con el dominio*: la reutilización de servicios comunes con interfaces estándares incrementa las oportunidades de negocio y tecnológicas además de facilitar la reducción de costos.
- *Abstracción*: la autonomía de los servicios proveen un débil acoplamiento y elevado nivel de abstracción.
- *Descubrimiento*: los servicios pueden exponer sus descripciones para que otras aplicaciones y servicios los localicen y determinen de manera automática su interfaz.
- *Interoperabilidad*: debido a que los protocolos, formatos de datos e interfaces están basados en estándares de la industria del software, los proveedores y consumidores de servicios pueden estar construidos y desplegados en diferentes plataformas.
- *Racionalización:* los servicios pueden ser granulares para proveer funcionalidades específicas, en lugar de duplicar las funcionalidades en distintas aplicaciones ser reutilizan los servicios evitando la duplicación.

Arquitecturas de publicación/suscripción

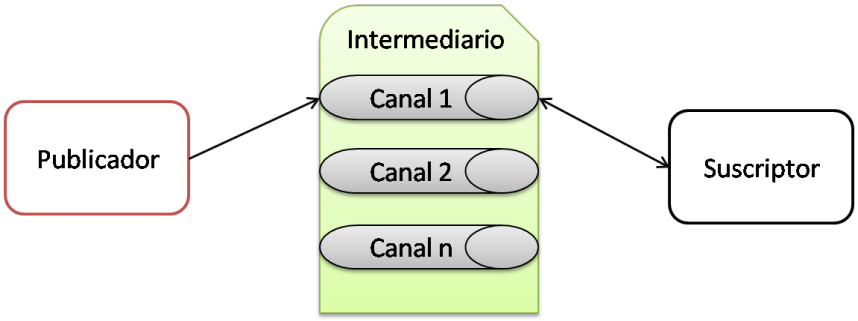
Las Arquitecturas de publicación/suscripción han venido a resolver problemas relacionados con la minimización de interacciones entre los componentes de Sistemas Distribuidos. La idea subyacente en estas arquitecturas predica que para casos donde los elementos de un sistema necesiten mantenerse

actualizados sobre el estado o la actividad de otro componente, no deberían mantenerse encuestando de manera periódica a dicho componente.

La Arquitectura de publicación/suscripción establece que un productor de información debe anunciar la disponibilidad de un tipo de información, el consumidor interesado se suscribe y es responsabilidad del productor la publicación periódica de la información. Es posible acoplar la lógica de publicación/suscripción a otros tipos de arquitecturas con el objetivo de aumentar el rendimiento y disminuir la sobrecarga de la infraestructura de red que soporta al sistema.

Como queda reflejado en la Figura 2.7 la publicación/suscripción está compuesta por cuatro elementos básicos: el publicador -como aplicación o componente que posee la información-, el suscriptor –aplicación o componente que le interesa recibir la información-, un intermediario (broker) –como encargado de recibir la información de los publicadores y las peticiones de los suscriptores, este elemento puede ser alternativo en dependencia de la implementación de la arquitectura- y el canal –como el conector lógico entre los publicadores y suscriptores de información, que determina la naturaleza y propiedades de esta interacción.

Figura 2.7 Arquitectura de publicación/suscripción



Las Arquitecturas de publicación/suscripción están pensadas para proporcionar servicios de coordinación de procesos, replicación de contenidos y alertas. Entre las aplicaciones más importantes implementadas con estas arquitecturas se encuentran los grupos de noticias, las listas de distribución de correo, los sistemas de alertas de bolsa y noticias, los servicios de alertas, monitorización y control, entre otras.

Arquitecturas de código móvil

Las Arquitecturas de código móvil están enfocadas en el aprovechamiento de la movilidad de los componentes en un Sistema Distribuido para cambiar dinámicamente la distancia entre el procesamiento de los datos, la fuente de los datos y el destino de los resultados del procesamiento. Con este cambio de ubicación se pretende la mejora de la eficiencia, la reducción del número de interacciones y el costo de estas.

Para lograr estas metas existen tres enfoques o paradigmas dentro de código móvil: la evaluación remota, el código bajo demanda y los agentes móviles.

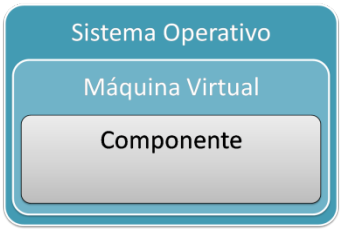
La *evaluación remota* soluciona la deficiencia de recursos en los clientes cuando el cliente tiene los conocimientos necesarios para realizar el procesamiento pero no cuenta con la potencia de cálculo o de almacenamiento de los datos. En estos casos el conocimiento es descrito y enviado a un servidor remoto que utilizando los datos y el conocimiento acerca del procesamiento evalúa y los resultados de esta evaluación los devuelve al cliente.

En un caso diferente aplica el *código bajo demanda*. Aquí el cliente posee los recursos necesarios para realizar el procesamiento de los datos pero no tiene el conocimiento que se necesita para llevar acabo dichas operaciones. Para que la ejecución se efectúe, el cliente envía una petición a un servidor remoto y este responde con el código necesario. Con este código y los recursos a su disposición el procesamiento se ejecuta localmente. En el código bajo demanda es necesaria una relación de confianza entre el cliente y el servidor debido a que el primero está permitiendo la ejecución de un código desconocido en su ambiente de ejecución.

Tanto en el *código bajo demanda* como en la *evaluación remota* es el código el que se mueve del cliente al servidor o viceversa. Existe una tercera opción dentro de las Arquitecturas de código móvil que es conocida como *agente móvil,* como una mezcla entre las dos opciones anteriores. Un *agente móvil* es una unidad computacional capaz de desplazarse entre ordenadores llevándose consigo su estado, el segmento de código y los datos necesarios para realizar operaciones de procesamiento. Esta Arquitectura

es ventajosa utilizarla cuando resulte más eficiente mover el código de procesamiento que mover los datos a procesar a la localización de procesamiento.

Figura 2.8 Arquitectura de máquina virtual



La versión más común de las Arquitecturas de código móvil son las máquinas virtuales. Las máquinas virtuales constituyen plataformas o entornos que controlan la ejecución de componentes implementados en la tecnología base de la máquina virtual -Figura 2.8, en estas los componentes se ejecutan en un ambiente seguro y confiable. Los casos de la ejecución de los applets o mini-aplicaciones escritas en Java, Flash u otros lenguajes en el entorno de ejecución de un navegador, constituyen ejemplos de código móvil, tanto de máquinas virtuales como de demanda de código.

Retos en el diseño de Sistemas Distribuidos

En el diseño y la construcción de Sistemas Distribuidos es necesario considerar un grupo de características o cualidades que forman la base para obtener los beneficios reales en la distribución de aplicaciones, sistemas y datos. Lograr que los Sistemas Distribuidos presenten estas características puede considerarse como una métrica de la calidad del resultado final de los procesos de diseño y construcción de estos sistemas.

Heterogeneidad: El sistema puede estar compuesto por elementos varios en los distintos niveles de su estructura. Debe ser posible diseñar, construir y lograr la interacción de los componentes de un Sistema Distribuido sobre la base de diferentes tipos de redes, sistemas operativos, lenguajes de programación, plataformas de desarrollo y hardware del ordenador o dispositivo sobre el que se ejecutan.

Apertura o Extensibilidad: La apertura o extensibilidad es la capacidad que debe tener el sistema de ser ampliable. La ampliación de un sistema proviene de la adición de recursos y servicios. Se considera un

Sistema Distribuido abierto aquel donde las interfaces de sus componentes se encuentran claramente separadas y disponibles públicamente para permitir la extensión mediante la adición de estos recursos y servicios.

Seguridad: La información que está contenida dentro de los límites de un sistema distribuido debe ser protegida debido a la importancia que esta tiene para los clientes. Es deseable que todo Sistema Distribuido sea capaz de cumplir con los tres fundamentos de la seguridad: la confidencialidad, la integridad y la disponibilidad. Es necesario recordar que debido a la distribución de componentes e información propia de los Sistemas Distribuidos estos deberán atravesar límites empresariales y viajar a través de redes en muchos casos no seguras y deben llegar a su destino sin sufrir alteración o pérdida.

Escalabilidad: La escalabilidad es una propiedad de los sistemas de mantener su rendimiento cuando se incrementa la cantidad de usuarios y de aumentar su desempeño cuando se incrementan la cantidad de recursos. Existe un grupo de parámetros que se necesitan controlar para garantizar la escalabilidad: el coste de la adición de nuevos recursos, la pérdida de rendimiento, el agotamiento de los recursos y los cuellos de botellas en tiempo de ejecución.

Tolerancia a fallos: El fallo de uno o más componentes del sistema no debe provocar el colapso del sistema completo. Es recomendable además que sea posible detectar el fallo, aislarlo, que el sistema continúe funcionando siendo tolerante –se conoce del fallo y se espera por la corrección o se decide utilizar otro servicio- y que sea posible recuperar el estado del sistema luego de la ocurrencia del fallo.

Concurrencia: Un Sistema Distribuido debe estar diseñado para permitir el acceso a un mismo recurso por uno o más componentes de manera simultánea. Es necesario que la concurrencia de las operaciones sobre la información no afecte su integridad ni su consistencia.

Transparencia: La transparencia es la cualidad del sistema que permite que este sea apreciado como un todo y que la complejidad de las interacciones entre los distintos componentes esté oculta para los propios componentes y para los usuarios finales del sistema. Esta cualidad es una de las más relevantes debido a

que permite la abstracción de la complejidad inherente de los Sistemas Distribuidos. La transparencia es aplicable a todos los factores anteriormente mencionados y que de una forma u otra se ven beneficiados.

- **Transparencia de ubicación**: permite el acceso a los recursos sin tener la necesidad de conocer su ubicación.
- **Transparencia de acceso**: permite que los recursos locales y remotos sean accedidos de la misma manera.
- **Transparencia de concurrencia**: permite el acceso a un mismo recurso de manera simultánea por varios usuarios o componentes.
- **Transparencia de fallos**: garantiza que el sistema continúe funcionando a pesar de la ocurrencia de un fallo en uno o varios componentes.
- **Transparencia de movilidad**: posibilita que usuarios y recursos cambien su posición respecto al sistema sin afectar el correcto funcionamiento de este último.
- **Transparencia de replicación**: garantiza la redundancia de instancias de los recursos. La redundancia de recursos incrementa la disponibilidad del sistema y la calidad de su tolerancia a fallos.
- **Transparencia de rendimiento**: permite que el sistema se reconfigure en función de la variación de carga.
- **Transparencia de escalabilidad**: permite al sistema y las aplicaciones componentes ser aumentar la escala sin cambiar su estructura ni los algoritmos de cómputo empleados en la prestación de los servicios.

Capítulo 3 Propuesta de Documentación de la Arquitectura de Sistemas Distribuidos.

Los Sistemas Distribuidos son complejos de diseñar. Quizás debido a la naturaleza secuencial y centralizada de lo que rodea a un individuo es difícil percibir la distribución como algo normal. En el Capítulo 2 se mencionaban los retos de diseñar un Sistema Distribuido. Muchos de ellos estaban guiados a minimizar el impacto de la distribución y a aumentar la percepción de un Sistema Distribuido como un todo. Lograr la visualización de sistemas con estas características más allá de la simple suma de las partes componentes y para comprender que el todo es más que la suma de sus elementos es necesaria una abstracción lo suficientemente completa para describir cada uno de los aspectos relevantes dentro de la estructura de un Sistema Distribuido.

Después de hacer un balance entre las Arquitecturas de referencia en la construcción de los Sistemas Distribuidos y los retos enfrentados, se puede concluir que esta necesita ser documentada. El valor de la Arquitectura de Software como un medio para la comunicación de las decisiones relevantes entre los stakeholders y como un plano de construcción de un sistema se enfatiza en la concepción de los Sistemas Distribuidos debido a la interacción de múltiples componentes en los escenarios más variados. **Una documentación correcta tributa a la Arquitectura de los Sistemas Distribuidos un valor agregado imprescindible para su construcción.**

Documentación de la Arquitectura de Software.

La Documentación de la Arquitectura de Software -la documentación, de aquí en lo adelante- es la herramienta que permite argumentar el valor de la Arquitectura de Software como el vehículo primario para la comunicación entre stakeholders, además de cumplir el propósito de servir como medio de educación a estos. La documentación de la Arquitectura debe servir para varios propósitos bien definidos y para esto se caracteriza por:

- Ser lo suficientemente abstracta para ser comprendida por nuevos stakeholders.

- Ser suficientemente detallada para constituir un plano de construcción del sistema documentado.
- Contener la cantidad de información suficiente para servir como base para el análisis.

La intención de la documentación es a la vez prescriptiva y descriptiva. Para algunos stakeholders prescribe lo que debe ser considerado como cierto, en referencia a las restricciones impuestas sobre el diseño y las decisiones que se han tomado. Para otros describe lo que ya es considerado como cierto, sirviendo como base para el conocimiento de decisiones que ya fueron tomadas.

El término documentación "*denota la creación de un artefacto: generalmente, documentos, que pueden, por supuesto, ser archivos electrónicos, páginas Web, o papeles. Así, documentar una arquitectura de software se convierte en una tarea concreta: producir un documento de arquitectura de software. (…) La documentación puede ser formal o no, como sea apropiado, y puede contener modelos o no, como sea apropiado. Los documentos pueden describir, o pueden especificar.*"[32]

Vistas, Tipos de Vistas, y Estilos en la Documentación Arquitectónica

La documentación de una Arquitectura no es más que la documentación de las vistas que recogen los aspectos más relevantes de la estructura de un Sistema. Por vista se entiende lo que se establece en (Clements, et al., 2003) donde "*una vista es la representación de un conjunto de elementos del sistema y las relaciones asociadas con estos*"[33]. Aunque también en (IEEE, Septiembre, 2000) una vista es "*una representación de toda o parte de una arquitectura, desde la perspectiva de una o más incumbencias las cuales son mantenidas por uno o más de sus stakeholders.*".

Las vistas exponen –a diferentes niveles de abstracción- cualidades sistémicas, es por ello que en función de los stakeholders y la forma en que estos se involucran en el sistema se selecciona documentar una vista u otra. Es necesario comprender que la Arquitectura de Software es un ente complejo, por tanto, ninguna vista es capaz de documentar la Arquitectura en su totalidad, sin embargo un conjunto de vistas

[32] (Clements, et al., 2003) pp. 22
[33] (Clements, et al., 2003) pp. 32

puede converger hacia la completitud de esta documentación y documentar con la mayor exactitud posible los aspectos más relevantes de la Arquitectura de un Sistema de Software.

Entre estas vistas, las más difundidas quizás sean el conjunto de vistas de las 4+1 Vistas de Krutchen en (The 4+1 View Model of Architecture, 1995), UML empleado para la documentación arquitectónica y las Vistas de SEI-CMU –propuestas estas últimas en (Clements, et al., 2003).

Las 4+1 Vistas de la Arquitectura en RUP

Como todo en el Proceso Unificado de Rational, la Arquitectura también se analiza a través de los casos de usos, o más exactamente, a través de los casos de usos arquitectónicamente significativos, los cuales documentan la intención del sistema así como su interacción con el ambiente y constituyen el método de contrato entre desarrolladores y clientes. Para documentar la Arquitectura en (The 4+1 View Model of Architecture, 1995) Krutchen propone 4 vistas adicionales a la ya mencionada Vista de Casos de Uso:

- *Vista Lógica*: contiene las clases más importantes del diseño.
- *Vista de Implementación*: recoge las decisiones arquitectónicas tomadas al respecto de la implementación.
- *Vista de Procesos*: contiene la descripción de las tareas, procesos e hilos involucrados.
- *Vista de Despliegue*: describe los nodos físicos para las configuraciones de plataformas más típicas que se pueden encontrar.

A la forma de Documentación de la Arquitectura que proponen Krutchen ligadas a RUP se le puede señalar que no provee ninguna forma de documentar interfaces, razonamientos de diseños y comportamientos dinámicos, defectos poco aceptables a la hora de modelar sistemas con niveles de interacción tan altos como los son los Sistemas Distribuidos.

UML empleado para la documentación arquitectónica

A pesar de que UML no fue definido como un Lenguaje de Descripción Arquitectónica, puede ser utilizado como tal. Los diagramas de estructura estática: de clases y de objetos, los diagramas de componentes, de despliegue y los diagramas de comportamiento: casos de uso, secuencia, colaboración, de actividades y

de estados, proveen un visión detallada de la estructura de un sistema y han probado ser útiles en su documentación.

A UML se le han señalado un grupo importante de deficiencias a la hora de su empleo para describir Arquitecturas de Software. En (Reinoso, 2004) se señalan algunas de las que se han seleccionado las más relevantes desde la perspectiva de esta investigación:

- Se admiten las deficiencias de UML en cuanto al modelado de subsistemas distribuidos, los mapeos entre elementos de diferentes vistas para formar vistas combinadas –concepto que se verá más adelante-, además que no tiene soporte para modelar elementos de configuración dinámica de un sistema.
- Se ha documentado también las limitaciones que presenta para realizar el proceso conocido como *round-trip engineering*, debido a que existen relaciones binarias entre elementos que no se encuentran claramente definidas como los casos de asociación, agregación o composición. Esto provoca que herramientas de generación de código o de ingeniería inversa produzcan salidas incongruentes e inconsistentes cuando generan código a partir de un diagrama y cuando generan diagramas a partir de código.

Las Vistas de SEI-CMU[34]

En (Clements, et al., 2003) un grupo de investigadores pertenecientes a SEI-CMU proponen un grupo de vistas y tipos de vistas que permiten documentar la Arquitectura de Software a través de los estilos arquitectónicos utilizados en la construcción de la Arquitectura.

Además de la ya referenciada definición de vista arquitectónica, en (Clements, et al., 2003) se introduce la noción de *tipo de vista*. Un tipo de vista define "*los tipos de elementos y los tipos de relaciones usadas para describir la arquitectura de un sistema de software desde una perspectiva particular.*"[35] Se exhiben

[34] Siglas en inglés de: Software Engineering Institute from Carnegie Mellon University (tr. Instituto de Ingeniería de Software de la Universidad de Carnegie Mellon).
[35] (Clements, et al., 2003) pp.35

además un conjunto de tipos de vistas que pretenden documentar tres niveles de abstracción en las que se debe concebir de manera simultánea la Arquitectura de un sistema:

- Cómo está estructurado en un conjunto de unidades de implementación.
- Cómo está estructurado en un conjunto de elementos con comportamiento en tiempo de ejecución e interacciones.
- Cómo se relaciona con elementos ajenos al software en su ambiente.

Para esto proponen tres tipos de vistas: *de módulos*, *de componentes y conectores*, y *de localización*. Cada uno de estos tipos vistas se emplea en la documentación de estilos que comúnmente son utilizados en los niveles de abstracción referidos. A cado uno de estos tipos se han asociado catálogos de estilos, y a cada estilo se le ha aplicado un elemento de documentación conocido como *guía de estilo.* Una guía de estilo especifica el vocabulario de diseño -conjunto de tipos de elementos y relaciones- y las reglas -conjunto de restricciones topológicas y semánticas- de cómo estos pueden ser utilizados.

El lugar dentro de los estilos dentro de la documentación de la Arquitectura propuesta por SEI-CMU, es la consideración de estos como un refinamiento o especialización de los tipos de vistas. Para documentar las vistas, los tipos de vistas y los estilos, se propone una tabla que representa el contenido de una guía de estilo. La definición de cada uno de los campos de la tabla y su propósito se encuentran detallados en la Tabla 3.1.

Tabla 3.1 Elementos de una Guía de Estilos/Tipos de Vistas

Campos de la Tabla	Descripción
Elementos	Los elementos son los bloques de construcción arquitectónicos nativos al estilo o tipo de vista. La descripción de los elementos documenta el rol que juega en la Arquitectura y fundamenta las bases para su documentación de manera efectiva en las vistas.
Relaciones	Las relaciones determinan como los elementos cooperan para acometer el trabajo del sistema. El término *relación* denota las relaciones entre los elementos y provee reglas sobre cómo estos pueden o no estar relacionados.
Propiedades	Las propiedades son información adicional acerca de los elementos y sus relaciones asociadas.
Restricciones	Establecen las limitaciones semánticas y topológicas sobre los elementos y las relaciones de un estilo o tipo de vista.

Tipo de Vista de Módulos

Permite documentar las estructuras modulares de un sistema de software mediante la enumeración de unidades de implementación o módulos de un sistema, conjunto con las relaciones entre estas unidades. El Tipo de Vista de Módulos identifica cuatro estilos arquitectónicos:

- *Estilo de descomposición*: representa la descomposición del código en sistemas, subsistemas, subsubsistemas, etc. Representa un vista de arriba hacia abajo del sistema.
- *Estilo de usos*: indica las relaciones de dependencia funcional entre módulos.
- *Estilo de generalización*: indica las relaciones de especialización/generalización entre módulos.
- *Estilo en capas*: se utiliza para indicar la relación *permitido-usar* de manera restricta entre módulos.

