UNIVERSIDAD AUTÓNOMA METROPOLITANA

Maestría en Ciencias y Tecnologías de la Información

"Adaptación de una Metodología de Desarrollo Arquitectónico al Contexto de Equipos de Desarrollo Pequeños"

Idónea Comunicación de Resultados que para obtener el grado de

MAESTRO EN CIENCIAS

(Ciencias y Tecnologías de la Información)

PRESENTA:

Lic. José Ismael Nuñez Reyna

Asesor:

Dr. Humberto Cervantes Maceda

Sinodales:

Dr. Cuauhtémoc Lemus Olalde

M. en C. Eduardo Rodríguez Flores

Dr. Humberto Cervantes Maceda

23 de octubre de 2009

Resumen

La propuesta que se presenta en este trabajo consiste en adaptar los procesos de desarrollo arquitectónico que propone el Instituto de Ingeniería de Software (SEI) al contexto de equipos de desarrollo pequeños con el fin de que estos se puedan emplear en las empresas de tecnologías de la información. El proceso propuesto es evaluado mediante el diseño arquitectónico de una aplicación de la vida real que emplea una variante de la Arquitectura Orientada a Servicios (SOA). Adicionalmente, se presentan las diferentes variantes de SOA encontradas en la literatura junto con sus características principales. Por último, este proyecto proporciona las guías para integrar las adaptaciones de los métodos dentro del Proceso de Software en Equipo nivel Introductorio (TSPi), explicando específicamente como se ajusta con respecto de las actividades de desarrollo de requerimientos y diseño de alto nivel.

Palabras clave: SOA, TSPi, Desarrollo Arquitectónico.

Abstract

The presented proposal in this work consists in adapting the architectural development processes that the Software Engineering Institute (SEI) suggests to the small development teams with the purpose that they can be used in the information technologies enterprises. The process presented in this work is evaluated by using the architectural design of a real-life application that utilizes a variant of the Services Oriented Architecture (SOA). In addition, different variants found on the literature of SOA and their main features are presented. Finally, this project provides guidelines for integrating the adapted methods to the Introductory Team Software Process (TSPi), specifically explaining where they fit with respect to requirements development and high-level design activities.

Keywords: SOA, TSPi, Architectural Development.

Agradecimientos

Al **Consejo Nacional de Ciencia y Tecnología (CONACyT)** y a la **Universidad Autónoma Metropolitana (UAM)** por haber otorgado el financiamiento para la realización de este proyecto de investigación.

A **José Guadalupe** y **María Joaquina**, mis padres, por confiar siempre en que las decisiones que he tomado han sido las mejores.

A **Humberto Orellana** y **Romelia Vázquez** por haberme adoptado como su nieto y criarme como su hijo.

A **Antonio**, **Grisel** y **Jorge** porqué aunque no lo sepan he sido muy afortunado de tenerlos como hermanos.

A mis tíos, primos y demás familiares porqué siempre he podido contar ellos en todo momento.

A mi asesor el **Dr. Humberto Cervantes**, por la confianza que ha depositado en mi y por su constante apoyo en la realización de este proyecto de investigación.

Al profesor **Luis Castro**, quién se tomó la molestia de aportar valiosas observaciones a este proyecto de investigación

A todos los profesores y amigos de la Maestría en Ciencias y Tecnologías de la Información por haber compartido mucho de lo que saben conmigo.

A **Ramón Charnichard** y **Gabriel Martínez**, por ser los más grandes amigos que he tenido.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

En la actualidad los sistemas de software son un componente crucial en las organizaciones, quienes requieren de software cada vez más complejo para satisfacer sus necesidades. Con el fin de hacer manejable la complejidad del software, este es divido en elementos de software, se establecen las relaciones entre ellos y se determinan las propiedades que cada elemento ofrece a los demás. Esto determina la arquitectura del software que Len Bass et al [Bass2003] define como *"la estructura o estructuras del sistema que comprenden elementos de software, sus relaciones y sus propiedades externamente visibles"*. El propósito principal de la arquitectura de software es establecer un puente entre objetivos de negocio y los sistemas de cómputo. Además, la arquitectura de software permite facilitar la comunicación entre los involucrados en el desarrollo y asignar responsabilidades a los miembros del equipo [OBrien2005].