Tipo de Vista de Componentes y Conectores

Las vistas basadas en el Tipo de Vista de Componentes y Conectores definen modelos consistentes en elementos que tiene presencia en tiempo de ejecución (Componentes), tales como procesos, objetos, clientes, servidores, almacenes de datos, entre otros. Además en estas vistas se documentan mecanismos de interacción (Conectores) tales como enlaces de comunicación, protocolos, flujos de información, acceso a recursos compartidos y más.

Este Tipo de Vista es factible para la documentación de los estilos de:

- *Estilo de tuberías y filtros*: donde el patrón de interacción es caracterizado por transformaciones sucesivas de los datos.
- *Estilo de datos compartidos*: centrado en la retención de datos persistentes. Múltiples elementos pueden acceder a los datos persistentes retenidos en al menos un repositorio de datos.
- *Estilo de publicación/suscripción*: se caracteriza por describir componentes que interactúan mediante el anuncio de eventos.
- *Estilo cliente-servidor:* muestra la interacción entre componentes mediante la petición de servicios de manera jerárquica y estructurada.

- *Estilo de igual-a-igual (peer-to-peer)*: se caracteriza por la interacción directa y anárquica entre componentes intercambiando servicios.
- *Estilos de procesos y comunicación*: se distingue por la interacción de componentes que se ejecutan concurrentemente a través de varios mecanismos conectores.

Tipo de Vista de Localización

Es necesario también documentar las interacciones de los elementos del ambiente (hardware, sistemas de fichero, estructura y personal de los equipos de desarrollo, etc) con el sistema. Con este objetivo se define el Tipo de Vista de Localización. Este se centra en la definición de elementos -de las vistas de módulos y componentes y conectores - y relaciones de correspondencia de los elementos de software con el hardware u otro tipo de elementos ambientales.

Las vistas de localización son útiles a la hora de documentar:

- *Estilo de despliegue*: describe la correlación entre los componentes y conectores con el hardware donde son ejecutados.
- *Estilo de implementación*: describe la localización de los módulos en el sistema de archivos que los contiene.
- *Estilo de asignación de tareas*: describe la correlación entre los módulos y las personas, grupos y equipos que se le asigna el trabajo con los módulos.

Como quizás se haga evidente por el diferente grado de profundidad con que se han abordado en relación con el resto de los conjuntos, las vistas de SEI-CMU se han considerado en el marco de esta investigación como las más apropiadas para documentar la Arquitectura de Sistemas Distribuidos.

Su eminente orientación hacia los estilos arquitectónicos se aprovecha de las bondades expuestas sobre estos últimos en el Capítulo 1, fundamentalmente aquellas relacionadas con el análisis automático de propiedades de los estilos y la facilidad de representación gráfica. Además de esta eminente orientación hacia estilos, las vistas de SEI-CMU promueven la extensibilidad mediante el establecimiento de una

metodología para describir estilos, vistas y tipos de vistas propios. El predicamento de que ningún grupo de vista es exactamente adecuado para documentar un sistema invita a los arquitectos y documentadores de manera general, a crear su propio conjunto de vistas que se adapten con mayor facilidad a lo que pretendan documentar.

Otra característica importante a tener en cuenta es que, a diferencia de otras vistas, en (Clements, et al., 2003) logran expresar los grupos de vistas más importantes en los términos de las guías de estilos propuestas. Cabe agregar que además, la mayoría de las Arquitecturas analizadas en el Capítulo 2 tiene un soporte implícito en los tipos de vistas propuestos, lo que simplifica el trabajo de definición de vistas específicas para Sistemas Distribuidos.

Vistas combinadas

A pesar del carácter de divide y vencerás de la Arquitectura de Software y de su documentación como valor inherente, sería imposible entender la Arquitectura de un sistema como un todo desde su documentación, si las vistas que describen los diferentes aspectos no se pueden interrelacionar. La interrelación entre las vistas es una parte importante de la documentación y debe quedar plasmada en algún lugar dentro de la misma.

En algunos casos más allá de plasmar esta relación entre las vistas, es recomendable combinarlas en una solo vista que recoja los aspectos de ambas en una sola *vista combinada.* Una vista combinada es "*una vista que contiene elementos y relaciones que provienen de dos o más vistas*"[36]. Esta combinación puede provenir de combinar vistas o estilos arquitectónicos.

En algunos casos durante la construcción de un sistema los arquitectos combinan estilos para atribuir al sistema con las propiedades que cada uno promueve, en estos casos se dice que se está creando un *estilo híbrido*. Más concretamente, un *estilo híbrido* es "*la combinación de dos o más estilos existentes (…) En adición, la correspondencia entre los estilos que constituyen el híbrido debe ser documentada. Los estilos híbridos, cuando son aplicados a un sistema en particular, producen vistas.*"[37]

[36] (Clements, et al., 2003) pp.184
[37] (Clements, et al., 2003) pp.184

En otros casos, con el objetivo de analizar una porción de la Arquitectura o comunicarla a algún grupo de stakeholders es necesario combinar las representaciones de diferentes vistas se crea una *vista solapada.* Las vistas solapadas son combinaciones de las representaciones primarias de dos o más vistas con el propósito de analizar o comunicar porciones de la arquitectura.

Perspectivas arquitectónicas

Uno de los problemas fundamentales dentro del campo de la Arquitectura de Software es asegurar que los sistemas poseen ciertas y determinadas cualidades –cualidades sistémicas- importantes para sus stakeholders. Es destacable que ni las vistas, ni los tipos de vistas y aunque en mayor medida, los estilos arquitectónicos, aseguran por completo la presencia de alguna cualidad en específico en un sistema.

Para promover un enfoque guiado a asegurar la presencia de una cualidad en una Arquitectura dada (Woods, y otros, 2006) definen el término de *perspectiva arquitectónica* como "*una colección de actividades, listas de chequeo, tácticas y directrices que guían el proceso de asegurar que un sistema exhiba un conjunto particular de propiedades de calidad estrechamente relacionadas, las cuales requieren consideración a través de un número de vistas arquitectónicas del sistema*"[38].

La característica que se considerará para esta investigación es que las perspectivas arquitectónicas se aplican sobre las vistas para asegurar que las cualidades especificadas se encuentren presentes. La estructura de una perspectiva debe contener los campos establecidos en la Tabla 3.2

Tabla 3.2 Estructura de una perspectiva arquitectónica

Campo de la Perspectiva	Descripción
Incumbencia	Describe la propiedad a la que se refiere la perspectiva.
Aplicabilidad	Denota la posibilidad de aplicar la perspectiva a diferentes vistas arquitectónicas de un sistema.
Actividades	Conjunto de actividades que se sugieren como parte del proceso de aseguramiento de la presencia de la propiedad en las vistas.
Tácticas Arquitectónicas	Conjunto de tácticas arquitectónicas[39] que los arquitectos consideren como parte del diseño.
Problemas y dificultades	Lista de problemas y dificultades de las que el arquitecto debe permanecer alerta y las soluciones más comunes a estas.
Lista de chequeo	Lista de puntos que el arquitecto debe utilizar para verificar que no se ha pasado por alto ningún aspecto.

[38] (Woods, y otros, 2006) pp.2
[39] Ver (Bass, et al., 2003) Capítulo 5.

Reglas para una correcta documentación.

Para evaluar la corrección y completitud de la documentación de un sistema de software (Clements, et al., 2003) plantean siete reglas o principios básicos que deben guiar a los que pretendan construir una documentación con ciertos parámetros de calidad.

1. *Escribir la documentación desde el punto de vista del lector.*

   Considerar que a pesar de que un documento pueda ser escrito una vez, puede ser revisado y leído cientos de veces. Hacer que el lector considere que la documentación fue creada para ayudarlo a entender es útil para que este la utilice una y otra vez para extraer el conocimiento reflejado en ella. Se considera una práctica correcta escribir para el lector, no importa el nivel de preparación o procedencia. Por esto es recomendable además evitar la utilización de términos propios de un grupo de stakeholders o alguna jerga en específico.

2. *Evitar la repetición innecesaria.*

   Cada tipo de información ***debe ser registrada en solo un lugar***. Esto facilita el uso de la documentación, además de hacer más llevadero el proceso de actualización de esta. Evitando la repetición innecesaria se elimina además confusión en caso de que la información sea repetida de manera diferente.

3. *Evitar la ambigüedad.*

   La razón fundamental de la Arquitectura es eliminar detalles innecesarios para construir un sistema. Evitar la ambigüedad de los elementos y las representaciones dentro de la documentación evita que no se interpreten o -en el peor de los casos- se mal interpreten las decisiones de diseño

documentadas. Para esto deben eliminarse todo tipo de ambigüedades lingüísticas y ***utilizar notaciones precisas y bien definidas***.

4. *Utilizar una organización estándar.*

   Es imprescindible establecer un esquema de organización estándar y planificado y lograr que la documentación se apegue a este, además de socializar el conocimiento de la existencia de dicho esquema. Esto posibilitará que los lectores de la documentación puedan encontrar con facilidad cualquier información dentro de esta, además de ayudar a aquellos que escriben la documentación como guía para identificar que resta por documentar.

5. *Registrar los razonamientos.*

   Registrar las decisiones respecto al diseño de la Arquitectura, las estrategias que se valoran y se desechan y las razones por las que se desechan constituye una forma de evitar revalorar un conjunto de caminos en momentos posteriores donde se necesite cambiar algunos aspectos. Esto posibilitará ahorrar grandes cantidades de tiempo, aunque se requiere la disciplina de ***registrar estas decisiones en el momento en que son tomadas***.

6. *Mantener la documentación actualizada, pero no tan actualizada.*

   La actualización de la documentación debe hacerse de manera periódica, en períodos no muy largos pero tampoco tan cortos, de manera que las decisiones que no persisten en el tiempo se vean constantemente afectadas y desafectadas. Se implica entonces que las decisiones de diseño concernientes a la Arquitectura de un sistema se reflejen no se reflejen en el momento que son tomadas. Aparentemente, esta regla contradice lo expuesto en la Regla 5, sin embargo, lo que se pretende es lograr un balance, considerando la Documentación de la Arquitectura un artefacto

más, sujeto al control de versiones y a las liberaciones. De aquí se pueda lograr mantener actualizada la documentación, pero hacer liberaciones solo versiones finales.

7. *Revisar que la documentación se ajuste a sus propósitos.*

   Es necesario poner en consideración de los stakeholders de la documentación -y de la Arquitectura de manera general- el trabajo realizado. La objetividad con que estos analicen los artefactos producidos y su beneplácito al respecto en cuanto a la forma y contenido de la documentación conllevarán a que la documentación cumpla sus propósitos. A fin de cuentas, la documentación es creada para sus stakeholders.

De acuerdo con la fuente, seguir estas reglas distingue una documentación precisa y usable de una documentación pobre e ignorada.

Modelos y metamodelos en la Documentación de la Arquitectura de Software

Quizás el mayor problema en el momento de documentar la Arquitectura de Software es lo complejo, tedioso y -en muchos casos- propenso a error, del completamiento de plantillas de documentación que proponen muchas metodologías y luego de la extracción del conocimiento representado sobre la estructura del sistema por parte de los stakeholders. En la mayoría de los casos, documentar grandes sistemas –como es el caso de la mayoría de los Sistemas Distribuidos- conlleva la construcción de extensos documentos de Arquitectura parar describir los distintos aspectos para los diferentes stakeholders de la Arquitectura relacionados con el proceso de construcción de sistemas con estas características.

Tratar de facilitar el proceso de documentación en proyectos de grandes dimensiones representa un reto para los Arquitectos de Software.

La mejor forma de entender conceptos y sus relaciones es a través de la representación gráfica. Ruth Malan en (Malan, 2009) expone la importancia de las ideas visuales en un proceso tan creativo como la construcción de la Arquitectura de un sistema. Señala Malan que las ideas visuales representan la mejor

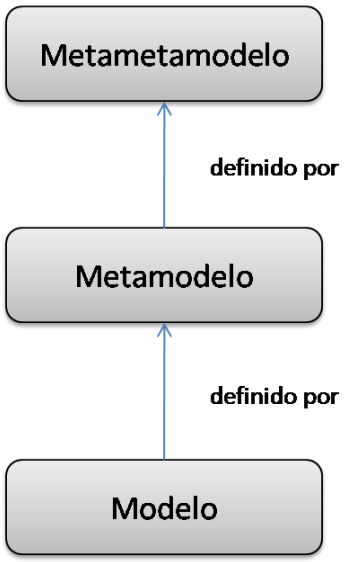
forma de comunicación, pensadores de la talla de Albert Einstein o Richard Feynman –aunque no muy relacionados con la Arquitectura de Software- resaltaban la importancia de las imágenes a la hora de manipular abstracciones.

De aquí que se consideren variantes de documentación con mucha más flexibilidad, fiabilidad y facilidad que los documentos de texto. En el epígrafe dedicado a las tendencias actuales dentro de la Arquitectura de Software en el Capítulo 1, se ha explicado la Arquitectura y el Desarrollo basados en Modelos. La utilización de modelos en la documentación de la Arquitectura es una tendencia que ha ido ganando adeptos dentro del campo de la Arquitectura de Software.

Un modelo para un objeto o ente es una colección de atributos y un conjunto de reglas que gobiernan como estos atributos interactúan. No existe un único modelo correcto para un objeto, para conocer sobre diferentes aspectos del objeto es generalmente necesario utilizar diferentes modelos, con mayor fundamento en aquellos casos donde la estructura del objeto sea compleja. Los modelos constituyen una herramienta para representar objetos, sus propiedades y relaciones. Ejemplo existen varios: los Modelos de UML, los propuestos por RUP basados en UML, entre otros.

Modelar es la acción de describir los conceptos asociados a un dominio, problema u objeto en función de conceptos provistos por un lenguaje de modelado. Un término muy afín a los modelos es el concepto de metamodelos. Los metamodelos permiten modelar lenguajes de modelado. Más claramente, un metamodelo es un modelo de un lenguaje de modelado. Para definir un metamodelo también es necesario un lenguaje conocido como lenguaje de meta-metamodelado, en la Figura 3.1 se representa la relación entre estos términos.

Figura 3.1 Relación entre modelo, metamodelo y meta-metamodelo



La definición de un modelo entonces puede seguir -a rasgos generales- los siguientes pasos:

- Definir los elementos del dominio a representar.
- Seleccionar las abstracciones capaces de describir los elementos definidos.
- Diseñar un metamodelo con el objetivo de crear un lenguaje de modelado que permitan la descripción de los conceptos, sus relaciones y propiedades.
- Construir el modelo.

Si tomamos como objeto a modelar la Arquitectura de Software, podemos equiparar los modelos que la describen con las vistas de la documentación de la Arquitectura. Llamémosle a esta correspondencia *Modelado por Vistas de la Arquitectura de Software.* Se define el término *Modelado por Vistas de la Arquitectura de Software* como el ***proceso de hacer corresponder una vista de la Arquitectura de Software con un modelo donde se representen los elementos, relaciones, propiedades y restricciones que se documentan en dicha vista.***

Modelos para la Documentación de la Arquitectura de Sistemas Distribuidos

La propuesta de Documentación basada en Modelos de la Arquitectura de Sistemas Distribuidos que resulta de la investigación expuesta en este documento consta de tres vistas fundamentales: la Vista de Módulo, la Vista de Aplicación y la Vista de Sistema. Cada una de ellas introduce o reutiliza elementos que constituyen el espacio conceptual de los *Modelos para la Documentación de la Arquitectura de Sistemas Distribuidos*.

Para documentar la Arquitectura de Sistemas Distribuidos se precisa determinar diferentes niveles de abstracción. Desde un nivel más concreto y cercano al diseño y la implementación, hasta un nivel de abstracción de sistemas, subsistemas y las formas de interacción entre estos. Además, las vistas deben ser capaces de atacar los retos en el diseño de Sistemas Distribuidos, así como los problemas que se han planteado como marco teórico para el surgimiento de la investigación.

En la Tabla 3.3 se muestran las características necesarias que deben presentar los modelos para solucionar los problemas planteados en la construcción y documentación de Sistemas Distribuidos. Sobre la base de estas características y como primer paso para la construcción de los *Modelos para la Documentación de la Arquitectura de Sistemas Distribuidos*, se ha construido el *Metamodelo para la Documentación de la Arquitectura de Sistemas*.

Tabla 3.3 Características esperadas para los Modelos de Documentación de la Arquitectura de Sistemas Distribuidos

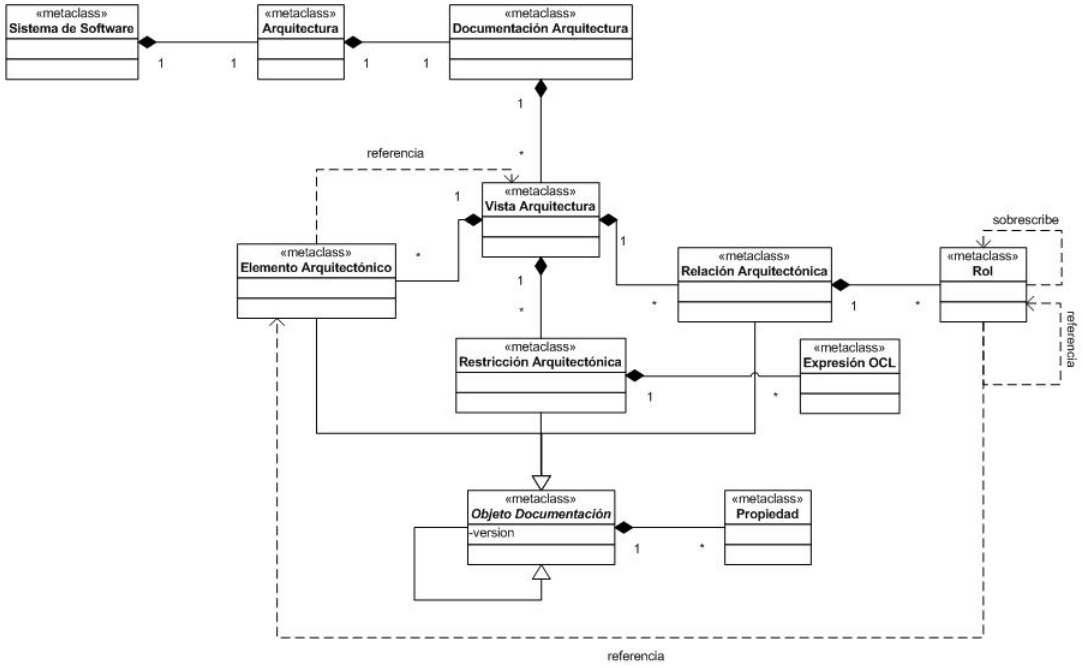
Características esperadas	Nivel de Abstracción
Generales	
Relaciones de especialización entre elementos	Metamodelo
Independientes de plataforma	Modelos
Transformables a una plataforma	Modelos
Referencias entre modelos	Metamodelo
Relaciones y restricciones como elementos de primer orden	Metamodelo
Versiones de objetos de documentación	Metamodelo
Estandarización de la documentación	Metamodelo
Arquitectura de Sistemas	
Representación de estilos arquitectónicos	Metamodelo
Elementos arquitectónicos	Metamodelo
Relaciones arquitectónicas	Metamodelo
Restricciones arquitectónicas	Metamodelo
Vistas selectivas por niveles de abstracción	Metamodelo
Extensión de estilos, vistas y tipos de vistas	Metamodelo
Vistas combinadas	Metamodelo
Reconstrucción de la Arquitectura	Modelos
Diferenciación de liberaciones del sistema	Modelos
Comparativa y análisis de la Arquitectura	Modelos

Metamodelo para la Documentación de la Arquitectura de Sistemas

Con el objetivo de construir modelos capaces de corresponder con las Vistas de la Arquitectura de Software es necesario encontrar el dominio apropiado y las relaciones entre estos elementos de dominio y sus propiedades. Como se había establecido, para cumplir este propósito las Vistas de SEI-CMU han definido –de manera implícita- un conjunto de términos capaces de representar un gran número de vistas de la Arquitectura. Los conceptos de elementos, relaciones, propiedades y restricciones definen el espacio conceptual de las vistas –ver Tabla 3.1.

Así entonces el metamodelo propuesto en la Figura 3.2 representa la conversión de los elementos de la Tabla 3.1 y sus relaciones con la finalidad de construir modelos basados en las Vistas de la Arquitectura de Software. Además de los términos propuestos por las Vistas de SEI-CMU, se han adicionado relaciones y términos con el fin de enriquecer el metamodelo y extender sus capacidades de representación de las Vistas de la Arquitectura de Software.

Figura 3.2 Metamodelo para la Documentación de la Arquitectura de Sistemas.