Para determinar a los elementos de software, sus relaciones y las propiedades que integran a la arquitectura de un sistema de software se realiza un descubrimiento gradual partiendo de los objetivos de negocio de los clientes hasta que se llega al código fuente (ver Figura 1.1). Una vez que los clientes han proporcionado sus objetivos de negocio es posible determinar los *atributos de calidad* que conducirán el diseño de la arquitectura. Los atributos de calidad son elementos que permiten medir y evaluar propiedades tales como el desempeño, disponibilidad o modificabilidad de un sistema. Estos atributos de calidad, que son parte de los requerimientos del sistema, proveen un medio para expresar el grado de satisfacción de

los usuarios y desarrolladores con respecto al sistema. De acuerdo con lo que dice Len Bass et al en [Bass2003], los atributos de calidad se pueden lograr a través del uso de *tácticas arquitectónicas*, que son decisiones de diseño que permiten controlar una respuesta particular de un atributo de calidad. Las tácticas arquitectónicas, a su vez, son implementadas haciendo uso de *patrones arquitectónicos*, los cuales representan una estructura de organización fundamental para el sistema [Buschmann96]. En software, un patrón arquitectónico es una solución conceptual reutilizable a un problema recurrente en la práctica. En el patrón arquitectónico se describen los elementos que lo comprenden, sus responsabilidades, las relaciones entre los elementos del patrón y las restricciones bajo las cuales se puede emplear. La arquitectura de software comprenderá entonces a los elementos, sus relaciones y las propiedades que se deriven de todos los patrones empleados para satisfacer a los objetivos de negocio planteados por los clientes.

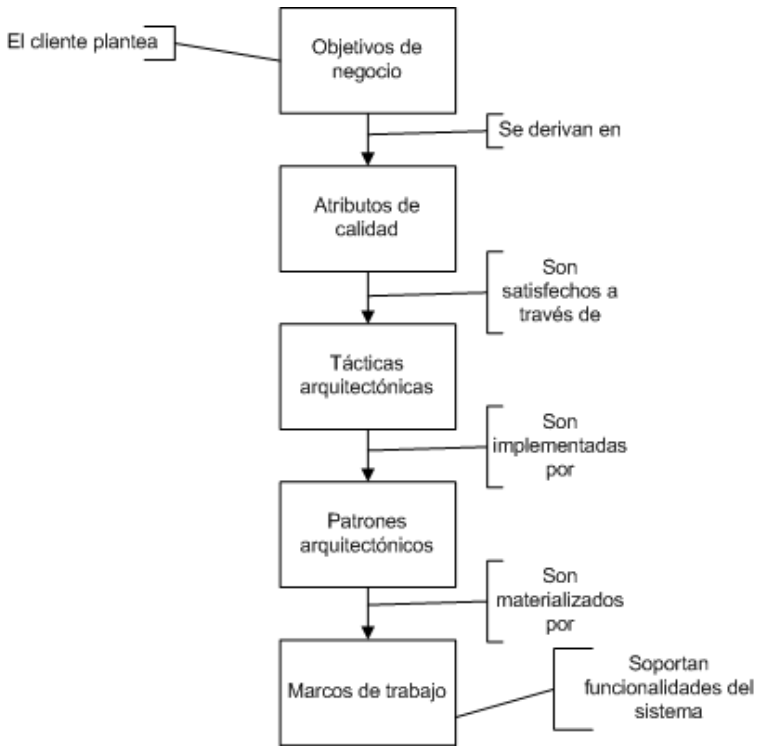


Figura 1.1: Elementos arquitectónicos que muestran la relación entre objetivos de negocio y arquitectura de software

Usualmente los arquitectos de software realizan el descubrimiento de los elementos que componen a la arquitectura basándose en su experiencia (entre muchos otros factores), pero en años recientes han surgido metodologías de desarrollo arquitectónico que permiten realizar este descubrimiento de forma más sistemática y controlada. Tal vez el esfuerzo más importante en la creación de metodologías de desarrollo arquitectónico sea el realizado por el Instituto de Ingeniería de Software (SEI), el cual propone la metodología mostrada en la Figura 1.2.

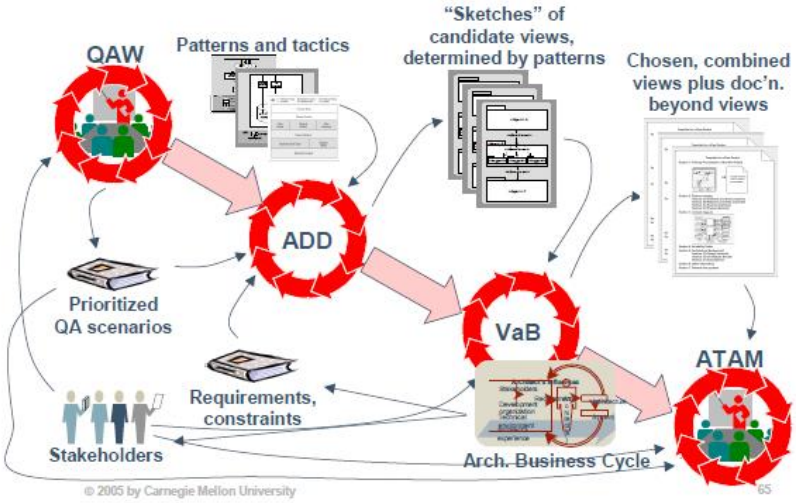


Figura 1.2: Metodología de desarrollo arquitectónico propuesta por el SEI [wwwSEI]

Esta metodología toma como base a los objetivos de negocio y contempla métodos para:

- Identificar y documentar atributos de calidad (Taller de Atributos de Calidad, QAW por sus siglas en inglés)
- Diseño de la arquitectura de software a partir de la descomposición del sistema basada en tácticas y patrones arquitectónicos que satisfacen a los atributos de calidad (Diseño Guiado por Atributos, ADD por sus siglas en inglés)
- Realizar una correcta documentación de las vistas arquitectónicas (Vistas y más allá, VaB por sus siglas en inglés). Este método es parte de la actividad de diseño pero no es una continuación.

- Evaluar el diseño propuesto contra los atributos de calidad documentados (Método de Análisis de Compromisos Arquitectónicos, ATAM por sus siglas en inglés)

1.2. Problemática

Los métodos que integran a la metodología de desarrollo arquitectónico propuesta por el SEI buscan ayudar a los arquitectos a realizar diseños arquitectónicos de forma sistemática, a documentar estos diseños y a comprender las consecuencias de las decisiones arquitectónicas que se toman con respecto de los atributos de calidad y objetivos de negocio del sistema [Nord2004]. Uno de los primeros métodos relacionados con el desarrollo arquitectónico es el método de evaluación arquitectónica llamado ATAM, el cual con el paso del tiempo ha permitido a los investigadores del SEI crear otros métodos derivados del mismo, ejemplos de estos métodos son QAW, ADD y VaB [Kazman2004].

Los métodos arquitectónicos del SEI antes mencionados tienen 3 características que dificultan su uso por equipos de desarrollo pequeños, estas características son:

1. No están integrados dentro de un proceso de desarrollo de software particular y para obtener mayor beneficio estos métodos se deben utilizar juntos, lo cual requiere de ajustes importantes (alguien en la organización de desarrollo debe tener mucho conocimiento en su uso para poder ajustarlos en un proceso cohesivo) [Lattanze2005]. Está característica es esencial ya que con los ajustes necesarios los métodos arquitectónicos del SEI se pueden usar dentro de cualquier proceso de desarrollo.