Este proceso de meta-modelar la Arquitectura de Software es conocido como meta-arquitectura. En el espacio de los formalismos, una meta-arquitectura es un Lenguaje de Descripción Arquitectónico formal o semi-formal. Definir una meta-arquitectura facilita:

- *Estandarizar*

  Describir varias arquitecturas utilizando la misma meta-arquitectura confiere a esta un rol de estandarización para al menos las arquitecturas que son descritas utilizándola.

- *Comparar*

  Crear una meta-arquitectura se considera una buena herramienta para comparar varias arquitecturas. Describir varias arquitecturas sobre la misma meta-arquitectura posibilita su comparación y análisis.

- *Integrar*

  Describir varias arquitecturas sobre la base de una misma meta-arquitectura facilita integrar e intercambiar varias arquitecturas.

- *Formalizar*

  La meta-arquitectura puede ser empleada para semi-formalizar la documentación de la Arquitectura. Esta semi-formalidad les brinda a los modelos construidos sobre la base de meta-arquitecturas semi-formales ventajas sobre los Lenguajes de Descripción Arquitectónicos debido a que requieren de menos formalismo y teorías matemáticas en muchos casos, haciéndolos más fáciles de utilizar.

Modelo de Módulo

El Modelo de Módulo es –de los tres Modelos de Documentación de la Arquitectura de Sistemas Distribuidos- el de más bajo nivel de abstracción. Se considera que el módulo es la unidad de abstracción más cercana a la implementación del sistema. Como se puede apreciar en la Figura 3.3 el Modelo de Módulo introduce los conceptos de módulo y rama.

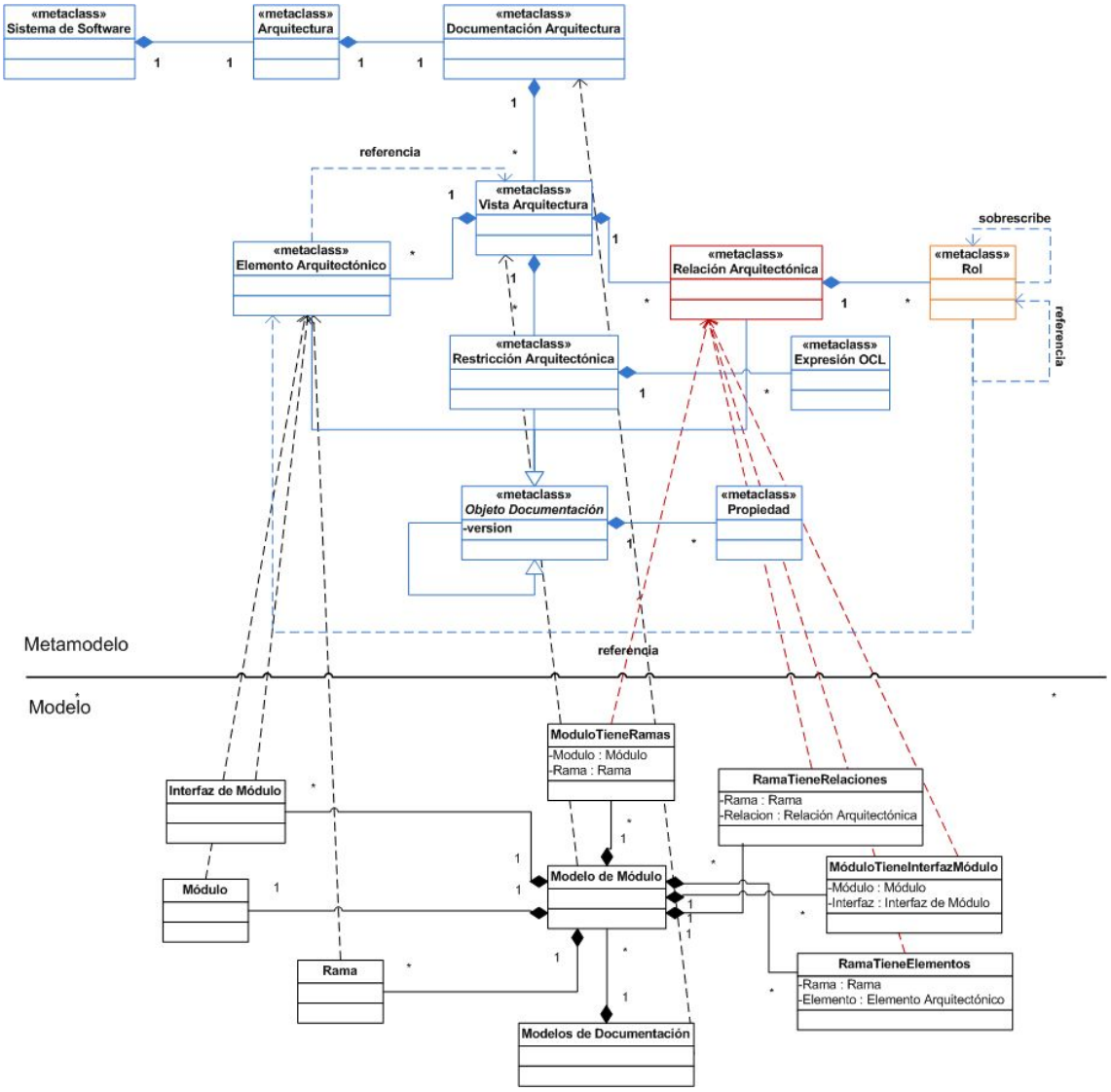
Por módulo se entiende lo que establece (Sánchez Téllez, 2008) como un "*conjunto de funcionalidades acopladas convenientemente para realizar una lógica de negocio determinada. Un módulo es totalmente autónomo por lo que define claramente sus fronteras administrativas de seguridad, auditoría, mecanismos de interacción, configuración y aislamiento*"[40] acotando que la lógica puede contener no solamente procesos de negocio sino también otros procesos. Cabe agregar que un módulo es un elemento de la arquitectura que contiene unidades de implementación genéricas y sus relaciones, permitiendo la extensibilidad y la utilización de estilos arquitectónicos dentro de la estructura modular. Para hacerlo más claro, dentro de un módulo los elementos pueden quedar organizados cumpliendo cualquier estilo o patrón arquitectónico, patrón y estilo por su cercanía a la frontera entre las decisiones de diseño y las de nivel de arquitectura.

[40] (Sánchez Téllez, 2008) pp.22.

De las funcionalidades especificadas en la Tabla 3.3, más allá del control de versiones que se aplicó de manera general a nivel de metamodelo, el Modelo de Módulo resuelve la necesidad de codificación paralela horizontal de componentes. Al ser el modelo más cercano a la implementación y el destino de representación de las unidades de implementación –librerías de enlace dinámico, clases, archivos .class, variando en función de una tecnología-, se ha decidido introducir a este nivel de abstracción el concepto de *rama*.

Muchos de los patrones de diseño empleados en la estructuración de las unidades de implementación resuelven el problema de la codificación en paralelo de componentes con relaciones de dependencia fuertes. En momentos durante el desarrollo de sistemas de manera general -y Sistemas Distribuidos, más específicamente- se puede hacer necesario impulsar el trabajo en determinados componentes introduciendo uno o más equipos de desarrollo. Es en este momento donde surge un conflicto en el control de versiones debido a que las unidades de implementación se encuentran generalmente versionadas y los cambios hechos por un equipo pueden afectar la implementación del otro.

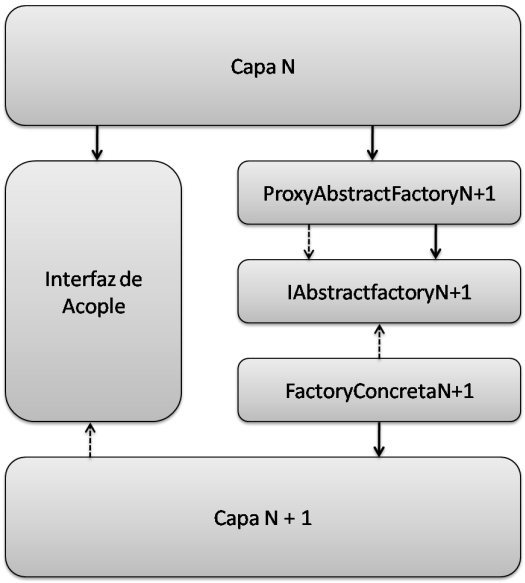
Figura 3.3 Modelo de Módulo



Se muestra un ejemplo concreto en la Figura 3.4, donde se evidencia la utilización de algunos patrones de diseño –fachada, proxy, fábrica concreta, fábrica abstracta y otros- para desacoplar la relación *permitido-usar* entre dos capas de una arquitectura con estilo multicapa. Es evidente que manteniendo estables las unidades de desacople es posible trabajar de manera simultánea en las capas N y N+1. Sin

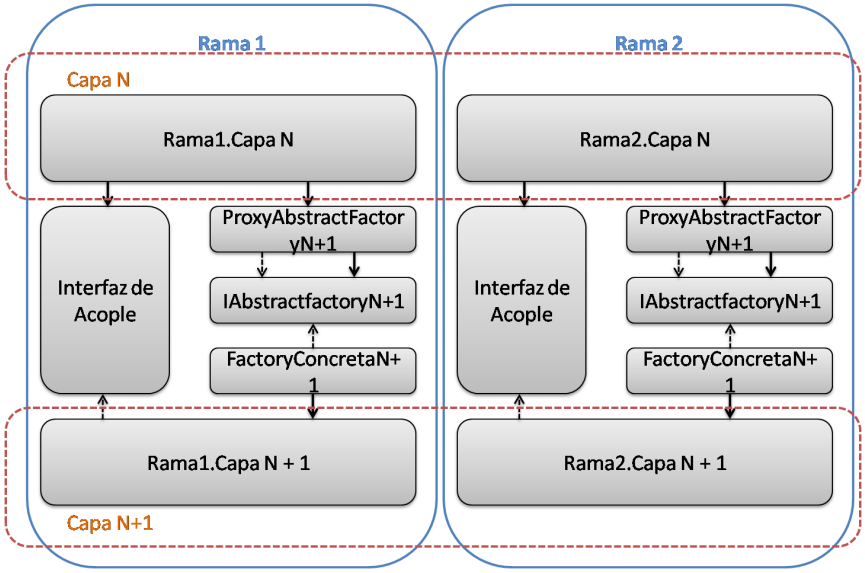
embargo, si se pretenden codificar paralelamente funcionalidades dentro de una misma capa –codificación horizontal- se hace particularmente difícil coordinar el trabajo.

Figura 3.4 Desacople entre capas usando patrones, adaptada de (Sánchez Téllez, 2008)



En el intento de coordinar este trabajo desde un nivel arquitectónico, se introduce el concepto de rama. La rama es, conceptualmente, ***la separación horizontal de un componente en componentes de igual propósito en el desarrollo***. En la práctica representa la solución al problema de codificación horizontal de componentes.

Figura 3.5 La rama en la codificación horizontal de componentes



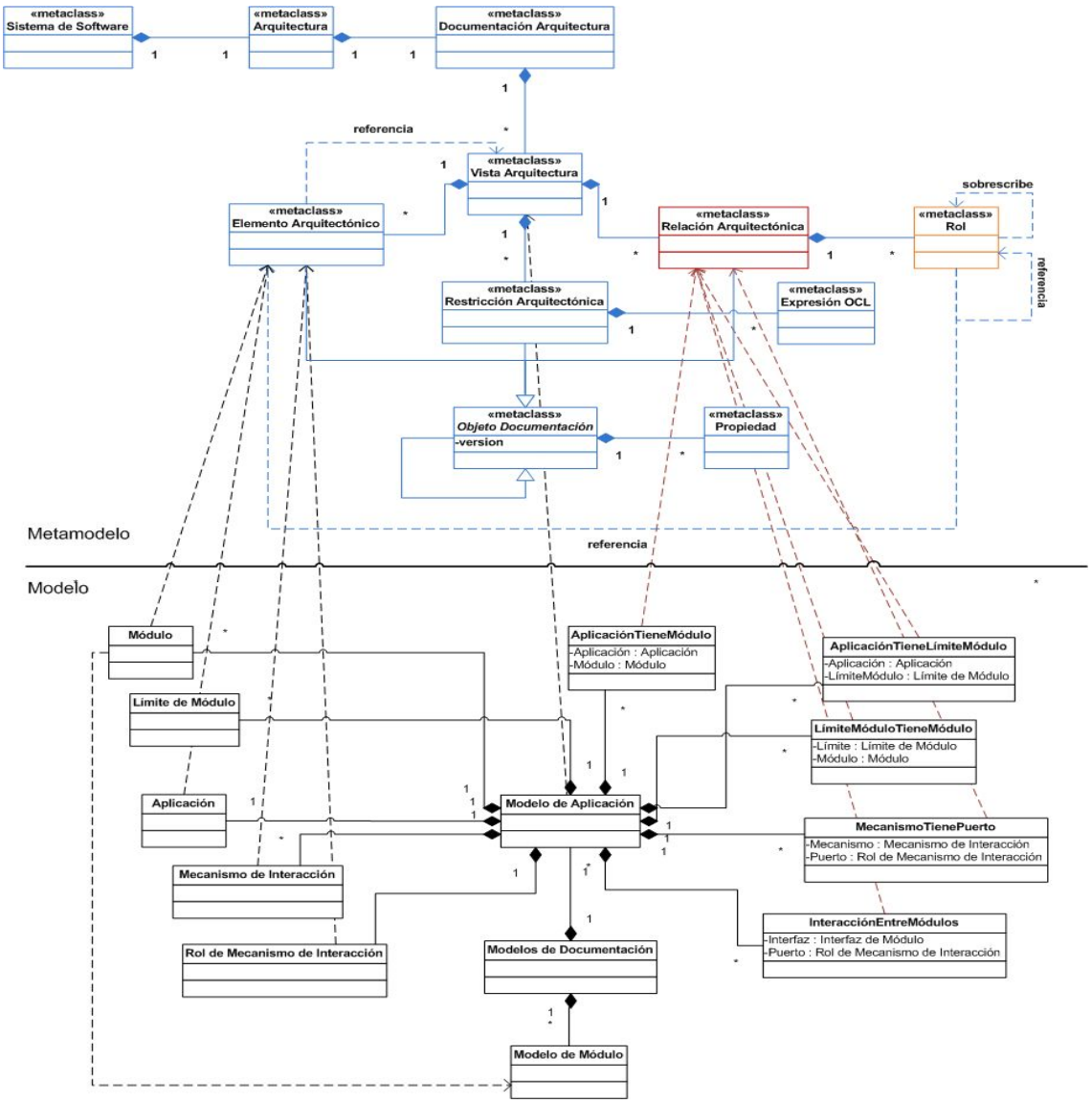
La Figura 3.5 ejemplifica la utilización de la rama para la separación de componentes en componentes de igual jerarquía vertical en una arquitectura multicapas. La heurística de descomposición de componentes puede ser tan compleja como se quiera, facilitando estructuras o estilos que sean inmutables para las ramas –ejemplo: componentes de dominio, mismo estilo para todas las ramas, entre otras- y las más diversas lógicas de descomposición. Señalar que se evidencia en la Figura 3.5 como la Capa N se descompone en dos subcapas, una para cada rama, sin embargo a nivel de módulo sigue existiendo un único componente Capa para ese nivel de abstracción.

La rama es posible de implementar usando diferentes tecnologías como el caso de las clases parciales en el lenguaje C# o en ensamblados multi-archivos dentro de la plataforma Microsoft .NET, u otras soluciones en varias plataformas. A pesar de no ser creada con ese objetivo, la rama puede contribuir además a solucionar el problema de la diferenciación de liberaciones del sistema en función de las versiones y partes componentes de este. También se puede equiparar la rama al concepto de *branch(rama)* que implementan los Sistemas de Control de Versiones.

Modelo de Aplicación

En el siguiente nivel de abstracción en los Modelos de Documentación de la Arquitectura de Sistemas Distribuidos se encuentra el Modelo de Aplicación -Figura 3.6. El Modelo de Aplicación propone una arquitectura diseccionada en unidades cohesionadas de funcionalidades conocidas por módulos y definidas en el Modelo de Módulo. De aquí que para el Modelo de Aplicación una aplicación sea ***la agrupación lógica ejecutable de un conjunto de módulos que interactúan a través de mecanismos que conectan las interfaces provistas/esperadas de los módulos permitiendo la combinación de las funcionalidades implementadas en dichos módulos.***

Figura 3.6 Modelo de Aplicación



La utilización de la estructura modular no es una elección casual. La descomposición de una aplicación en módulos permite fomentar requisitos de desacoplamiento entre funcionalidades, reutilización de las unidades de implementación definidas dentro de diferentes módulos, entre otras ventajas.

En cuanto a las posibilidades descriptivas del Modelo de Aplicación es posible afirmar que con este se puede describir una gran variedad de aplicaciones con diferentes estilos arquitectónicos. Al estar construido sobre la base del Metamodelo para la Documentación de la Arquitectura de Sistemas, el Modelo de Aplicación hereda las propiedades de extensibilidad inherentes a su metamodelo. Este modelo está construido con la perspectiva del tipo de vista arquitectónica de Componentes y Conectores propuesta en las Vistas de SEI-CMU y que está orientada a las vistas sobre componentes que tienen presencia en tiempo de ejecución como es el caso de una aplicación.

El Modelo de Aplicación ataca el problema de la pobre descripción de los contextos de ejecución de código existentes en distintas plataformas –léase Dominios de Aplicación en .NET Framework, contextos es Servidores Web, entre otros- a través de la introducción de la noción de *límite de ejecución de módulo.* Los límites de ejecución de módulo ***definen las propiedades y reglas que pueden regir el contexto de ejecución de un código determinado, así como las configuraciones de las que este código depende.*** En el marco de la Arquitectura de Software es importante la documentación de los contextos donde se ejecutan las rutinas de código. Contextos que pueden definir parámetros de seguridad, rendimiento, depuración, entre otros requisitos no funcionales que afectan la ejecución.

A destacar también dentro del Modelo de Aplicación la facilidad de descripción de los mecanismos de comunicación entre los módulos. Modelados como elementos de primer orden, los mecanismos de interacción ***permiten definir patrones de interacción entre módulos y establecer las interfaces necesarias para utilizarlos***. Los mecanismos de interacción se basan en la idea de conectores, presentada por las vistas de Componentes y Conectores donde el *Rol de Mecanismo de Interacción* constituye la interfaz externa de dicho mecanismo. Avalados por la relación entre un Elemento y una Vista Arquitectónica, los mecanismos pueden expresarse a través de relaciones entre componentes mucho más granulares.

Modelo de Sistema

El nivel de abstracción más elevado desde el que se propone analizar un Sistema Distribuido se documenta a través del Modelo de Sistema -Figura 3.7. En este modelo se documentan las relaciones entre las aplicaciones de un Sistema Distribuido, además de que se modelan los subsistemas que lo

componen. Por subsistema se entiende lo que se plantea en (Bachmann, y otros, 2000) como "*una parte de un sistema que porta un subconjunto cohesionado de funcionalidad de toda la misión del sistema y que puede ser ejecutado independientemente. Los subsistemas son subconjuntos del sistema que pueden ser desarrollados y desplegados incrementalmente*"[41].