2. Los métodos arquitectónicos del SEI están pensados para que sean usados en el diseño de sistemas complejos de alto riesgo. Está característica los hace ser pesados y costosos en términos de documentación, recursos financieros, de calendario, capacitación, etc. [Lattanze2005]

3. Están orientados hacia grandes equipos tanto de desarrollo como de involucrados. Está característica se podría ver como un derivado de la anterior debido a que sólo equipos de desarrollo grandes pueden desarrollar sistemas complejos con alto riesgo.

Adicionalmente, se ha identificado que la secuencia con la cual el SEI propone que los métodos sean realizados tiene potenciales riesgos de re-trabajo en la documentación arquitectónica, esto se debe a que una vez que el diseño ha sido documentado se evalúa y en caso de que se necesiten correcciones la documentación se tiene que actualizar.

1.3. Objetivos generales y particulares del proyecto

1.3.1. Objetivos generales

El objetivo general de este proyecto de investigación es:

- Adaptar la metodología de desarrollo arquitectónico del SEI para que pueda ser empleada por equipos de desarrollo pequeños.

1.3.2. Objetivos particulares

Los objetivos particulares de este proyecto de investigación son:

- Definir las adaptaciones a los métodos que integran a la metodología de desarrollo arquitectónico del SEI para que se pueda aplicar en el contexto de equipos de desarrollo pequeños.
- Identificar un conjunto de elementos arquitectónicos que permitan realizar la metodología de desarrollo arquitectónico del SEI adaptada a equipos de desarrollo pequeños en el contexto particular de la Arquitectura Orientada a Servicios a nivel Aplicación (ASOA).
- Seleccionar un caso de estudio de la vida real que use los elementos arquitectónicos identificados para ASOA con la finalidad de evaluar las adaptaciones propuestas.
- Crear, evaluar y documentar el diseño arquitectónico del sistema.
- Integrar la metodología adaptada con el Proceso de Software en Equipo nivel Introductorio (TSPi).

1.4. Justificación

La realización de este proyecto de investigación permitirá a las organizaciones de desarrollo reducir los costos de:

- *Adaptación de los métodos arquitectónicos del SEI*: Para realizar la adaptación de los métodos que integran a la metodología de desarrollo arquitectónico del SEI las organizaciones de desarrollo tendrían que contratar a expertos en la misma o asignar a parte de su personal en la realización de esta tarea, lo que involucra un gasto en recursos financieros, humanos y de calendario (costos que pocas organizaciones pueden pagar).

- *Integración de la metodología del SEI en un proceso de desarrollo*: De igual forma que en el punto anterior, las organizaciones deberían invertir recursos para determinar en qué etapas de su proceso de desarrollo se puede integrar a la metodología de desarrollo arquitectónico del SEI y realizar las actividades necesarias para adoptar estos cambios en su proceso.

- *Entrenamiento en la metodología*: Para realizar entrenamiento se requiere que expertos en la metodología propuesta por el SEI creen el material e impartan las sesiones de entrenamiento.

El uso de algún método de desarrollo arquitectónico, como el propuesto por el SEI, permite incrementar la posibilidad de éxito (entendido como entregar sistemas de calidad, a tiempo y con los recursos acordados) en el desarrollo de sistemas de cómputo, por lo cual es necesario dotar a los equipos de desarrollo pequeños con alguno de estos métodos adaptado a sus características y necesidades propias (posibilidad de comunicación persona a persona, pocos roles involucrados, desarrollan sistemas de complejidad media a baja, se considera que desarrollan sistemas no críticos[1], etc.). Lo anterior le permitirá a estos equipos realizar diseños arquitectónicos de calidad de forma repetible.

[1]Sin pérdidas financieras o de vidas humanas

1.5. Hipótesis

El desarrollo de este proyecto se basa en las siguientes hipótesis:

- Cada uno de los métodos que integran la metodología propuesta por el SEI se pueden adaptar al contexto de equipos de desarrollo pequeños.
  - Es posible agilizar los pasos que cada uno de los métodos comprende.
  - Es posible reducir el número de artefactos o reducir el contenido de algunos de estos.
  - Es posible que arquitectos con poca experiencia mejoren sus diseños arquitectónicos siguiendo las adaptaciones propuestas.
- Los métodos propuestos por el SEI pueden trabajar de forma coordinada.
- La metodología propuesta por el SEI se puede integrar con TSPi.
- Es posible determinar los elementos arquitectónicos específicos de ASOA con la literatura y aplicaciones disponibles.
- Es posible establecer un proceso general que permita identificar los elementos arquitectónicos para patrones distintos a ASOA.

1.6. Propuesta

Se propone diseñar un proceso, resultado de la adaptación de la metodología de desarrollo arquitectónico propuesta por el SEI al contexto de equipos de desarrollo pequeños.

Dicho proceso debe ser ligero, en el sentido que durante el desarrollo arquitectónico solo se crearan un conjunto mínimo de artefactos a partir de un conjunto de actividades indispensables que conserven el *"espíritu"* de la metodología de desarrollo arquitectónico del SEI. Este proceso se debe realizar a través de las guías, plantillas y listas de verificación que se proponen en este trabajo y que están inspiradas en el Proceso de Software Personal (PSP).

Adicionalmente, estas guías fueron incorporadas al Proceso de Desarrollo en Equipo nivel Introductorio (TSPi) con la finalidad de proporcionar un marco de trabajo completo para realizar el desarrollo de cualquier sistema de software.

La adaptación propuesta en este proyecto se considerá de menor costo debido a que:

- Las organizaciones de desarrollo no invertirían (o la inversión sería mínima) en las adaptaciones, al contexto de equipos de desarrollo pequeños, de la metodología de desarrollo arquitectónico propuesta por el SEI.
- El tiempo de capacitación se reducirá debido a que en las guías propuestas se condensan de froma simple todas las activades a realizar por cada rol en cada método.
- Se producirá una menor cantidad de artefactos.

1.7. Metodología para la realización del proyecto

La metodología propuesta en este proyecto de investigación consiste de las siguientes fases:

- *Investigación:* Considera la investigación de los siguientes elementos:
  - Conceptos relevantes de arquitectura de software.
  - La relación de la arquitectura de software con la calidad del sistema.
  - Las diferentes opciones de metodologías de desarrollo arquitectónico.
  - La identificación de los niveles de aplicación de la *Arquitectura Orientada a Servicios* (SOA) disponibles en la literatura.
- *Estudio:* Comprende el estudio de las actividades que integran cada método de la metodología de desarrollo arquitectónico propuesta por el SEI.
- *Adaptación:* Se adaptan los métodos que comprende la metodología de desarrollo arquitectónico propuesta por el SEI para que se puedan aplicar en el contexto de equipos de desarrollo pequeños.

- *Realización:* Se selecciona un caso de estudio de la vida real que considere el uso de ASOA. Por último, se realiza, con ayuda de un grupo de estudiantes de maestría, el diseño arquitectónico del caso de estudio siguiendo las adaptaciones propuestas a la metodología de desarrollo arquitectónico propuesta por el SEI. El resultado de esta fase es un diseño validado y documentado que permita implementar al sistema.
- *Recopilación y reporte de resultados:* Se recopilan los resultados obtenidos y se realiza el reporte de los mismos.

1.8. Estructura del documento

El presente documento está estructurado de la siguiente forma:

- Capítulo 2: Se presentan las definiciones que permiten entender qué es la arquitectura de software, cuáles son los elementos que participan en el desarrollo arquitectónico y las metodologías de desarrollo arquitectónico encontradas en la literatura.
- Capítulo 3: Se presenta de forma breve al patrón arquitectónico *Arquitectura Orientada a Servicios* (SOA), de este patrón se presentan los diferentes niveles encontrados en la literatura y por último, se presentan los elementos arquitectónicos que dan soporte a cualquier metodología de desarrollo arquitectónico del nivel de aplicación SOA llamado en este proyecto ASOA.
- Capítulo 4: Se presentan las propuestas de adaptación a la metodología de desarrollo arquitectónico del SEI y se propone una forma de integrarlas con TSPi.
- Capítulo 5: Se describe una visión general del caso de estudio seleccionado, se describe cómo fueron realizados los métodos adaptados y se analizan los resultados obtenidos.
- Capítulo 6: Discusión crítica de los resultados obtenidos.
- Capítulo 7: Síntesis, conclusión y posibles trabajos futuros.
- Al final del documento se encuentran los apéndices y las referencias utilizadas.