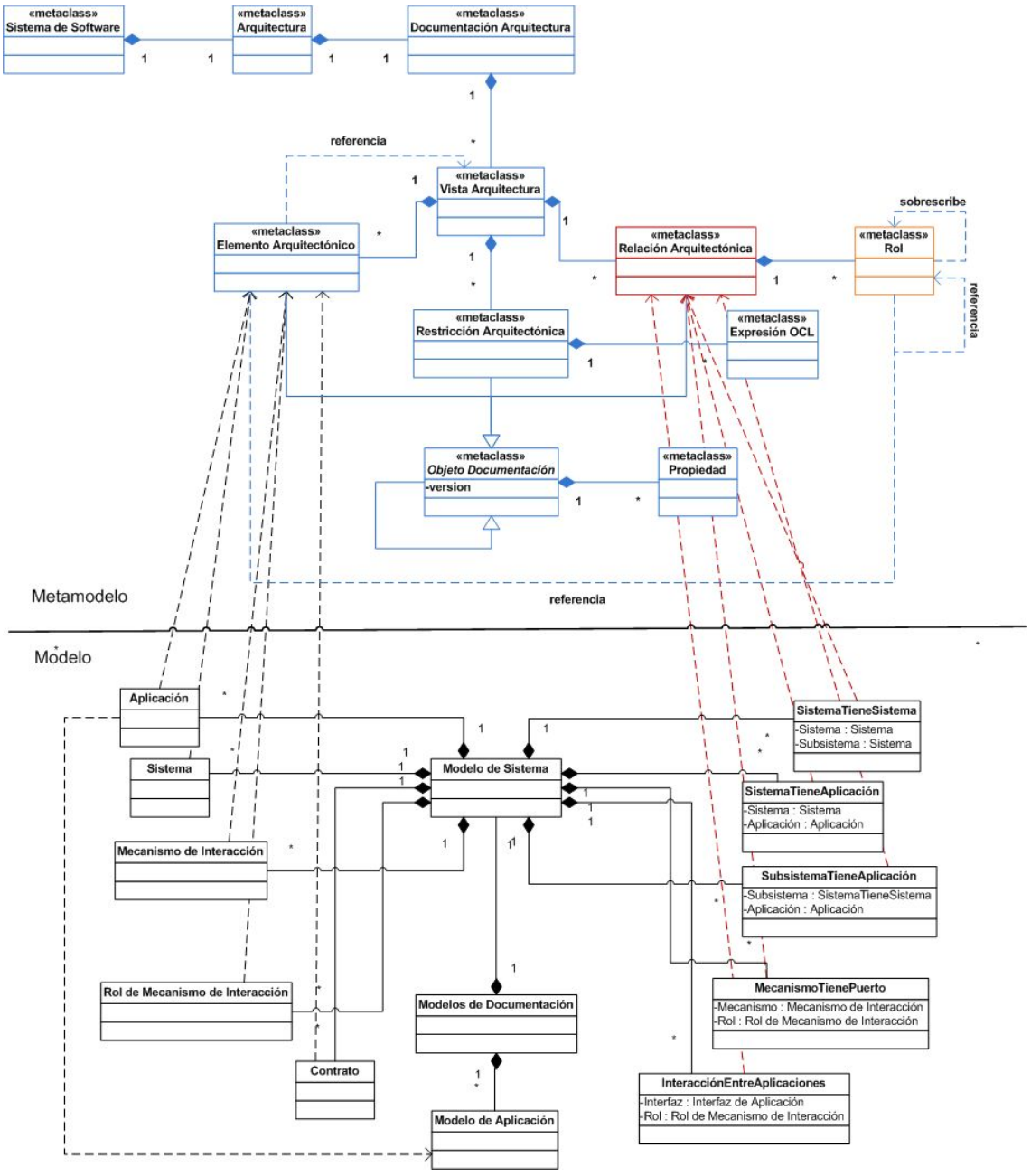
Una característica importante que aporta esta definición es que los subsistemas pueden ser desarrollados incrementalmente, y se le pudiera agregar que paralelamente, por equipos de desarrollo que se encuentren en muchos casos en localizaciones distantes. Se ha enfatizado a lo largo de esta investigación de la ingente necesidad de comunicación entre los stakeholders de la Arquitectura, sin embargo es necesario resaltar una vez la importancia que reviste para los desarrolladores e incluso arquitectos de diferentes subsistemas conocer lo que otros desarrolladores y arquitectos esperan de sus subsistemas y lo que se puede esperar que otros provean.

Para fomentar el desarrollo colaborativo y proveer un medio de comunicación entre desarrolladores y arquitectos de subsistemas se introduce el concepto de *contrato*. Un contrato ***refiere las responsabilidades que adquiere un subsistema con otro cuando existe una interacción entre las aplicaciones contenidas dentro de cada uno.*** Se aconseja la utilización de los contratos en el Modelo de Sistemas principalmente en los casos donde exista la interacción planteada y además los subsistemas estén siendo desarrollados por equipos de desarrollos diferentes o que se encuentren separados.

De manera general, este modelo permite hacer un análisis de cómo se elimina la complejidad impuesta por varios de los Retos en el diseño de Sistemas Distribuidos. La concreción a nivel de sistema de las relaciones de los elementos en los distintos niveles permite modelar con vista a la heterogeneidad característica en los Sistemas Distribuidos. En lo modelos de menor nivel de abstracción se han definido de manera explícita las interfaces de los componentes que se utilizan, promoviendo la reutilización, la apertura y la extensibilidad de los mismos. También se han descrito las dependencias entre estos componentes.

[41] (Bachmann, y otros, 2000) pp.10.

Figura 3.7 Modelo de Sistema



Al nivel de abstracción que propone el Modelo de Sistema, las relaciones entre aplicaciones permiten analizar las posibilidades de ocurrencias de fallos dentro del sistema debido al mal funcionamiento de sus dependencias o las fallas en los mecanismos de interacción entre estas. Otro de los factores del diseño de Sistemas Distribuidos que el Modelo de Sistema solventa es la transparencia. Utilizando el Modelo de Sistema es posible apreciar un Sistema Distribuido como un todo más allá de la simple suma de sus partes componentes, ocultando las complejidades inherentes al gran número de interacciones entre estos.

Capítulo 4 Tendencias hacia los Sistemas Distribuidos Administrables

Ciclo de vida de los sistemas de software.

Los sistemas de software no escapan a la dinámica con la que el mundo actual evoluciona, constantemente deben ser adaptados o cambiados en función de los nuevos requerimientos a los que deben dar respuesta. Esta vorágine de cambio y adaptación constante hace parecer a los sistemas de software modernos un ente vivo. Un ente vivo que sufre un proceso evolutivo como cualquier otro: nace, cambia, se adapta y se degrada.

El fenómeno de la evolución del software comenzó a ser estudiado a principios de la década de 1970. A partir de esta década y en años posteriores se formalizaron algunas bases teóricas al respecto de los procesos que rigen la evolución de los sistemas de software y aplicaciones. Un conjunto de ocho leyes, conocidas como las Leyes de Lehman dictan, por así decirlo, los procesos que determinan el comportamiento del software durante su evolución. El trabajo de M.M Lehman conjunto con J.F Ramil "*An approach to a theory of software evolution*"[42] contiene las bases de la teoría de evolución de los sistemas de software. Las leyes del Cambio Continuo, la Complejidad Incrementada, la Auto Regulación, la Conservación de la Estabilidad Organizacional, la Conservación de la Familiaridad, el Crecimiento Continuo, la Calidad Disminuida y la Retroalimentación del Sistema constituyen la fenoménica de la evolución del software.

Las leyes, -empíricamente enunciadas- obedecen inicialmente a la premisa de que los sistemas de software se encuentran cambiando constantemente. La introducción de estos cambios se debe en gran medida a que los ambientes donde se encuentran ejecutándose son dinámicos y necesitan que el software se mantenga a la par con sus requerimientos. Es de esperar que los cambios introducidos afecten parámetros como la calidad y la complejidad. La calidad es afectada porque con el cambio se introducen errores que no estaban presentes o se afectan funciones que no presentaban fallos en

[42] (Lehman, y otros, 1985)

versiones anteriores y la complejidad de los sistemas aumenta en la misma medida en que nuevas funcionalidades son requeridas para soportar la automatización de nuevos procesos.

Por tanto, los sistemas crecen de forma continua en el tiempo, cambian su apariencia -perdiendo en muchos casos la familiaridad con el usuario-, su calidad disminuye, se vuelven complejos. Esto hace que lleguen a un punto donde el mantenimiento y la operación se hacen costosos –no solo desde el punto de vista económico- y los sistemas deben ser puestos fuera de operación y reemplazados por sistemas que cumplan las mismas funciones de manera óptima.

Consecuentemente, si existe un proceso de evolución de los sistemas de software, se puede decir que existe un ciclo de vida de los sistemas software.

Continuando con la analogía con las Ciencias Biológicas, donde los seres vivos presentan un ciclo de vida, los sistemas de software poseen un ciclo de vida que comienzan en la idea, la concepción del propósito para el que son creados y termina con la puesta fuera de operación. La definición de ciclo de vida de los sistemas de software se adopta desde varias perspectivas que varían en relación con la implicación que tenga una organización con el sistema. Para David Chappell, el ciclo de vida de una aplicación "*incluye el tiempo durante el cual una organización emplea dinero en este activo, desde la idea inicial hasta el final de la vida de la aplicación*"[43].

Al igual que la vida de los seres vivos, el ciclo de vida de las soluciones de software es dividido en etapas, etapas demarcadas por hechos, momentos, hitos significativos dentro del proceso de evolución. La división en etapas del ciclo de vida del software es tratada con mayor o menor granularidad. Chappell propone una división en tres etapas: gobierno, desarrollo y operaciones, delimitadas por tres hitos o momentos: la idea, el despliegue y el fin de vida [44]. Mientras, la Oficina de Marcas y Patentes de los Estados Unidos en su Manual de Administración del Ciclo de Vida[45] propone una división mucho más

[43] (Chappell, David, 2008)
[44] Ibídem
[45] (United State Patents and Trademarks Office)

granular contando con seis etapas: iniciación, concepción, análisis y diseño, desarrollo, despliegue y operaciones.

Durante muchos años se han desarrollado infinidad de metodologías que cubren el ciclo de desarrollo con el objetivo de optimizar las actividades y el uso de los recursos humanos. Es entendible que tanto en la industria como en el sector académico surgieran estrategias que organizaran en roles y actividades, el personal y las tareas que de alguna manera se encontraban involucrados en estas etapas. La tendencia dentro de las grandes compañías desarrolladoras de sistemas indica que las actividades de despliegue y operaciones requieren mayor atención, debido -en gran medida- al sinnúmero de componentes de hardware y software heterogéneos que interactúan hoy en las soluciones más comunes.

Despliegue y operación de soluciones de software: tendencias actuales.

La propiedad de las soluciones de software, y de Tecnologías de la Información de manera general, comienza para una organización a partir del momento en que se empieza a desplegar una solución tecnológica. La propiedad de tecnologías es hoy una de las preocupaciones fundamentales en cuanto a los gastos incurridos por concepto de la automatización de los procesos críticos en las organizaciones. Sería ingenuo pensar que el único costo en el proceso de adquisición y explotación de una solución de software es el precio de su compra. El despliegue y mantenimiento posterior implican cuantiosas inversiones para cualquier organización, de aquí que exista un esfuerzo generalizado por organizar y optimizar las actividades que se cumplen durante el despliegue y operación de las soluciones.

Un hito en las investigaciones sobre el costo de adquirir y mantener soluciones tecnológicas lo marcó la presentación en 1987 -por Hill Kirwin, de Gartner Group Inc- del Modelo del Costo Total de Propiedad. Según la definición de Interpose Inc es *"un modelo que ayuda a las empresas a analizar todos los costos y beneficios relacionados con la adquisición, desarrollo y uso de componentes de Tecnologías de la Información a lo largo de su ciclo de vida".* Este modelo fue originalmente creado para los equipos de escritorios pero a lo largo del tiempo se ha ido extendiendo a las redes LAN, los sistemas cliente/servidor, la computación distribuida, telecomunicaciones, centros de procesamiento de datos y sistemas portátiles.

El modelo ayuda a los ejecutivos a determinar los costos directos e indirectos en la compra de cualquier activo pero fundamentalmente el software y el equipamiento. El alcance del modelo del Costo Total de Propiedad va más allá de la mera consideración del precio de compra del producto. Dentro del modelo se incluyen además los costos por uso y mantenimiento de los sistemas y equipos, incluyendo costos de implementación, capacitación de los usuarios, costos asociados a fallas o mantenimientos planeados o no, sucesos que afectan el rendimiento y otros muchos factores que inciden en la factibilidad económica de una tecnología para las empresas que las adquieren. Básicamente, el Modelo del Costo Total de Propiedad es una herramienta destinada a apoyar el proceso de toma de decisiones que involucran la propiedad y operación de sistemas de cómputo.

Las tendencias orientadas a la minimización de los costos se pueden dividir fundamentalmente en dos vertientes principales: los propietarios de tecnologías y el esfuerzo por estructurar eficientemente los procesos operacionales en los entornos corporativos y los desarrolladores de tecnologías y la intensión de facilitar la administración de sus soluciones en dichos entornos.

Soluciones centradas en el propietario.

El uso de distintas metodologías como herramientas para la organización de las personas y los procesos involucrados en el despliegue y operación de tecnologías en general, y de soluciones de software más específicamente, es una de las soluciones que mayor aceptación han tenido dentro de los propietarios de soluciones tecnológicas, con la estructuración de las actividades a través de guías y marcos de trabajo que indican como tratar de evitar y, en caso de ocurrencia, resolver problemas que acontecen en un entorno corporativo donde existe desplegada y en operación una solución de software o cualquier otro tipo de tecnología.

Entre estas metodologías resaltan la Biblioteca de Infraestructura de Tecnologías de la Información (ITIL)[46] , Los Objetivos de Control para la Información y la Tecnología relacionada (COBIT)[47] y el Marco de

[46] Siglas en inglés de: Information Technology Infrastructure Library.
[47] Siglas en inglés de: Control Objectives for Information and related Technology.

Trabajo de Microsoft para Operaciones (MOF)[48] como las de mayor aceptación dentro de las metodologías encargadas de organizar entornos de operaciones de soluciones tecnológicas y de software.

Biblioteca de Infraestructura de Tecnologías de la Información (ITIL).

A finales de la década de los ochenta la Agencia Central de Cómputo y Telecomunicaciones del gobierno del Reino Unido publicó la primera versión de la Biblioteca de Infraestructura de Tecnologías de la Información (ITIL). La biblioteca consistía en 42 libros que describían las mejores prácticas referentes a la administración de las Tecnologías de la Información. Estas recomendaciones fueron creadas sobre la base de la experiencia acumulada en la administración de grandes centros de datos. En ese momento los ordenadores personales no eran algo común, comenzaban a ser introducidos en las oficinas para ese entonces.

El objetivo fundamental de la biblioteca es organizar y orquestar una infraestructura de Tecnologías de la Información compleja para alcanzar los mejores resultados en los procesos de negocio. Es importante señalar que ITIL no es exactamente un estándar, entonces no existe algo como que un proceso o compañía sea “Certificado ITIL”.

La biblioteca ha ido evolucionando en el tiempo y ha sido adaptada a las necesidades y situaciones actuales. En el año 2000 fue lanzada la versión 2, ITIL v2. Para esta versión los autores advirtieron el rápido crecimiento de la comunidad internacional de Administración de Servicios de Tecnología de la Información. En esta revisión varios procesos relacionados fueron acoplados, reduciendo así el número de libros de 42 a ocho. La biblioteca a partir de esta versión se distingue por el énfasis en la estrecha relación necesaria entre cliente y proveedor, adoptando un carácter más orientado a los servicios. Además fue incluida como novedad una guía “Planificando la implementación de Administración de Servicios” que explicaba por primera vez cómo aplicar ITIL. Con la ayuda de ITIL v2 las organizaciones implementan de manera más eficiente algunos estándares de calidad como la norma ISO/IEC 9000.

Hoy ITIL cuenta con la versión 3, ITIL v3 -lanzada en 2007- que incluye el concepto de ciclo de vida donde el proceso de provisión de servicios es considerado un proceso continuo comenzando en el momento

[48] Siglas en inglés de: Microsoft Operations Framework.

donde el servicio es puesto en funcionamiento hasta que es sacado de explotación. Una vez más, el número de libros fue reducido hasta que quedaron solamente cinco, organizando los procesos por las fases del ciclo de vida de los servicios. Actualmente la biblioteca es propiedad de la Oficina de Comercio del Reino Unido y genera importantes ingresos para el Tesoro del Reino Unido.

Los Objetivos de Control para la Información y la Tecnología relacionada (COBIT)

Los Objetivos de Control para la Información y la Tecnología relacionada (COBIT) son un conjunto de buenas prácticas propuestas por el IT Governance Institute (ITGI), una organización establecida en 1998 con el objetivo de evolucionar el pensamiento y los estándares internacionales en relación con la dirección y el control de tecnologías de la información en una empresa. COBIT presenta las actividades relacionadas con el control de estas tecnologías en una estructura lógica y manejable a través de un marco de trabajo compuesto por dominios y procesos. Estas buenas prácticas exponen el criterio de los expertos y están enfocadas en el control por encima de la ejecución y ayudarán a optimizar las inversiones de TI, la calidad de los servicios prestados y una forma de medir el rendimiento en los entornos empresariales.

Es COBIT el intento por orientar hacia la coincidencia las metas del negocio con las metas de Tecnologías de la Información para obtener el mayor beneficio posible utilizando métricas y modelos de madurez para conmensurar los logros alcanzados en esa dirección. COBIT es una metodología con una marcada orientación hacia los procesos a través de un modelo de procesos que divide las actividades de TI en 34 procesos en función de las responsabilidades de planear, construir, ejecutar y monitorear desempeñadas por el personal de operación de las tecnologías dentro del ámbito de la empresa. Además, COBIT constituye un marco de referencia así como un juego de herramientas que permite a la gerencia de las empresas y organizaciones que lo implementan comprender y administrar los riesgos relacionados con la adopción de las Tecnologías de la Información.

Los beneficios de implementar COBIT como marco de referencia de gobierno sobre la TI incluyen:

- Mejor alineación, con base en su enfoque de negocios.
- Una visión, entendible para la gerencia, de lo que hace TI.
- Propiedad y responsabilidades claras, con base en su orientación a procesos.

- Aceptación general de terceros y reguladores.
- Entendimiento compartido entre todos los participantes, con base en un lenguaje común.

La implementación está soportada por un número de productos ISACA/ITGI incluyendo herramientas en línea, guías de implementación, guías de referencia y material educacional[49].

Marco de Trabajo de Microsoft para Operaciones (MOF)

Microsoft Operations Framework consiste en un conjunto de buenas prácticas, principios y actividades que proveen una guía general para alcanzar la confiabilidad en soluciones y servicios de Tencologías de la Información. Esta guía, basada en un cuestionario, permite a las empresas determinar que necesidades tienen y que actividades mantendrán el área tecnológica dentro de la propia organización ejecutándose de un una manera eficiente y efectiva en un futuro.

La guía de Microsoft Operations Framework recoge todas las actividades y procesos involucrados en la administración de un servicio: su concepción, desarrollo, operación, mantenimiento y finalmente su retiro. Organiza además estas actividades y procesos en Funciones de Administración de Servicios (SMF)[50] las cuales se agrupan en fases que simulan el ciclo de vida de los servicios de Tecnologías de la Información. Cada función está relacionada con una fase y contiene un conjunto único de metas y resultados que están relacionados con los objetivos de cada fase.

La meta fundamental de Microsoft Operations Framework es proveer una guía a las organizaciones que poseen tecnologías de la información para ayudarlos a crear, operar y darle soporte servicios de Tecnologías de la Información mientras se asegura que las inversiones realizadas entregan el valor esperado de negocio con un nivel de riesgo aceptable. El propósito de MOF es crear un ambiente donde los procesos de negocio y de administración de Tecnologías de la Información puedan trabajar en conjunción hacia la madurez en los procesos y actividades operacionales mediante la promoción de un acercamiento lógico a la toma de decisiones y comunicación fluyente en la planificación, despliegue y soporte de servicios de Tecnologías de la Información.

[49] La información más reciente sobre estos productos se puede consultar en http://*www.isaca.org/cobit*
[50] Siglas en inglés de: Service Management Functions.

Soluciones centradas en las empresas desarrolladoras.

Además de las estrategias propuestas por las distintas metodologías adoptadas para las mejoras orientadas a la optimización de procesos, las empresas desarrolladoras de tecnologías han comprendido la necesidad de facilitar la administración de sus soluciones en los entornos corporativos. La tendencia es a equipar las soluciones de software, fundamentalmente las soluciones distribuidas, con herramientas capaces de controlar el correcto funcionamiento de los sistemas y sus diferentes componentes mediante la automatización de tareas que requieren el esfuerzo del personal dedicado al mantenimiento de las tecnologías dentro de las organizaciones.

Empresas dedicadas a la producción de sistemas de características tan disímiles como sistemas operativos, sistemas de gestión de base de datos, e incluso sistemas antivirus han incluido módulos o componentes y en algunos casos grupos de herramientas para la administración y control de dichas soluciones. Para el análisis de la tendencia se tuvieron en cuenta un grupo de casos de estudio referidos al equipamiento de soluciones con herramientas que automatizan las actividades y a soluciones genéricas que de manera independientes realizan dichas tareas.

Caso de Estudio: IBM: CID para OS/2.

CID para OS/2 es una de las primeras aproximaciones a la automatización de la instalación, configuración y mantenimiento de sistemas. En octubre de 1992, IBM anunció el equipamiento de su sistema operativo OS/2 con un conjunto de capacidades conocidas como el método CID[51] y que muchos otros de sus productos estarían dotados de las capacidades de CID. Las investigaciones de IBM ya en ese entonces habían arrojado como resultado que, debido al crecimiento de la cantidad de ordenadores y la complejidad y tamaño de los sistemas modernos, el proceso de instalación y configuración –a través de disquetes y CD-ROM en aquel entonces- era tedioso y complicado y necesitaba de la presencia física en el lugar de los encargados de administrar los sistemas.

Detectaron además un grupo de dificultades que traía el método de instalación manual.[52]

[51] Siglas en inglés de: Configuration, Installation and Distribution.
[52] (IBM Corporation, International Technical Support Organization, 1996) pp.46.

- La intervención humana era requerida para personalizar el producto mediante el suministro de de la información de configuración a través de la interfaz de diálogo del programa de instalación.
- Dado que la mayoría de los productos eran suministrados en un gran número de disquetes, en muchos casos se requería el cambio de medios durante el proceso de instalación.
- La información acerca del progreso del proceso de instalación así como si la finalización del mismo era completada satisfactoriamente tenía que ser verificada para garantizar un sistema completamente funcional.
- Algunos sistemas requerían el reinicio de la estación de trabajo para la activación de los cambios de configuración.

Los objetivos que se perseguían con CID eran eliminar estas características no deseadas en el método tradicional de despliegue del software. CID estaba guiado a:

1. Eliminar la intervención humana en la estación de trabajo destino mientras se prepara y ejecutan los procesos de configuración, instalación, migración y/o mantenimiento que son necesarios para operar la estación de trabajo.
2. Permitir la ejecución de código en la estación de trabajo destino para realizar todas las tareas de instalación y configuración incluyendo la integración de personalizaciones previas.
3. Proveer la capacidad de centralizar la intervención humana en un administrador en un sitio de preparación central.

Caso de Estudio: Kaspersky Lab: Kaspersky Administration Kit.

La sociedad Kaspersky Lab fue creada en 1997 y es hoy uno de los principales fabricantes de productos de software de seguridad y protección de datos: antivirus, antispam y sistemas anti efracción. Para la administración de su suite antivirus Kaspersky Anti-virus, Kaspersky Lab implementó una solución conocida como Kaspersky Administration Kit. La misma fue diseñada con el propósito de facilitar el control centralizado de las principales tareas administrativas. Es una herramienta dirigida al administrador que permite la distribuir y eliminar de forma remota las aplicaciones de Kaspersky Lab en y desde los equipos en una red, facilitando su despliegue en ambientes distribuidos. Además se encarga de la administración

de las claves de licencias de las aplicaciones, supervisar la correspondencia entre las licencias y el número de usuarios y controlar la fecha caducidad de las mismas.

En cuanto a las características de operaciones, se encarga de administrar de manera remota las aplicaciones desde una ubicación centralizada, funcionalidad que es vital en organizaciones con varias sedes remotas. Permite actualizar automáticamente la base de antivirus por la que se pueden actualizar todos los motores antivirus que estén instalados en la red y evita la necesidad de distribuir manualmente dichas actualizaciones en cada uno de los nodos.

Equipada con un mecanismo de recopilación de informes, Kaspersky Administration Kit como solución permite generar estadísticas al respecto de los errores, problemas de configuración y además notifica de la ocurrencia de la ocurrencia de eventos específicos sobre el funcionamiento de las aplicaciones utilizando vías como el correo electrónico. Además permite el disfrute de un servicio de soporte para los usuarios utilizando la información que es recopilada acerca de los errores.

La utilización de una herramienta de estas características permite acelerar las tareas relacionadas con la solución antivirus. Mediante la automatización del despliegue, la actualización y la recopilación de información de soporte, además de las funciones de administración, facilita la utilización de la suite de Kaspersky Anti-virus y la hace hoy uno de los productos más competitivos y exitosos dentro del mercado.

Caso de Estudio: Microsoft Corporation: Microsoft Deployment Toolkit 2008 y Microsoft System Center.

Microsoft Deployment Toolkit (MDT) 2008 es una suite de herramientas brindada por Microsoft para la correcta planificación, construcción, prueba y despliegue de los sistemas operativos Microsoft Windows y la suite de herramientas de oficinas Microsoft Office 2007 y Microsoft Office 2003. Es un conjunto de buenas prácticas y herramientas con las que el personal de administración puede facilitar la operación de un entorno distribuido con tecnologías Microsoft.

Promueve además la utilización de un grupo de tecnologías destinadas a lograr estas metas: Application Compatibility Toolkit (ACT) versión 5.0 -para la recolección de inventario de aplicaciones y para la prueba y mitigación de los problemas de compatibilidad de aplicaciones-, Windows User State Migration

Tool(USMT) -para soportar la migración de las configuraciones y los datos de los usuarios de sistemas operativos Microsoft Windows antes de una actualización-, Windows Automated Installation Kit (Windows AIK) –para configurar la instalación desatendida-, Windows Deployment Services -para la ejecución de Windows Preinstallation Environment-, y Windows Preinstallation Environment versión 2.0 para iniciar en ordenadores sin sistemas operativos instalados.

Con MDT 2008 es posible crear un inventario de hardware y software necesario para la planificación del despliegue, probar las aplicaciones para la compatibilidad con los sistemas operativos Windows y mitigar los problemas de compatibilidad descubiertos durante el proceso. Además permite personalizar, empaquetar y desplegar aplicaciones, automatizar la creación de imágenes y su despliegue. Durante el proceso de despliegue es posible migrar los documentos de los usuarios y sus configuraciones hacia sus nuevos ordenadores y manipular el proceso con poco o ningún nivel de atención por parte del personal encargado de la tarea.

Para la administración y control posterior de los ambientes operativos con tecnologías Microsoft, la compañía ha generado una familia de productos conocida como Microsoft System Center. Los miembros de esta familia se caracterizan por utilizar un enfoque basado en modelos estándar, permitiendo la descripción de los sistemas que están siendo administrados. Para organizaciones con fuertes inversiones en la plataforma Microsoft Windows, la colección System Center puede proveer los cimientos correctos para administrar su entorno corporativo.

Entre las herramientas que provee la familia se encuentran:

- **System Center Operations Manager 2007**: para el monitoreo y la administración del hardware y el software en un ambiente distribuido moderno.
- **System Center Configuration Manager 2007**: provee herramientas para la automatización de la instalación de software y la administración de la configuración de sistemas.
- **System Center "Service Desk"**: proporciona la implementación de servicios básicos de administración como la gestión de incidentes, la administración de cambios y problemas entre otros.

- **System Center Data Protection Manager 2006**: tal y como sugiere su nombre, facilita el respaldo de datos para servidores de tecnología Microsoft.
- **System Center Essentials 2007**: funge como herramienta que unifica las dos más importantes funciones de administración: el monitoreo de sistemas distribuidos y la instalación automatizada de software.

Los servicios proporcionados por estas y el resto de las herramientas de la familia de soluciones System Center son la expresión concreta de las metas establecidas por Microsoft en la Iniciativa de Sistemas Dinámicos (DSI)[53] y el esfuerzo de la empresa y sus asociados de negocios para mejorar la capacidad de las soluciones de software de cumplir los requerimientos del negocio manteniendo bajos costos en la administración de sus recursos informáticos.

Iniciativas de Diseño y Desarrollo de Soluciones Administrables

Las tendencias de equipamiento de soluciones de software distribuidas con componentes de administración han cambiado la forma en que los desarrolladores de sistemas entienden el conocimiento acerca de la infraestructura donde se van a desplegar y operar. En este sentido, en la industria de desarrollo de software se han comenzado iniciativas con el propósito de incorporar este conocimiento al proceso de concepción e implementación de sistemas. Las mismas han ido encaminadas a una concepción integral y estándar acerca de que conocimiento sobre la infraestructura ha de ser tenido en cuenta y sobre la forma en que este conocimiento es capturado.

Entre las iniciativas globales guiadas por este propósito se encuentra la creación de la organización Distributed Management Task Force (DMTF). Desde 1992, el principal objetivo de DMTF ha sido promover la interoperabilidad entre los diferentes productos dedicados al despliegue y operación de empresas de la industria del software, logrando estándares y especificaciones independientes de plataforma o tecnología.

DMTF subdivide su trabajo en estándares e iniciativas. Hasta hoy ha logrado homologar seis especificaciones: el Modelo Común de Información (CIM)[54], la Administración Empresarial basada en Web

[53] Siglas en inglés de: Dynamic Systems Initiative.
[54] Siglas en inglés de: Common Information Model.

(WBEM)[55], las especificaciones de BIOS para la Administración del Sistema (SMBIOS)[56], el Formato Estándar para Alertas (ASF)[57], el Modelo Común de Diagnóstico (CDM)[58] y la Interfaz de Administración de Escritorio (DMI)[59]. Además, DMTF patrocina un grupo de iniciativas, propias y de sus asociados, para acelerar el proceso de adopción de los estándares propuestos y con el objetivo de ayudar en el desarrollo de soluciones interoperables basadas en estas especificaciones. Hoy DMTF mantiene las iniciativas de: Modelo Común de Diagnóstico (CDM-I)[60], Arquitectura de Escritorio y Móvil para Hardware de Sistema (DASH-I)[61], Arquitectura de Administración de Sistemas para Hardware de Servidores (SMASH-I)[62] y la de Administración de Virtualización (VMAN-I)[63]. También DMTF apoya iniciativas de otras organizaciones como el caso de la Iniciativa de Administración de Almacenamiento (SMI-S)[64] propuesta por la Storage Networking Industry Association.

Las iniciativas han sido respaldadas por más de 200 compañías de todo el mundo y posee hoy un número de colaboradores superior a los 3000. Entre ellos algunos ya conocidos por ser promotores como los casos de IBM, Microsoft, Novell, Oracle, Dell, Fujitsu, Intel, entre otras compañías. También más de 50 instituciones académicas han ido participando a lo largo de los años.

Las especificaciones y modelos propuestos por DMTF han sido adoptados por algunas empresas en las iniciativas emprendidas para la automatización de las actividades de despliegue y operación de soluciones de software. Estas iniciativas se han caracterizado por sugerir la incorporación de los modelos y estándares desde fases tempranas del desarrollo y que persistan a lo largo del ciclo de vida de la solución.

[55] Siglas en inglés de: Web-Based Enterprise Management.
[56] Siglas en inglés de: System Management Basic Input/Output System.
[57] Siglas en ingles de: Alert Standard Format.
[58] Siglas en ingles de: Common Diagnostic Model.
[59] Siglas en inglés de: Desktop Management Interface. Este estándar fue declarado como terminado desde el 31 de marzo de 2005.
[60] Siglas en ingles de: Common Diagnostic Model Initiative
[61] Siglas en inglés de: Desktop and mobile Architecture for System Hardware Initiative.
[62] Siglas en inglés de: System Management Architecture for Server Hardware Initiative
[63] Siglas en inglés de: Virtualization Management Initiative.
[64] Siglas en inglés de: Storage Management Initiative Specification.

Persiguiendo esta meta se han vislumbrado dos iniciativas fundamentales en la nueva forma de concebir aplicaciones -fundamentalmente distribuidas-, la Iniciativa de Sistemas Dinámicos de Microsoft Corporation y la Iniciativa de Informática Autónoma de IBM Corporation.

Microsoft Corporation y la Iniciativa de Sistemas Dinámicos (DSI)[65]

La Iniciativa de Sistemas Dinámicos es un esfuerzo de Microsoft Corporation y sus asociados por la mejora de la plataforma Microsoft Windows en cuanto a características de administración mediante el ofrecimiento de un conjunto coordinado de soluciones que simplifiquen y automaticen notablemente el modo en que las empresas diseñan, implementan y utilizan sistemas distribuidos. Microsoft ha estado realizando fuertes inversiones en la investigación y el desarrollo de software, y también cuenta con inversiones de asociados para ofrecer soluciones completas integradas a través de herramientas de desarrollo de aplicaciones, sistemas operativos, aplicaciones, hardware y herramientas de administración, para lograr costos reducidos, mayor confiabilidad y mejor capacidad de respuesta a lo largo del ciclo de vida de las tecnologías de la información.

De acuerdo a las investigaciones de Microsoft el principal problema relacionado con la dificultad de administración de muchas soluciones distribuidas es que son en primera instancia diseñadas para cumplir con los requerimientos funcionales del negocio. La arquitectura de la infraestructura donde se desplegará la solución nunca se ha valorado con la misma rigurosidad con la que se evalúa la arquitectura de la solución y en muchos casos no se considera en el momento de diseño. De aquí que la propuesta de Microsoft, dentro de la Iniciativa de Sistemas Dinámicos, para el desarrollo de soluciones distribuidas es la consideración de las características administrables de la solución dentro de la arquitectura de la propia solución.

Materializar esta propuesta de integración vertical de las características de administración en las diferentes herramientas requirió de un proceso de innovación por parte de Microsoft en cuanto a encontrar una forma de representar genéricamente el conocimiento referido a las infraestructuras de Tecnologías de la Información y una forma genérica de comunicación con los ordenadores en la red. Con este propósito

[65] Siglas en inglés de: Dynamic Systems Initiative.

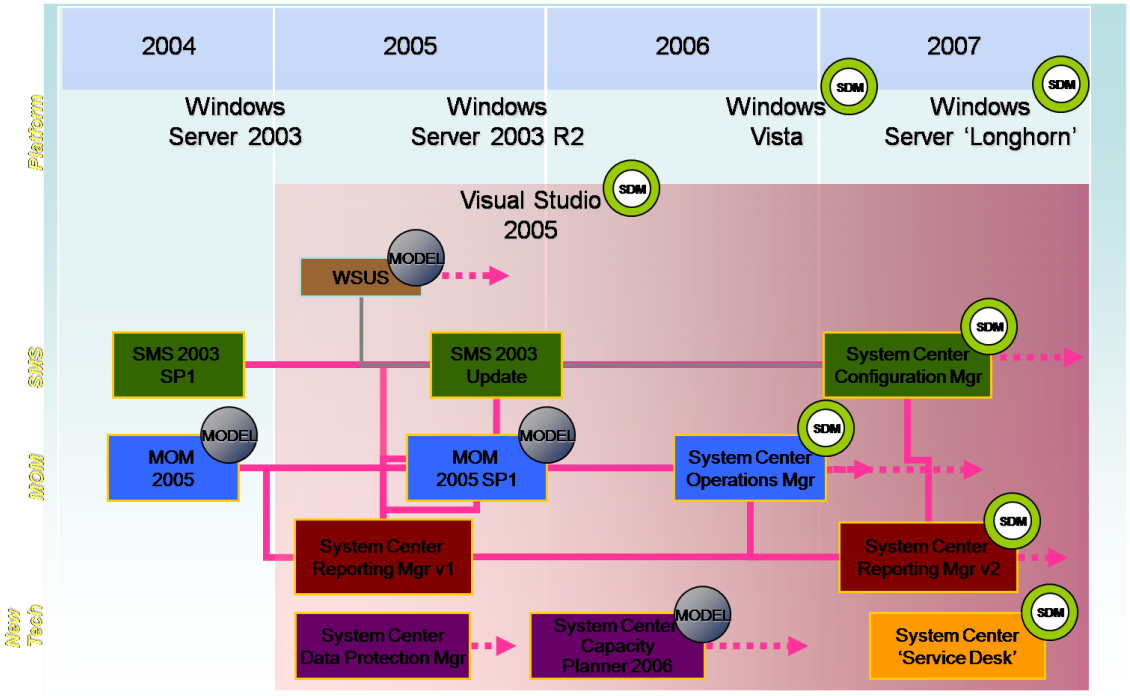
desarrolló el Modelo de Definición de Sistemas (SDM)[66] y adoptó el estándar WS-Management propuesto por DMTF, considerados ambos las piedras angulares de la estrategia.

Con la introducción del Modelo de Definición de Sistemas, Microsoft sienta el precedente de la administración basada en modelos, donde el conocimiento referido a la solución a lo largo de su ciclo de vida es capturado en modelos de conocimiento. Según Enrique Lopiz, Gerente de Soluciones de Microsoft System Center, "*La idea subyacente es simple pero extremadamente potente. Hasta ahora el conocimiento de cómo gestionar un sistema informático se plasma en un documento de texto: los requisitos de despliegue, este despliegue, la configuración, la seguridad, las políticas aplicables, la operación etc., son documentos de texto que alguien tiene que "traducir" al lenguaje de la plataforma que se utiliza para gestionar un sistema. Esto nos lleva a una "Babel" de formas de plasmar cómo se debe gestionar y a problemas de interoperabilidad: nadie habla el mismo lenguaje y, básicamente, no existe una forma estándar de describir cómo se debe gestionar el ciclo de vida de un servicio informático que una aplicación de gestión pueda entender directamente*"[67].

SDM es un lenguaje basado en XML y es además una plataforma de modelado a través de la cual Microsoft ha pretendido que se creen planos esquemáticos para la administración efectiva de sistemas distribuidos, brindando un entendimiento común en los procesos de desarrollo, despliegue y operaciones. Las capacidades de crear modelos y de interpretarlos se han ido agregando a las herramientas tanto de desarrollo como de administración a lo largo y ancho de la plataforma Microsoft Windows. Las herramientas de la familia de productos Microsoft System Center, que se mencionaron anteriormente, son el ejemplo del impacto que ha tenido la iniciativa dentro de la compañía. La Figura 4.1 muestra como gradualmente Microsoft ha ido incorporando el Modelo de Definición de Sistemas en la plataforma.

[66] Siglas en inglés de: System Definition Model.
[67] (Computerworld Magazine, 2007)

Figura 4.1 SDM en la plataforma Microsoft Windows



La otra piedra angular dentro de la iniciativa es la comunicación genérica entre los dispositivos en una infraestructura tecnológica. Para alcanzar esta meta la empresa consideró la utilización de un estándar de administración propuesto por DMTF, el estándar de Administración basada en Servicios Web (WS-Man)[68]. Este especifica un protocolo basado en SOAP[69] para la administración de sistemas tales como ordenadores, servidores, dispositivos, servicios web y otras entidades administrables. A diferencia de las tecnologías anteriores promovidas para la administración, como el Instrumental de Administración de Windows (WMI) -que era basado en la tecnología COM+ propiedad de exclusiva de la compañía-, la utilización de un estándar internacional demuestra el compromiso adquirido con la interoperabilidad en los ambientes heterogéneos donde coexisten soluciones tanto de software como de hardware de diversas

[68] Siglas en inglés de: Web Services for Management.
[69] Siglas en inglés de: Simple Object Access Protocol.

plataformas, posición alejada de algunas otras que tomaba Microsoft donde trataba de imponer su estándares por encima de los estándares generalmente aceptados.

La Iniciativa de Sistemas Dinámicos es una tarea a largo plazo y de largo alcance. La visión de Microsoft al respecto de los sistemas dinámicos en el futuro son los sistemas auto-administrados, pero de acuerdo a su estrategia, para que las soluciones sean auto-administradas tanto estas como la plataforma que las soportan deben ser concebidas pensando en la administración. DSI es entonces el primer paso encaminado a la tarea de eliminar paulatinamente la permanente intervención humana en el control de la tecnología.

IBM y la Iniciativa de Informática Autónoma

A mediados de octubre de 2001, IBM publicó un manifiesto[70] donde apuntaba que el principal obstáculo para el progreso de la industria de las Tecnologías de la Información era la alarmante crisis de la complejidad del software debido a que había aumentado el número de elementos interconectados, así como la diversidad de los mismos. Siendo así, la complejidad volvería prácticamente imposible la instalación, configuración, optimización, mantenimiento y mezcla de los componentes de las soluciones de software.

Para atacar este problema Paul Horn -vicepresidente de investigaciones de IBM- propone a la Academia Nacional de Ingenieros en la Universidad de Harvard en 2001 utilizar la concepción de autonomía, término derivado de la biología del cuerpo humano para, haciendo una analogía, auto-controlar las tecnologías con tecnologías. Se acuña así el término de *informática autónoma.*

Existen ocho características que describen a un sistema informático autónomo:

1. *Identidad del Sistema:* tiene conocimiento de sus componentes, estado actual, funciones y sus interacciones con el ambiente.

2. *Autoconfiguración y reconfiguración:* tiene la habilidad de automáticamente realizar ajustes dinámicos en su funcionamiento en un ambiente variable e impredecible.

[70] (IBM Corporation, 2001)

3. *Constante auto-optimización:* mediante la monitorización de sus partes constituyentes adapta su comportamiento para alcanzar determinadas metas de rendimiento.

4. *Auto-corrección:* puede descubrir las causas de las fallas y entonces encontrar formas alternativas de utilizar los recursos o reconfigurar el sistema para mantenerlo funcionando.

5. *Auto-protección:* detecta, identifica y se protege a sí mismo contra varios tipos de ataques para mantener la seguridad e integridad de todo el sistema.

6. *Utiliza la auto-adaptación:* es capaz de encontrar la mejor forma de interactuar con sistemas vecinos y describirse a sí mismo a estos y descubrir estos sistemas en el ambiente.

7. *Solución no propietaria abierta basada en estándares:* los estándares proveen las bases para la interoperabilidad a través de los límites del sistema.

8. *Complejidad oculta:* automatiza las tareas de infraestructura de Tecnologías de la Información y releva a los usuarios de las actividades administrativas.

La esencia de los sistemas informáticos autónomos es la auto-administración y el intento de liberar a los administradores de sistemas de los detalles de la operación y mantenimiento de los mismos y proveer a los usuarios de ordenadores que los sistemas se ejecuten al máximo de sus capacidades todo el tiempo. La auto-administración es dividida por IBM en cuatro aspectos fundamentales: auto-configuración, auto-optimización, auto-corrección y la auto-protección. En la Tabla 4.1 se especifican como ocurren hoy estos aspectos sin la consideración de la autonomía de los sistemas y como ocurrirían en sistemas que implementen la informática autonómica.

Tabla 4.1 Aspectos de la auto-administración

Conceptos	Sistemas actuales	Sistemas autónomos
Auto-configuración	Los centros de datos corporativos poseen múltiples vendedores y plataformas. Instalar, configurar e integrar sistemas consume tiempo y es propenso al error.	La configuración automática de componentes y sistemas siguen políticas de alto nivel. El resto de los sistemas se ajustan automáticamente y sin fallos.
Auto-optimización	Los sistemas poseen cientos de parámetros de rendimiento no lineales y fijados manualmente, y sus números se incrementan con cada liberación.	Componentes y sistemas continuamente buscan oportunidades de mejorar su propio rendimiento y eficiencia.

Auto-corrección	La determinación de problemas en sistemas largos y complejos puede tomar a un equipo de programadores semanas.	Los sistemas automáticamente detectan, diagnostican, y reparan los problemas de hardware y de software localizados.
Auto-protección	La detección y recuperación de ataques y fallas en cascada es manual.	Los sistemas automáticamente se defienden contra ataques maliciosos o fallas en cascadas. Utilizan la alerta temprana para anticipar y prever fallas a lo largo del sistemas

La incorporación de las capacidades de autonomía en un ambiente computacional es un proceso evolutivo logrado a través de la tecnología, pero esto es en última instancia implementado por cada empresa mediante la adopción de esas tecnologías, procesos de soporte y conocimientos. A lo largo de la evolución, la industria continuará entregando herramientas de auto-administración para mejorar la productividad de los profesionales IT.

Para entender el nivel de complejidad de las herramientas y capacidades que son, y serán, entregadas por la industria, IBM considera los siguientes cinco niveles de madurez autonómica:

- *Nivel Básico*: los profesionales IT administran cada elemento de la infraestructura independientemente y lo configuran, lo monitorean y ocasionalmente lo reemplazan.
- *Nivel Administrado*: las tecnologías de administración de sistemas pueden ser utilizadas para recolectar información de sistemas dispersos en unas pocas consolas, ayudando a reducir el tiempo que le toma al administrador recolectar y sintetizar esa información, a medida que el entorno IT se vuelve cada vez más complejo.
- *Nivel Predecible:* nuevas tecnologías se introducen para proveer correlación a lo largo de varios elementos de la infraestructura. Esos elementos pueden comenzar a reconocer patrones, predecir la configuración óptima y proveer aviso sobre qué curso de acción debería tomar el administrador.
- *Nivel Adaptativo:* el contexto IT puede automáticamente tomar acciones basado en la información disponible y el conocimiento de qué está sucediendo en el ambiente. A medida que estas tecnologías mejoran y que la gente se va sintiendo más cómoda con los avisos y el poder de predicción de estos sistemas, las tecnologías pueden progresar hacia el nivel adaptativo.

- *Nivel Autónomo:* las políticas y los objetivos de negocio gobiernan la operatoria de la infraestructura IT. Los usuarios interactúan con las herramientas de tecnología autónoma para monitorear los procesos de negocio, alterar los objetivos, o ambos.

Para la adopción paulatina de la Iniciativa de Informática Autónoma, IBM presenta en la industria el Kit de Herramientas de Informática Autónoma. El Kit de Herramientas de Informática Autonómica es una colección de tecnologías, herramientas, ejemplos, escenarios y documentación que es diseñada para usuarios que desean aprender, adaptar y desarrollar capacidades autonómicas en sus productos y sistemas. El kit se divide en cuatro áreas fundamentales de componentes: tecnologías, herramientas, escenarios e información y documentación.

Las tecnologías provistas en el kit pueden ser utilizadas para desarrollar o mejorar ciertas capacidades en las soluciones. Estas capacidades incluyen: determinación de problemas, administración común de sistemas, e instalación y despliegue de soluciones. Las tecnologías se encuentran soportadas por varias herramientas. Autonomic Manager Engine, Generic Log Adapter y Log and Trace Analyzer cubren las áreas de determinación de problemas. Se provee en el kit además del Integrated Solutions Console, una herramienta basada en Eclipse[71] para construir capacidades efectivas de administración, entre otras herramientas que pueden ser usadas para depurar problemas complejos en el ambiente del sistema.

El camino hacia una informática autonómica parece hoy una tarea de varios años. La adopción por parte de la industria del software de los diferentes niveles de madurez autónoma es la meta para reducir la complejidad de los ambientes distribuidos.

[71] Entorno de desarrollo genérico escrito en Java.

Capítulo 5 Modelos de Arquitectura de Sistemas Distribuidos Administrables

Sistemas administrables: el nuevo reto en el diseño de Sistemas Distribuidos

Genial, cierto, pero no exenta de inconvenientes, la idea detrás de los sistemas distribuidos se ha ganado un lugar privilegiado entre las tendencias dentro de la industria para la arquitectura de las soluciones de software modernas. El diseño de sistemas con estas características, implica un grupo de retos y problemas que ya se han mencionado: heterogeneidad, apertura, seguridad, escalabilidad, comportamiento ante fallos y otros que comúnmente afectan, siendo factores claves para lograr sistemas distribuidos eficientes y fiables.

Más allá de la eficiencia y fiabilidad, los sistemas distribuidos hoy -por la escala que han alcanzado- se enfrentan con un nuevo reto: la capacidad de un sistema distribuido de ser administrable. Instalar, actualizar y de manera general, mantener en operación e incluso sacar de operación un sistema distribuido, requiere esfuerzo y consumo de recursos tanto económicos como de personal para aquellas empresas en que los procesos de negocio se encuentran soportados por una plataforma de software.

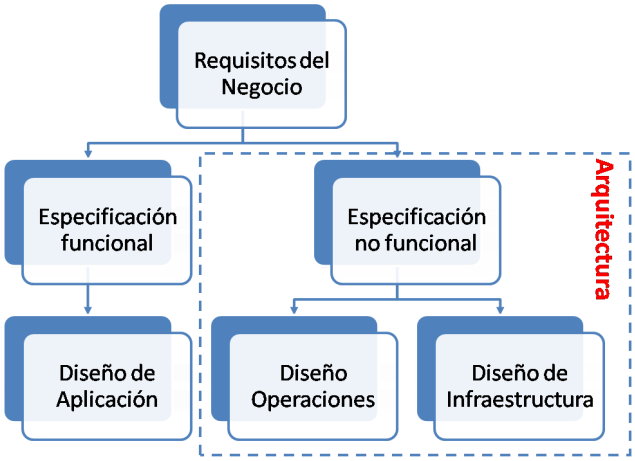
El problema de los sistemas distribuidos sin características o con características limitadas de administración parece, en una primera aproximación, un problema que afecta solamente a grandes empresas y a grandes sistemas distribuidos, pero la práctica demuestra que incluso sistemas distribuidos de una escala media pueden ser afectados.

Anteriormente se ha hecho referencia a las iniciativas de diseño y desarrollo de soluciones administrables. Tanto la Iniciativa de Sistemas Dinámicos (DSI) de Microsoft y la Iniciativa de Informática Autónoma de IBM, coinciden en la incorporación de los detalles concernientes al ambiente operacional de los Sistemas Distribuidos desde las fases de análisis y diseño y la ponderación de la arquitectura de la infraestructura de la solución y su interrelación con la arquitectura tradicional.

De acuerdo con (Microsoft Corporation, 2008) una aplicación o Sistema Distribuido es administrable cuando:

1. Es compatible con el ambiente de despliegue destino.
2. Es configurable de manera dinámica en tiempo de ejecución.
3. Interacciona correctamente con herramientas de operación y es coherente con los procesos de operación de sistemas.
4. Provee visibilidad al respecto del estado de funcionamiento del sistema.

Figura 5.1 Nueva concepción de la Arquitectura de Sistemas Distribuidos



El diseño y desarrollo de sistemas administrables tiene como objetivo fundamental facilitar el proceso de operaciones de los sistemas: desde el despliegue de la solución hasta su puesta fuera de funcionamiento. Para lograr esto es necesario tener en cuenta las características mencionadas desde la concepción de la Arquitectura del sistema, mediante la extracción de la especificación no funcional del sistema de las características de administración –operaciones e infraestructura (Figura 5.1)- al mismo nivel de abstracción que otros requisitos no funcionales como la portabilidad, disponibilidad, rendimiento, entre otros.

Arquitectura de Sistemas Distribuidos Administrables

La incorporación de las características de los Sistemas Distribuidos Administrables como un nuevo grupo de requisitos no funcionales del software, implica considerar un grupo de conceptos que -si bien eran tratados con anterioridad en la documentación o descripción de la Arquitectura de Software- no tenían un papel relevante en el diseño arquitectónico inicial.

De las características mencionadas, quizás la *compatibilidad con el ambiente de despliegue destino* era parcialmente tratada. Parcial porque la descripción del ambiente de despliegue se hacía por parte de los arquitectos de la solución, y no sería correcto afirmar "*compatibilidad*" en la justa medida en que la compatibilidad con el ambiente de despliegue destino requiere de la coherencia de características de dos especificaciones: la infraestructura ideal del ambiente de ejecución y la infraestructura real.

Del resto de las características, poco o nada. Es un hecho que la *configuración dinámica en tiempo de ejecución* de componentes de un Sistema Distribuido y como estos cambios de configuración afectan a otros componentes relacionados, es generalmente omitida en las consideraciones del diseño de múltiples Sistemas Distribuidos. Las propuestas de interacción con herramientas de operación y la visibilidad del estado de funcionamiento se pueden considerar novedosas.

Si se percibe la documentación de la Arquitectura de Software -y la Arquitectura de Software en sí- como un plano estructural de un sistema, es posible afirmar que no considerar de manera extensiva e intencional la arquitectura de la infraestructura de ejecución de un Sistema Distribuido, es un error equivalente a no considerar el suelo sobre el que se construye un edificio en la arquitectura tradicional: **puede conllevar al fracaso**.

La perspectiva fragmentada en la Arquitectura de Sistemas Distribuidos

En el Capítulo 1 se ha hecho referencia a los grupos de stakeholders involucrados en la Arquitectura de un Sistema de Software. Tradicionalmente –y sin restarle consideración al resto de los grupos- la Arquitectura reviste un interés mayor en los grupos de Desarrolladores, Arquitectos y Probadores -y en buena medida al Personal de Mantenimiento- debido a la interrelación estrecha de estos con lo establecido en la Arquitectura del sistema. Es evidente que para incorporar conceptos, diseños y conocimientos acerca de la infraestructura de despliegue y operaciones de un Sistema Distribuido, los grupos de Personal de Soporte y Administradores del Sistema deben tomar un papel más activo en la elaboración y la toma de las decisiones del diseño de alto nivel que conforma la Arquitectura de un Sistema Distribuido Administrable. Estos serían los usuarios finales potenciales de las nuevas características adicionadas,

además de ser el personal capacitado con el conocimiento necesario para tomar decisiones en el diseño de la infraestructura.

Se necesita entonces la unificación de las perspectivas de los grupos de ***stakeholders tradicionales*** (Desarrolladores, Arquitectos, Probadores) y los grupos relacionados con la administración (Personal de Soporte y Administradores del Sistema). Conocido es que, en la mayoría de los casos, ni unos ni otros son especialistas en las áreas experiencias respectivas al otro grupo: Arquitectura de Software y Telecomunicaciones. Por tanto se evidencia que:

I. Actualmente, la visión al respecto de la Arquitectura de un Sistema Distribuido ***se encuentra fragmentada*** en: Arquitectura de Solución –descripción tradicional de la Arquitectura de Software en elementos, relaciones, restricciones y configuraciones- y Arquitectura de Infraestructura –como la descripción del ambiente de ejecución de un sistema en términos más cercanos a las telecomunicaciones (enrutadores, conmutadores, etc) que a la Arquitectura de Software.
II. Se hace necesario reparar dicha fragmentación en un nivel de abstracción que permita a ambos grupos –tradicionales y de administración- tributar a la Arquitectura de un Sistema Distribuido de manera conexa y coordinada.

Arquitectura de Infraestructura y Arquitectura de Solución

La Arquitectura de Infraestructura en (The Open Group, 2007) "*denota la arquitectura de bajo nivel del hardware, las redes y los sistemas de software (algunas veces llamados "middlewares") que soportan las aplicaciones de software y sistemas de negocio en una empresa.*"[72] El término de Arquitectura de Infraestructura es relativamente nuevo. Hasta hace algunos años no se consideraba que la infraestructura que soportaba a un sistema de software debía ser analizada desde un punto de vista estructural. Con el surgimiento de los Sistemas Distribuidos y la creciente complejidad que han alcanzado, la infraestructura de ejecución y soporte para estos ha crecido también en complejidad y dimensión. De ahí la necesidad de aplicar principios de diseño y documentación arquitectónicos para mitigar esta complejidad creciente, además de constituir un vehículo de comunicación entre los responsables.

[72] (The Open Group, 2007)

El papel de la infraestructura consiste en dar soporte a requisitos de los sistemas. Desde el punto de vista lógico, el rol de soporte ocurre a través de tres servicios fundamentales: ***alojamiento***, ***comunicación*** y ***mejora de cualidades sistémicas***. El alojamiento y la comunicación han sido las relaciones fundamentales de los sistemas con su infraestructura. Los componentes de los sistemas de software se alojan en componentes de hardware de la infraestructura utilizando su capacidad de procesamiento o almacenamiento y se comunican utilizando protocolos y vías a través de medios físicos de comunicación existentes.

Existen componentes dentro de la infraestructura de ejecución de un sistema, generalmente presentes en Sistemas Distribuidos, cuyo propósito es el de mejorar o potenciar una o varias cualidades sistémicas (requisitos no funcionales) –ver Tabla 5.1.

Tabla 5.1 Componentes de infraestructura que mejoran cualidades sistémicas

Componentes	Cualidades afectadas
Servidores de Seguridad (Certificados, Antivirus, Firewalls, Detección de Intrusos)	Seguridad
Servidores de directorio(LDAP)	Seguridad
Clústeres (Balance de Carga, Fallo)	Desempeño, Escalabilidad, Rendimiento, Disponibilidad
Servidores de Monitorización	Manejabilidad
Servidores de Virtualización	Rendimiento, Escalabilidad, Coste de infraestructura

Estos componentes constituyen piedras angulares en la construcción de Sistemas Distribuidos fiables y eficientes. Se hace necesario aclarar que los componentes referenciados en la Tabla 5.1 no son los únicos presentes dentro de una infraestructura de ejecución de un sistema. Existen otros componentes –enrutadores, conmutadores, etc- que si bien su mal funcionamiento puede incidir en fallos dentro del sistema, no tienen incidencia directa a la hora de diseñar un Sistema Distribuido Administrable y escapan a la competencia de la Arquitectura de Infraestructura desde la perspectiva de la Arquitectura de Solución.

Unificando las perspectivas: modelos de Infraestructura y Despliegue

Para la unificación de las perspectivas componentes de la Arquitectura de Sistemas Distribuidos es necesario definir un marco de entendimiento común entre las partes. Así, es imprescindible que los Desarrolladores, Arquitectos y Personal de Mantenimiento de un Sistema Distribuido sean capaces de

entender términos afines a la descripción de la infraestructura de ejecución de la solución y, que los Administradores y Personal de Soporte sean capaces de describir los elementos significativos en conceptos que sean comprensibles por el resto de los grupos de stakeholders.

La idea subyacente es remover los detalles que se consideren poco relevantes o irrelevantes para ambas partes. La fórmula para lograr el balance en la selección de los términos más apropiados recae en que para los Arquitectos, Desarrolladores y Personal de Mantenimiento es más factible entender ciertos términos relacionados con la infraestructura que para los grupos de Administradores y Personal de Soporte entender conceptos de la Arquitectura de Software. No hay que perder de vista que el propósito final es el diseño y construcción de un Sistema Distribuido y la concepción de su Arquitectura. Por tanto el sistema debe quedar descrito con una perspectiva lo más familiar posible a los responsables principales de la Arquitectura: Arquitectos, Desarrolladores, Probadores y Personal de Mantenimiento en general.

Modelo de Infraestructura

Para lograr el entendimiento común se propone -como primer paso- el *Modelo de Infraestructura de Sistemas Distribuidos (Modelo de Infraestructura)* -Figura 5.2-, como extensión a los *Modelos de Arquitectura de Sistemas Distribuidos* propuestos en el Capítulo 3. Construido sobre la base del *Metamodelo para la Documentación de la Arquitectura de Sistemas Distribuidos*, el *Modelo de Infraestructura* recoge las características fundamentales presentes en la Arquitectura de Infraestructura Lógica que soporta a un Sistema Distribuido, además de constituir una forma de documentación para los servicios de soporte fundamentales que esta provee y para los elementos que la componen.

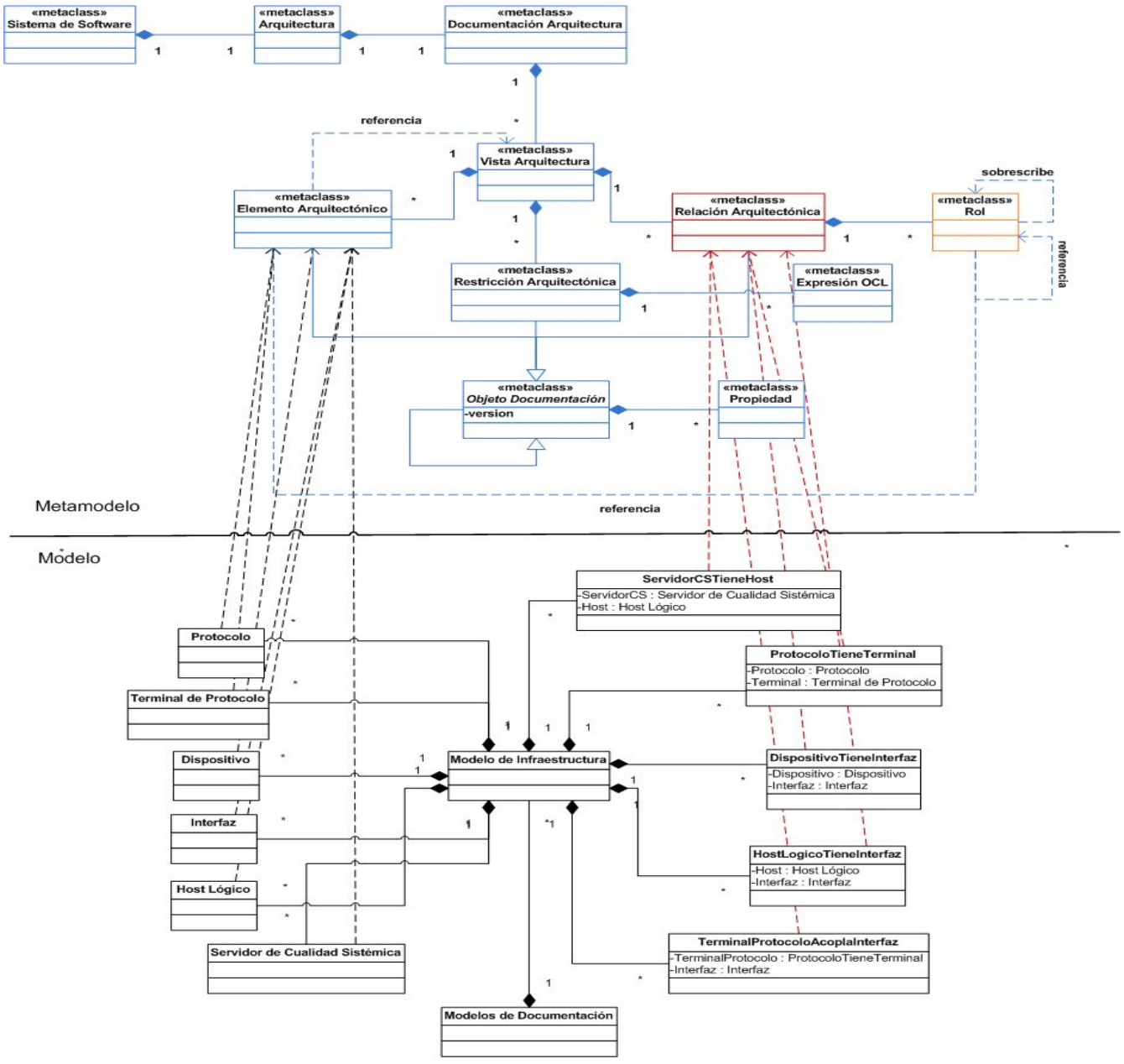
Para la confección del modelo se ha utilizado la ventaja que ofrece el *Metamodelo para la Documentación de la Arquitectura de Sistemas Distribuidos* de representar las tres tipos de vistas clásicos de documentación de la Arquitectura de Software: de módulo, de componentes (elementos) y conectores, y de localización. El Modelo de Infraestructura se encuentra orientado al estilo Cliente/Servidor -aunque no de manera purista, como consecuencia de la permisibilidad de relaciones de comunicación entre hosts con cualquier rol- con el objetivo de soportar la descripción de Arquitecturas de Infraestructuras de Sistemas Distribuidos con Arquitecturas de igual a igual y otras donde no existen relaciones jerárquicas.

Como efecto de la selección del estilo Cliente/Servidor, se utiliza el tipo de vista de componentes (elementos) y conectores, por ser considerado como base para la documentación de sistemas con este estilo.

Dominio del modelo

Para la construcción del Modelo de Infraestructura se han extraído un grupo de conceptos afines con la Arquitectura de Infraestructura. Estos han sido llevados a un nivel de abstracción lógico, alejados de las especificaciones físicas concretas que generalmente dominan el campo de la Infraestructura. Consecuentemente, es necesario esclarecer el significado de cada uno de estos términos para los propósitos del modelo con el objetivo de que sean correctamente empleados durante la elaboración de documentaciones.

Figura 5.2 Modelo de Infraestructura


«metaclass»
Sistema de Software
«metaclass»
Arquitectura
«metaclass»
Documentación Arquitectura
referencia
«metaclass»
Vista Arquitectura
«metaclass»
Elemento Arquitectónico
«metaclass»
Relación Arquitectónica
«metaclass»
Rol
sobrescribe
referencia
«metaclass»
Restricción Arquitectónica
«metaclass»
Expresión OCL
«metaclass»
Objeto Documentación
-version
«metaclass»
Propiedad
Metamodelo
referencia
Modelo
ServidorCSTieneHost
-ServidorCS : Servidor de Cualidad Sistémica
-Host : Host Lógico
Protocolo
ProtocoloTieneTerminal
-Protocolo : Protocolo
-Terminal : Terminal de Protocolo
Terminal de Protocolo
Dispositivo
Modelo de Infraestructura
DispositivoTieneInterfaz
-Dispositivo : Dispositivo
-Interfaz : Interfaz
Interfaz
HostLogicoTieneInterfaz
-Host : Host Lógico
-Interfaz : Interfaz
Host Lógico
TerminalProtocoloAcoplaInterfaz
-TerminalProtocolo : ProtocoloTieneTerminal
-Interfaz : Interfaz
Servidor de Cualidad Sistémica
Modelos de Documentación

Host Lógico

Un host es un ordenador en una red que provee procesamiento y almacenamiento -entre otros servicios- a usuarios u otros ordenadores. Generalmente un host ejecuta un sistema operativo multiusuario, algún componente middleware, etc. En la Arquitectura de Infraestructura de Sistemas Distribuidos, los hosts lógicos cumplen los roles de servidores, clientes y en arquitecturas de iguales adoptan el rol de peer (iguales).

Interfaz de Host

Especifica el punto de interacción de un host con su ambiente. Las interfaces de host describen el comportamiento provisto/esperado de un host, de aquí que las interfaces puedan ser de entrada (esperadas) o de salida (provistas).

Dispositivo

Componente de infraestructura que no posee capacidad de procesamiento independiente -al nivel de abstracción que se modela- que provee a los hosts lógicos de capacidades adicionales.

Componente de Cualidad Sistémica

Los Componentes de Cualidad Sistémica favorecen –como su nombre implica- la mejora de requisitos no funcionales a través de la incorporación de elementos de la infraestructura. Estos requisitos son persistente a los largo de los diferentes niveles de la infraestructura y permiten definir un conjunto de características y mecanismos para grupos de host. En la Tabla 5.1 se ejemplifican los Componentes de Calidad Sistémica más comunes que se pueden encontrar en la infraestructura de ejecución de un Sistema Distribuido.

Protocolo

Patrón de eventos o acciones que pueden tener lugar para una interacción entre host lógicos. Implica la obligación de las partes a cumplir convenciones acerca del orden de las interacciones, manipulación de errores, tiempos de espera, entre otros. Los protocolos existen a distintos niveles de abstracción dentro de una infraestructura. Desde los protocolos que describen la intercomunicación de las interfaces de hardware a nivel de dispositivos, hasta los protocolos para el intercambio de datos a nivel de aplicación.

Terminal de Protocolo

El terminal es la interfaz de un protocolo, define la forma en que el protocolo puede ser utilizado por un host lógico para llevar a cabo las interacciones. Los terminales de los protocolos se acoplan a las interfaces de los host permitiendo la comunicación de estos a través del protocolo del que forman parte.

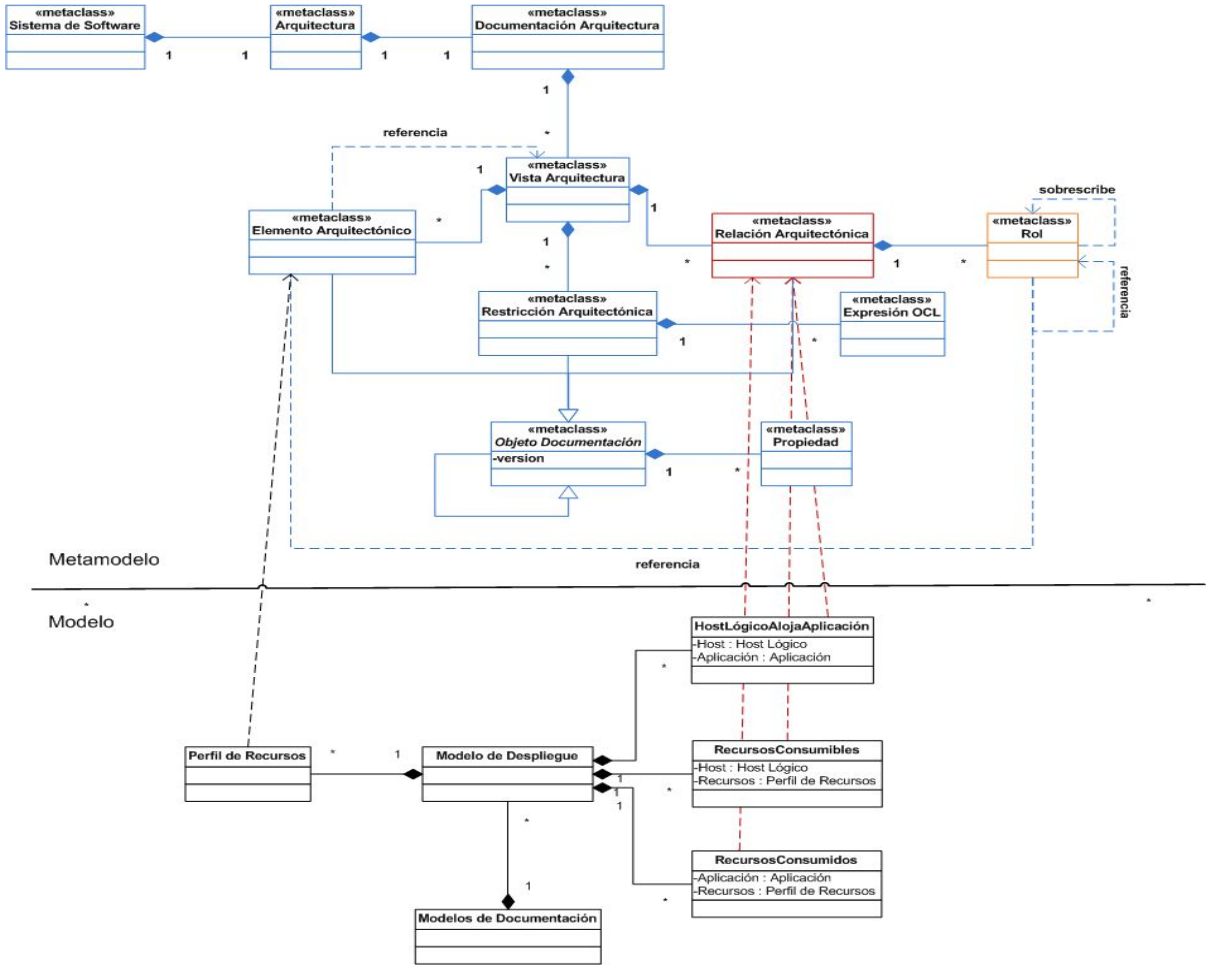
Alojamiento de Software en la Infraestructura: Modelo de Despliegue.

Como se ha visto, el ciclo de vida de un sistema de software transita por etapas, etapas que se definen en función de la visión de cada organización y la granularidad con que se quiera tratar cada uno de los hitos fundamentales dentro de este ciclo. Por poner un ejemplo: el hito del despliegue de una solución en algunas aproximaciones se trata como una fase independiente y en otras, forma parte de la fase de Operaciones de un sistema de software. Lo cierto es que el Despliegue es el punto o momento donde se interceptan los esfuerzos y el conocimiento de los Desarrolladores y Arquitectos con los de los Administradores y Arquitectos de la Infraestructura.

Un factor fundamental para lograr un despliegue exitoso es la coherencia entre los requisitos no funcionales descritos en la Arquitectura de la Solución y los servicios provistos por los elementos que componen la infraestructura. La descripción de la relación entre los componentes de software desarrollados con el ambiente donde se ejecutan y la infraestructura de comunicación, así como con los componentes de cualidades sistémicas descritos anteriormente, es fundamental para lograr la concordancia entre la Arquitectura de Solución y la Arquitectura de Infraestructura.

Para documentar estas relaciones se propone el *Modelo de Despliegue de Sistemas Distribuidos (Modelo de Despliegue)* -Figura 5.3. El Modelo de Despliegue unifica las perspectivas de la Arquitectura de Sistemas Distribuidos mediante la localización de las unidades desplegables de software propuestas en los Modelos de Aplicación y Sistema, en el ambiente de ejecución ideal para la solución, propuesto en el Modelo de Infraestructura. Como todos los modelos propuestos, el Modelo de Despliegue está construido sobre la base del *Metamodelo para la Documentación de la Arquitectura de Sistemas Distribuidos.*

Figura 5.3 Modelo de Despliegue



La idea base de este modelo es lograr la concordancia a través de los Perfiles de Recursos. Los Perfiles de Recursos, tanto los Consumibles -provistos por los elementos de la infraestructura- como los Consumidos -necesarios para los elementos desplegables- constituyen un contrato entre los Arquitectos de Infraestructura y los Arquitectos de Solución (Software) sobre las necesidades, restricciones y propiedades de unos y otros en el diseño de un Sistema Distribuido de Software.

Dominio del Modelo

El Modelo de Despliegue está compuesto por elementos extraídos en el diseño de la Arquitectura de Solución del Sistema Distribuido -expresados en los Modelos de Sistema y Aplicación- y que son considerados unidades desplegables. Se modelan las relaciones de estos con los componentes de la infraestructura y la coherencia entre los Perfiles de Recursos de ambos. La propuesta de Despliegue basado en Perfiles de Recurso posibilita la automatización posterior de las actividades dentro de esta fase producto del detalle de las descripciones con los que son abordados.

Perfil de Recursos

Conjunto de recursos necesarios para la ejecución de un Sistema Distribuido. Los Perfiles de Recursos son la base la coordinación entre la Arquitectura de Solución y la Arquitectura de Infraestructura sobre las necesidades de los Sistemas y las capacidades de los elementos de la infraestructura. La descripción de los Perfiles de Recursos para los host y para las aplicaciones que componen un Sistema Distribuido constituyen posibilitan la unificación de las perspectiva de la Arquitectura de Infraestructura desde la vista de la Arquitectura de Solución.

Conclusiones

Los Modelos para la Documentación de la Arquitectura de Sistemas Distribuidos, así como su extensión para los Sistemas Distribuidos Administrables propuestos como resultado de esta investigación, constituyen un entorno de integración mediante el cual los Arquitectos de Software y los Arquitectos de Infraestructura lograrán coordinar el conocimiento concerniente a cada una de sus áreas de experiencia con el objetivo de diseñar soluciones de software que sean administrables e interactúen dinámicamente con los requerimientos cambiantes del ambiente donde se ejecutan.

Construidos sobre la base del Metamodelo para la Documentación de Sistemas Distribuidos y apoyados en las propiedades que este les confiere, cabría agregar que los Modelos para la Documentación de la Arquitectura de Sistemas Distribuidos promueven un marco de análisis arquitectónico integrado debido a su enfoque marcadamente dirigido a estilos arquitectónicos aprovechándose así de las ventajas que ofrece este marco teórico dentro del campo de la Arquitectura de Software, fundamentalmente sus eminentes facilidades para la representación gráfica y el análisis.

Es importante señalar que tanto los Modelos como el Metamodelo subyacente están diseñados pensando en la flexibilidad y la extensibilidad, permitiendo que se agreguen posteriormente otros modelos que describan la Arquitectura de Software desde perspectivas diferentes. La simplicidad y a la vez formalidad con que se aborda la documentación de la Arquitectura de Sistemas Distribuidos hacen de los modelos y el metamodelo propuestos una especificación automatizable de la descripción de un sistema y que llegan a conformar un excelente vehículo de comunicación entre los stakeholders de un proyecto de software.

Recomendaciones

Escenarios de utilización

Utilizar la facilidad de conversión del metamodelo a DSL para la automatización.

OCLs.

Diagramas. Selección de estereotipos.

Referencias bibliográficas

Analyzing Architectural Styles with Alloy. **Garlan, David and Jung Soo, Kim. 2006.** Portland, Maine : ACM, 2006. Proceedings of Workshop on the Role of the Software Architecture for Testing and Analysis(ROSATEA). p. 11. 1-59593-459.

Bachmann, Felix, y otros. 2000. *Software Architecture Documentation in Practice: Documenting Architectural Layers.* [Documento PDF] s.l. : Software Engineering Institute Carnegie Mellon University, 2000.

Bass, Len, Clements, Paul and Kazman, Rick. 2003. *Software Achitecture in Practice.* s.l. : Addison Wesley, 2003.

Black, Keith, y otros. 2007. *Datacenter Reference Guide.* [Documento PDF] s.l. : Sun Microsystems, 2007.

Brunelli, Mark. 2004. The Holy Grail of model-driven development. *SOA News.* [En línea] 10 de Agosto de 2004. [Citado el: 2010 de Marzo de 31.] http%3A%2F%2FsearchSOA%2Etechtarget%2Ecom%2Fnews%2Finterview%2F0%2C289202%2Csid26%5Fgci999474%2C00%2Ehtml&title=The+Holy+Grail+of+model%2Ddriven+development.

Buschmann, Frank, y otros. 1996. *Pattern-oriented Software Architecture: A System of Patterns.* s.l. : John Wiley & Sons, 1996. pág. 457. 0-471-95869-7.

Chappell, David. 2007. *Introducing Microsoft System Center.* s.l. : Microsoft Corporation, 2007.

Chappell, David. 2008. *What is application lifecycle management?* s.l. : Chappell & Associates, 2008.

Chiu, David y Tsui, D.L. 2004. *Modeling The Enterprise IT Infrastructure. An IT Service Management Approach.* [Documento PDF] 2004.

Clements, Paul, et al. 2003. *Documenting Software Architectures: Views and Beyond.* [ed.] Pearson Education. Segunda Edicion. s.l. : Addison Wesley, 2003. p. 560. 0-201-70372-6.

Comentario en discusión sobre teoría y práctica de la ingeniería de software. **Sharp, I. P. 1979.** [ed.] J. N. Bruxton y B. Randall. Roma : Petrochelli/Charter, 1979. Software Engineering Concepts and Techniques: Proceedings of the NATO conferences.

Computerworld Magazine. 2007. Hacia una gestión de sistemas dinámicos La mejora de la gestión de sistemas y servicios de TI es. *IDG.* [En línea] 15 de Junio de 2007. [Citado el: 2009 de Octubre de 30.] http://www.idg.es/computerworld/articulo.asp?id=184569..

Cook, Steve, y otros. 2007. *Domain-Specific Development with Visual Studio DSL Tools.* s.l. : Addison-Wesley, 2007. 0-321-39820-3.

Coulouris, G., Dollimore, J. y Kindberg, T. 2005. *Distributed Systems. Concepts and Design.* [ed.] Addison Wesley. Tercera Edición. Londres : Pearson Education, 2005.

Distributed Management Task Force. 2006. *Web Services for Management (WS-Management).* [Documento] s.l. : Distributed Management Task Force, Inc, 2006.

Foundations for the study of software architecture. **Perry, Dewayne E. y Wolf, Alexander L. 1992.** s.l. : ACM SIGSOFT, 1992. ACM SIGSOFT Software Engineering Notes.

Fowler, Martin. 2009. Using Domain Specific Language. [En línea] 9 de Abril de 2009. [Citado el: 20 de Abril de 2010.] http://martinfowler.com/dslwip/UsingDsls.html.

Greenfield, Jack y Short, Keith. 2004. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.* Indianapolis : John Wiley & Sons, 2004. 0-471-20284-3.

Hernández León, Rolando Alfredo. 2010. *Árbol de Problemas de la Gestión de Proyectos.* [Documento Microsoft Word] Ciudad de La Habana : Universidad de las Ciencias Informáticas, 2010.

IBM Corporation. 2005. *Autonomic Computing Toolkit User's Guide.* s.l. : IBM Corporation, 2005.

—. 2001. *Autonomic Computing: IBM's Perspective of State of Information Technology.* New York : IBM Corporation, 2001.

—. 2005. *Rational Unified Process v7.0.1.* [Digital] s.l. : IBM, 2005.

IBM Corporation, International Technical Support Organization. 1996. *OS/2 Installation Techniques: The CID Guides.* Boca Raton, Florida, Estados Unidos de America : s.n., Mayo 1996.

IEEE. Septiembre, 2000. *Recommended Practice for Architectural Description of Software-Intensive Systems.* s.l. : IEEE-Std-1471- 2000, Septiembre, 2000.

IT Governance Institute. 2005. *COBIT 4.0. Objetivos de Control. Directrices Gerenciales. Modelos de Madurez.* 2005.

Jumelet, Daniel. 2007. Infrastructure Architecture. *MSDN Library.* [En línea] Marzo de 2007. [Citado el: 28 de Abril de 2010.] http://msdn.microsoft.com/en-us/library/bb402960.aspx.

Kaspersky Lab. 2007. *Kaspersky Administration Kit 6.0. Guía de implementación.* 2007.

—. 2007. *Kaspersky Administration Kit 6.0. Manual del administrador.* 2007.

Kephart, Jeffrey O. y Chess, David M. 2003. *The Vision of Autonomic Computing.* [Documento] s.l. : IEEE Computer Society, 2003.

Lehman, M. M. y Ramil, J. F. 1985. *An approach to a theory of software evolution.* 1985.

Malan, Ruth. 2009. Picture IT: The Art of Drawing People in. *Bredemeyer.* [En línea] 16 de Diciembre de 2009. [Citado el: 24 de Marzo de 2010.] http://www.bredemeyer.com/ArchitectingProcess/VAPColumns/PictureIT.htm.

Marquès i Puig, Joan Manuel, Vilajosana i Guillén, Xavier y García López, Pedro A. 2007. *Arquitecturas, paradigmas y aplicaciones de los sistemas distribuidos.* Barcelona : Universitat Oberta de Catalunya, 2007.

Meta Architecting: Towered a New Generation of Architecture Description Languages. **Smeda, Adel, Khammaci, Tahar y Oussalah, Mourad. 2005.** 4, Nantes, Francia : Sciencie Publications, 2005, Journal of Computer Science, Vol. 1. 1549-3636.

Michigan Department of Technology, Management & Budget. 2010. DIT - Infrastructure Architecture. *DMTB.* [En línea] 2010. [Citado el: 10 de Mayo de 2010.] http://www.michigan.gov/dit/0,1607,7-139-34305-108229--,00.html.

Microsoft Corporation. 2008. *Design for Operations: Designing Manageable Applications.* [Documento PDF] s.l. : Patterns & Practice, 2008.

—. 2003. *Enterprise Solution Patterns Using Microsoft .NET.* [Documento PDF] s.l. : Microsoft Patterns & Practices, 2003.

—. 2009. *Microsoft Application Architecture Guide.* [Documento PDF] s.l. : Microsoft Patterns & Practices, 2009. 9780735627109.

—. 2004. *Microsoft Dynamic Systems Initiative Overview.* 2004.

Microsoft Corporation. Solution Accelerators. 2008. *Microsoft Deployment Toolkit 2008. Getting Started Guide.* 2008.

Nadiminti, Krishna, Dias de Assunção, Marcos y Buyya, Rajkumar. 2006. *Distributed Systems and Recent Innovations: Challenges and Benefits.* Department of Computer Science and Software Engineering, University of Melbourne. Melbourne, Australia : Grid Computing and Distributed Systems Laboratory, 2006.

Naranjo, Mauricio. 2005. *Fundamentos de Definición de Arquitectura de Software.* [Presentación de Microsoft PowerPoint] Bogotá : Lucasian Labs Ltda, 2005.

Object Management Group. 2003. *OMG Unified Modeling Language Specification v.1.3.* [Documento PDF] s.l. : Object Management Group, 2003.

Radhakrishnan, Ramesh y Radhakrishnan, Rakesh. 2004. *IT Infrastructure Building Blocks.* [Documento PDF] s.l. : Sun Microsystems. Sun Professional Services, 2004.

Reinoso, Billy. 2005. *Arquitectura de Software orientada a servicios.* [Webcast] Buenos Aires : s.n., 2005.

Reinoso, Carlos Billy. 2005. *Architect Academy Webcast.* [Presentación de Microsoft PowerPoint] Buenos Aires : Microsoft Patterns & Practices, 2005. Vol. II.

—. 2004. *Introducción a la Arquitectura de Software.* Buenos Aires : Universidad de Buenos Aires, 2004.

—. 2004. *Lenguajes de Descripción de Arquitectura (ADL).* [Documento PDF] Buenos Aires : Universidad de Buenos Aires, 2004.

Reinoso, Carlos Billy y Kicillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* [Documento] Buenos Aires : Universidad de Buenos Aires, 2004.

rPath Corporation. 2008. *rPath for Release Automation. Automating Application Deployment and Maintenance.* [Documento] s.l. : rPath Corporation, 2008.

Sánchez Téllez, Eddy. 2008. *Especificación de la Arquitectura Base del Sistema de Gestión de Emergencias de Seguridad Ciudadana 171.* [Documento] La Habana : Universidad de las Ciencias Informáticas, 2008.

Schneider, Fred B. 1993. What Good are Models and What Models are Good? *Distributed Systems.* s.l. : Addison Wesley, 1993.

Shaw, Mary y Garlan, David. 1996. *Software Architecture: Perspectives on an emerging discipline.* Upper Saddle River : Prentice Hall, 1996.

Soley, Richard; OMG Staff Strategy Group. 2000. *Model Driven Architecture.* [Documento PDF] s.l. : Object Management Group, 2000.

Sommerville, Ian. 2004. *Software Engineering.* Séptima Edición. 2004.

Sun Microsystems. 2001. *Suntone Architecture Methodology: A 3-D approach to architectural desgin. Key Concepts and Overview.* [Documento PDF] Palo Alto, CA : Sun Microsystems, 2001.

Taylor, Richard N., Medvidovic, Nenad and Dashofy, Eric M. 2009. *Software architecture: Foundations, Theory, and Practice.* s.l. : John Wiley & Sons, 2009. 978-0470167748.

The 4+1 View Model of Architecture. **Krutchen, Phillip. 1995.** 6, s.l. : IEEE Software, 1995, Vol. XII.

The Open Group. 2007. *TOGAF: Version 8.1 Enterprise Edition Study Guide.* s.l. : Van Haren Publishing, 2007. 978-9087530938.

United State Patents and Trademarks Office. Life Cycle Management Manual. *United State Patents and Trademarks Office.* [Online] http://www.uspto.gov/web/offices/cio/lcm/lcm.htm.

Universidad de Buenos Aires.Facultad de Ciencias Exactas. Departamento de Computación. 2004. *Tendencias Tecnológicas en Arquitectura y Desarrollo de Aplicaciones.* [Electrónico] s.l. : Universidad de Buenos Aires, Universidad de Buenos Aires, 2004.

Vallecillo, Antonio. 2005. *Model Driven Development.* [Documento PDF] Málaga : Universidad de Málaga, 2005.

von Löwis, Martin. 2009. *Middleware and Distributed Systems.* [Documento PDF] 2009.

Woods, Eoin y Rozanski, Nick. 2006. *Using Architectural Perspectives.* [Documento PDF] 2006.

Bibliografía

Analyzing Architectural Styles with Alloy. **Garlan, David and Jung Soo, Kim. 2006.** Portland, Maine : ACM, 2006. Proceedings of Workshop on the Role of the Software Architecture for Testing and Analysis(ROSATEA). p. 11. 1-59593-459.

Bachmann, Felix, y otros. 2000. *Software Architecture Documentation in Practice: Documenting Architectural Layers.* [Documento PDF] s.l. : Software Engineering Institute Carnegie Mellon University, 2000.

Bass, Len, Clements, Paul and Kazman, Rick. 2003. *Software Achitecture in Practice.* s.l. : Addison Wesley, 2003.

Black, Keith, y otros. 2007. *Datacenter Reference Guide.* [Documento PDF] s.l. : Sun Microsystems, 2007.

Brunelli, Mark. 2004. The Holy Grail of model-driven development. *SOA News.* [En línea] 10 de Agosto de 2004. [Citado el: 2010 de Marzo de 31.] http%3A%2F%2FsearchSOA%2Etechtarget%2Ecom%2Fnews%2Finterview%2F0%2C289202%2Csid26%5Fgci999474%2C00%2Ehtml&title=The+Holy+Grail+of+model%2Ddriven+development.

Buschmann, Frank, y otros. 1996. *Pattern-oriented Software Architecture: A System of Patterns.* s.l. : John Wiley & Sons, 1996. pág. 457. 0-471-95869-7.

Chappell, David. 2007. *Introducing Microsoft System Center.* s.l. : Microsoft Corporation, 2007.

Chappell, David. 2008. *What is application lifecycle management?* s.l. : Chappell & Associates, 2008.

Chiu, David y Tsui, D.L. 2004. *Modeling The Enterprise IT Infrastructure. An IT Service Management Approach.* [Documento PDF] 2004.

Clements, Paul, et al. 2003. *Documenting Software Architectures: Views and Beyond.* [ed.] Pearson Education. Segunda Edicion. s.l. : Addison Wesley, 2003. p. 560. 0-201-70372-6.

Comentario en discusión sobre teoría y práctica de la ingeniería de software. **Sharp, I. P. 1979.** [ed.] J. N. Bruxton y B. Randall. Roma : Petrochelli/Charter, 1979. Software Engineering Concepts and Techniques: Proceedings of the NATO conferences.

Computerworld Magazine. 2007. Hacia una gestión de sistemas dinámicos La mejora de la gestión de sistemas y servicios de TI es. *IDG.* [En línea] 15 de Junio de 2007. [Citado el: 2009 de Octubre de 30.] http://www.idg.es/computerworld/articulo.asp?id=184569..

Cook, Steve, y otros. 2007. *Domain-Specific Development with Visual Studio DSL Tools.* s.l. : Addison-Wesley, 2007. 0-321-39820-3.

Coulouris, G., Dollimore, J. y Kindberg, T. 2005. *Distributed Systems. Concepts and Design.* [ed.] Addison Wesley. Tercera Edición. Londres : Pearson Education, 2005.

Distributed Management Task Force. 2006. *Web Services for Management (WS-Management).* [Documento] s.l. : Distributed Management Task Force, Inc, 2006.

Foundations for the study of software architecture. **Perry, Dewayne E. y Wolf, Alexander L. 1992.** s.l. : ACM SIGSOFT, 1992. ACM SIGSOFT Software Engineering Notes.

Fowler, Martin. 2009. Using Domain Specific Language. [En línea] 9 de Abril de 2009. [Citado el: 20 de Abril de 2010.] http://martinfowler.com/dslwip/UsingDsls.html.

Greenfield, Jack y Short, Keith. 2004. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.* Indianapolis : John Wiley & Sons, 2004. 0-471-20284-3.

Hernández León, Rolando Alfredo. 2010. *Árbol de Problemas de la Gestión de Proyectos.* [Documento Microsoft Word] Ciudad de La Habana : Universidad de las Ciencias Informáticas, 2010.

IBM Corporation. 2005. *Autonomic Computing Toolkit User's Guide.* s.l. : IBM Corporation, 2005.

—. 2001. *Autonomic Computing: IBM's Perspective of State of Information Technology.* New York : IBM Corporation, 2001.

—. 2005. *Rational Unified Process v7.0.1.* [Digital] s.l. : IBM, 2005.

IBM Corporation, International Technical Support Organization. 1996. *OS/2 Installation Techniques: The CID Guides.* Boca Raton, Florida, Estados Unidos de America : s.n., Mayo 1996.

IEEE. Septiembre, 2000. *Recommended Practice for Architectural Description of Software-Intensive Systems.* s.l. : IEEE-Std-1471- 2000, Septiembre, 2000.

IT Governance Institute. 2005. *COBIT 4.0. Objetivos de Control. Directrices Gerenciales. Modelos de Madurez.* 2005.

Jumelet, Daniel. 2007. Infrastructure Architecture. *MSDN Library.* [En línea] Marzo de 2007. [Citado el: 28 de Abril de 2010.] http://msdn.microsoft.com/en-us/library/bb402960.aspx.

Kaspersky Lab. 2007. *Kaspersky Administration Kit 6.0. Guía de implementación.* 2007.

—. 2007. *Kaspersky Administration Kit 6.0. Manual del administrador.* 2007.

Kephart, Jeffrey O. y Chess, David M. 2003. *The Vision of Autonomic Computing.* [Documento] s.l. : IEEE Computer Society, 2003.

Lehman, M. M. y Ramil, J. F. 1985. *An approach to a theory of software evolution.* 1985.

Malan, Ruth. 2009. Picture IT: The Art of Drawing People in. *Bredemeyer.* [En línea] 16 de Diciembre de 2009. [Citado el: 24 de Marzo de 2010.] http://www.bredemeyer.com/ArchitectingProcess/VAPColumns/PictureIT.htm.

Marquès i Puig, Joan Manuel, Vilajosana i Guillén, Xavier y García López, Pedro A. 2007. *Arquitecturas, paradigmas y aplicaciones de los sistemas distribuidos.* Barcelona : Universitat Oberta de Catalunya, 2007.

Meta Architecting: Towered a New Generation of Architecture Description Languages. **Smeda, Adel, Khammaci, Tahar y Oussalah, Mourad. 2005.** 4, Nantes, Francia : Sciencie Publications, 2005, Journal of Computer Science, Vol. 1. 1549-3636.

Michigan Department of Technology, Management & Budget. 2010. DIT - Infrastructure Architecture. *DMTB.* [En línea] 2010. [Citado el: 10 de Mayo de 2010.] http://www.michigan.gov/dit/0,1607,7-139-34305-108229--,00.html.

Microsoft Corporation. 2008. *Design for Operations: Designing Manageable Applications.* [Documento PDF] s.l. : Patterns & Practice, 2008.

—. 2003. *Enterprise Solution Patterns Using Microsoft .NET.* [Documento PDF] s.l. : Microsoft Patterns & Practices, 2003.

—. 2009. *Microsoft Application Architecture Guide.* [Documento PDF] s.l. : Microsoft Patterns & Practices, 2009. 9780735627109.

—. 2004. *Microsoft Dynamic Systems Initiative Overview.* 2004.

Microsoft Corporation. Solution Accelerators. 2008. *Microsoft Deployment Toolkit 2008. Getting Started Guide.* 2008.

Nadiminti, Krishna, Dias de Assunção, Marcos y Buyya, Rajkumar. 2006. *Distributed Systems and Recent Innovations: Challenges and Benefits.* Department of Computer Science and Software Engineering, University of Melbourne. Melbourne, Australia : Grid Computing and Distributed Systems Laboratory, 2006.

Naranjo, Mauricio. 2005. *Fundamentos de Definición de Arquitectura de Software.* [Presentación de Microsoft PowerPoint] Bogotá : Lucasian Labs Ltda, 2005.

Object Management Group. 2003. *OMG Unified Modeling Language Specification v.1.3.* [Documento PDF] s.l. : Object Management Group, 2003.

Radhakrishnan, Ramesh y Radhakrishnan, Rakesh. 2004. *IT Infrastructure Building Blocks.* [Documento PDF] s.l. : Sun Microsystems. Sun Professional Services, 2004.

Reinoso, Billy. 2005. *Arquitectura de Software orientada a servicios.* [Webcast] Buenos Aires : s.n., 2005.

Reinoso, Carlos Billy. 2005. *Architect Academy Webcast.* [Presentación de Microsoft PowerPoint] Buenos Aires : Microsoft Patterns & Practices, 2005. Vol. II.

—. 2004. *Introducción a la Arquitectura de Software.* Buenos Aires : Universidad de Buenos Aires, 2004.

—. 2004. *Lenguajes de Descripción de Arquitectura (ADL).* [Documento PDF] Buenos Aires : Universidad de Buenos Aires, 2004.

Reinoso, Carlos Billy y Kicillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* [Documento] Buenos Aires : Universidad de Buenos Aires, 2004.

rPath Corporation. 2008. *rPath for Release Automation. Automating Application Deployment and Maintenance.* [Documento] s.l. : rPath Corporation, 2008.

Sánchez Téllez, Eddy. 2008. *Especificación de la Arquitectura Base del Sistema de Gestión de Emergencias de Seguridad Ciudadana 171.* [Documento] La Habana : Universidad de las Ciencias Informáticas, 2008.

Schneider, Fred B. 1993. What Good are Models and What Models are Good? *Distributed Systems.* s.l. : Addison Wesley, 1993.

Shaw, Mary y Garlan, David. 1996. *Software Architecture: Perspectives on an emerging discipline.* Upper Saddle River : Prentice Hall, 1996.

Soley, Richard; OMG Staff Strategy Group. 2000. *Model Driven Architecture.* [Documento PDF] s.l. : Object Management Group, 2000.

Sommerville, Ian. 2004. *Software Engineering.* Séptima Edición. 2004.

Sun Microsystems. 2001. *Suntone Architecture Methodology: A 3-D approach to architectural desgin. Key Concepts and Overview.* [Documento PDF] Palo Alto, CA : Sun Microsystems, 2001.

Taylor, Richard N., Medvidovic, Nenad and Dashofy, Eric M. 2009. *Software architecture: Foundations, Theory, and Practice.* s.l. : John Wiley & Sons, 2009. 978-0470167748.

The 4+1 View Model of Architecture. **Krutchen, Phillip. 1995.** 6, s.l. : IEEE Software, 1995, Vol. XII.

The Open Group. 2007. *TOGAF: Version 8.1 Enterprise Edition Study Guide.* s.l. : Van Haren Publishing, 2007. 978-9087530938.

United State Patents and Trademarks Office. Life Cycle Management Manual. *United State Patents and Trademarks Office.* [Online] http://www.uspto.gov/web/offices/cio/lcm/lcm.htm.

Universidad de Buenos Aires.Facultad de Ciencias Exactas. Departamento de Computación. 2004. *Tendencias Tecnológicas en Arquitectura y Desarrollo de Aplicaciones.* [Electrónico] s.l. : Universidad de Buenos Aires, Universidad de Buenos Aires, 2004.

Vallecillo, Antonio. 2005. *Model Driven Development.* [Documento PDF] Málaga : Universidad de Málaga, 2005.

von Löwis, Martin. 2009. *Middleware and Distributed Systems.* [Documento PDF] 2009.

Woods, Eoin y Rozanski, Nick. 2006. *Using Architectural Perspectives.* [Documento PDF] 2006.

Anexos

Anexo 1 Caso de Estudio de Sistema Distribuido: Sistema de Gestión de Emergencias de Seguridad Ciudadana (SIGESC 171).

Autor: Eddy Sánchez Téllez. Arquitecto Principal.

Arquitectónicamente, la solución está compuesta por nueve aplicaciones y tres servidores lógicos –ver Tabla 1. Cada uno de estos aplicaciones están diseñados con estilo arquitectónico multicapas y las aplicaciones se encuentran compuestas por unidades de implementación menores llamadas módulos[73].

Tabla 1 Subsistemas y servidores de la arquitectura de SIGESC 171

Subsistemas	Servidores
Recepción de Llamada	Servidor de Sincronización
Despacho	Servidor de Aplicaciones
Supervisión de Operadores	Servidor de Base de Datos
Supervisión de Despacho	
Supervisión General	
Administración y Control de Recursos (Web)	
Configuración de Operaciones	
Administración Informática	
Mapificación	

Durante el proceso de desarrollo de la solución se pudieron identificar algunos problemas fundamentales que provienen básicamente de la concepción y la documentación de la arquitectura del sistema. El problema más palpable es quizás la falta de concordancia entre las restricciones de la Arquitectura y la implementación del sistema. Componentes con dependencias a componentes en capas donde se restringe su relación, mala aplicación de estándares de codificación, así como problemas en la localización de lógicas de aplicación en componentes de negocio o lógicas de negocio en componentes de acceso a datos los cuales afectan las ventajas de utilización de la arquitectura seleccionada con las consecuencias que esto implica.

[73] Ver (Sánchez Téllez, 2008) para una descripción más detallada.

También durante el desarrollo fue necesaria la incorporación de equipos adicionales para impulsar el trabajo en algunos subsistemas. Aquí se detectaron dos situaciones fundamentales: primero, que la transferencia del conocimiento acerca de la Arquitectura del Sistema no poseía una herramienta capaz de introducir a los nuevos miembros de los equipos y segundo, que el personal adicional que se le asignaba trabajar sobre un subsistema se le imposibilitaba desarrollar nuevas funcionalidades sobre una misma capa debido a la deficiencia de las herramientas tanto de desarrollo como de control de versiones para permitir el desarrollo horizontal en paralelo sobre un mismo componente.

Se puede señalar además la falta de herramientas para configurar un entorno de prueba fiable y que se asemeje a la infraestructura donde se desplegará el sistema. Ha quedado claro que el entorno de desarrollo –debido a que en la mayoría de los ordenadores se encuentran instalados una gran variedad de aplicaciones- no es propicio para realizar las pruebas del sistema.

Durante el proceso de refinamiento se hizo particularmente difícil dimensionar el impacto que pueden tener los cambios realizados sobre los componentes y sus dependencias, además de que no existe una forma precisa de describir los componentes y sus asociaciones con los cambios. Además, en el refinamiento que se llevó a cabo luego de las Pruebas Piloto y el despliegue de la solución en más de un Centro de Emergencia fue posible evidencia las limitaciones de las herramientas utilizadas para analizar el impacto de la sustitución de componentes de diferentes versiones así como la diferenciación de las liberaciones del sistema en funciones de los clientes y los errores reportados por cada uno de estos.

Como características importantes del despliegue de la solución es importante tener en consideración la cantidad de subsistemas así como la fuerte dependencia entre ellos aumentan la complejidad de la configuración. Todos los subsistemas pueden ser desplegados en más de un nodo físico y en un nodo físico pueden encontrarse desplegados más de un subsistema. También el número de ordenadores dedicados a la ejecución de cada subsistema puede variar en dependencia de las necesidades del Centro de Emergencia donde es desplegada la solución.

Se puede señalar que durante la ejecución del sistema no existen herramientas propias o de terceros capaces de monitorear y controlar el estado de la ejecución de cada uno de los subsistemas y los

servicios que proveen los servidores con los que cuenta la solución, algunos de los cuales manejan información en tiempo real –por ejemplo, el Servidor de Sincronización o el Servidor de Base de Datos, y subsistemas críticos como Despacho o Recepción de Llamadas- que en caso de dejar de funcionar de manera correcta afectarían el funcionamiento del Centro con consecuencias para la seguridad ciudadana.

Uniendo estas características arquitectónicas de la solución y, conociendo que se configura y despliega de manera manual y que además el mantenimiento, la actualización, el monitoreo y el reporte de fallos para el soporte de la solución ocurre también manualmente se implican un conjunto de problemas que dan al traste con la eficiencia del proceso de despliegue y soporte de la solución, así como el mantenimiento de la solución por parte del personal técnico de los Centros de Emergencia.

Resumiendo, los problemas fundamentales son:

- El despliegue manual es propenso al error humano, los equipos de despliegue pueden cometer errores a la hora de configurar tanto la solución como el ambiente de ejecución de la misma, y una vez desplegada la solución el personal del Centro de Emergencia también pueden cometer errores en caso de redespliegue en ordenadores con fallas o otras situaciones que se presenten.
- Como el proceso de configuración y despliegue de la solución ocurre de manera manual los equipos de refinamiento y soporte de la solución no poseen la certeza de que el ambiente de configuración y ejecución del sistema es el correcto a la hora de corregir fallos o dar soporte técnico a los problemas que se puedan presentar.
- En relación con el reporte de fallos, la información de los fallos de la solución es recogida por el personal del Centro de Emergencias y puesta por escrito en un modelo que no posee información técnica suficiente para poder reproducir los errores en el ambiente de desarrollo.
- La prevención, detección y corrección de los fallos, así como el despliegue de las actualizaciones necesitan de la participación del personal de TI del Centro en el ordenador donde ocurren.
- La preparación del personal del Centro muchas veces no es la requerida y para configurar la solución, además de que es necesario conocer la lógica del negocio, son necesarios un grupo de conocimientos específicos –léase, variables de entorno del sistema operativo, registro, entre otros.