Capítulo 2

Conceptos de Arquitectura de Software y Estado del Arte

2.1. Arquitectura de software

Actualmente, una gran cantidad de organizaciones requieren de sistemas de software para lograr sus objetivos de negocio. Estos sistemas de software son cruciales para el éxito de una organización, pero ¿Cómo se pueden relacionar los objetivos de negocio de una organización con las funcionalidades de un sistema de cómputo? Esta relación puede establecerse mediante la arquitectura de software (cabe aclarar que no solo la arquitectura de software soporta a los objetivos de negocio, también los requerimientos funcionales participan). La arquitectura de software y la forma en que relaciona objetivos de negocio con funcionalidades de un sistema de cómputo son temas que se discutirán en las siguientes secciones.

2.1.1. Concepto de arquitectura de software

Existen varias definiciones de arquitectura de software, tal vez la definición más aceptada de arquitectura de software es la que propone Len Bass et al en [Bass2003], donde la define como:

"La arquitectura de un programa o sistema de cómputo es la estructura o estructuras del sistema, las cuales comprenden elementos de software, sus propiedades externamente visibles

y las relaciones entre estos."

Los elementos que componen esta definición son explicados a continuación:

- La arquitectura está compuesta de elementos de software. Estos elementos pueden ser objetos, módulos, clases, procesos, funciones, unidades de código y unidades de ejecución entre otros.

- Las relaciones entre los elementos pueden ser estáticas (por ejemplo: es submódulo de, hereda de, etc.), dinámicas (por ejemplo: envía datos a, invoca a, etc.) o de asignación (por ejemplo: está instalado en, se desarrollará por, etc.). Las relaciones dinámicas son establecidas a tiempo de ejecución, mientras que las estáticas y de asignación no.

- Las propiedades externamente visibles son las características a cerca del servicio que provee un elemento, o de las características emanadas entre las interacciones de los elementos, por ejemplo, cuánto tiempo tarda en realizar cierta funcionalidad, la manera en que maneja las fallas o la forma en que se pueden compartir recursos. El que estas propiedades sean externas expresa el hecho de que los elementos encapsulan detalles de implementación.

En la arquitectura de software son definidos los elementos de software que habrán de satisfacer a los objetivos de negocio, y además sirve como soporte para implementar las funcionalidades requeridas. La arquitectura de software se determina de forma gradual partiendo de los objetivos de negocio planteados tanto por la organización cliente como por la organización de desarrollo. Como se muestra en la Figura 1.1 los objetivos de negocio se van derivando hasta llegar al código fuente del sistema, los elementos que participan en esta descomposición se explicarán a continuación.

2.1.2. Atributos de calidad

Los *atributos de calidad* son elementos que soportan la medición y evaluación de propiedades del sistema tales como modificabilidad, facilidad de pruebas, desempeño, escalabilidad, disponibilidad, etc. Estos atributos forman parte de los requerimientos del sistema y proveen un

medio para expresar el grado de satisfacción de usuarios y desarrolladores con respecto al sistema. Dado que la calidad en general es un concepto abstracto, un conjunto de atributos de calidad expresados de forma cuantificable permitirán acordar y evaluar la calidad del sistema que se está desarrollando. Len Bass et al en [Bass2003] propone el uso de *escenarios de atributos de calidad* (los cuales son frases que describen de forma precisa como el sistema responde ante un estímulo) como medio para documentar de forma cuantificable a los atributos de calidad específicos a un sistema. Una ventaja de esta técnica de documentación de atributos de calidad es que no requiere que exista una definición única para cada categoría de atributo de calidad, ya que uno o varios escenarios definen lo que es el atributo de calidad. Len Bass et al [Bass2003] sugiere que los escenarios de atributos de calidad tengan los siguientes elementos:

- *Fuente del estímulo:* Entidad que genera el estímulo (un humano, computadora, etc.).
- *Estímulo:* Condición que se debe considerar cuando llega al sistema.
- *Entorno:* Condiciones bajo las cuales ocurre el estímulo.
- *Artefacto:* Artefacto estimulado (puede ser el sistema entero o parte de él).
- *Respuesta:* Actividad realizada después del que el estímulo llega al sistema.
- *Medida de la respuesta:* Cuando la respuesta es generada ésta se debe medir con la finalidad de que el atributo se pueda probar.

En la Tabla 2.1 y en la Tabla 2.2 se muestran ejemplos de escenarios de atributos de calidad.

Adicionalmente, Len Bass et al [Bass2003] propone el uso de tablas de generación de escenarios de atributos de calidad, las cuales proveen un marco genérico e independiente de la aplicación para producir escenarios de atributos de calidad específicos al contexto de un sistema particular. La Tabla 2.3 muestra un ejemplo de tabla de generación de escenarios de modificabilidad, la cual a su vez puede producir el escenario de atributo de calidad mostrado en el ejemplo de la Tabla 2.2 para un sistema particular.

Escenario	Un agente externo sin autorización intenta ingresar al sistema mientras este se encuentra condiciones de operación normal, el sistema no permite que el agente ingrese y escribe una entrada en una bitácora.
Fuente	Un agente externo
Estímulo	Intenta ingresar al sistema
Entorno	El agente externo no tiene autorización para ingresar al sistema
Artefacto	El sistema
Respuesta	El agente externo no logra ingresar al sistema
Medida de la respuesta	El sistema no permite que el agente ingrese al sistema y registra el intento de ingreso en una bitácora

Tabla 2.1: Ejemplo de escenario de atributo de calidad de Seguridad

2.1.3. Tácticas arquitectónicas

Los atributos de calidad derivados de los objetivos de negocio son logrados a través de *tácticas arquitectónicas*, una táctica arquitectónica representa una decisión de diseño que influencia el control de una respuesta particular de un atributo de calidad [Bass2003]. Como ejemplo de táctica arquitectónica se puede mencionar la *Introducción de concurrencia*, la cual mejora al atributo de calidad de desempeño.

2.1.4. Patrones arquitectónicos

Las tácticas arquitectónicas son implementas a través de *patrones arquitectónicos*. En software, un patrón arquitectónico es una solución conceptual reusable a un problema recurrente en la práctica (similar a los patrones de diseño descritos por Gamma et al en [Gamma94]). Los patrones arquitectónicos expresan una forma de organización fundamental que describe a los elementos de software, sus responsabilidades, los tipos de relaciones entre elementos y las restricciones bajo las cuales se puede usar al patrón arquitectónico [Buschmann96].

Escenario	Durante un momento de actividad normal del sistema un usuario realiza una petición mientras otros 50 realizan peticiones aleatorias, el sistema debe responder a la petición del usuario en menos de 10 segundos.
Fuente	Un usuario
Estímulo	Realiza una petición al sistema
Entorno	Momento de actividad normal
Artefacto	El sistema
Respuesta	La petición es respondida por el sistema
Medida de la respuesta	La petición es respondida en menos de 10 segundos

Tabla 2.2: Ejemplo de escenario de atributo de calidad de Desempeño

Cada patrón arquitectónico facilita en menor o mayor grado ciertos atributos de calidad y puede darse el caso en que algunos se puedan mezclar para dar lugar a patrones híbridos [Land2002]. Cabe señalar que un patrón arquitectónico no es una arquitectura en él mismo. Existen diversos catálogos de patrones arquitectónicos en los cuales, desde la perspectiva del autor del catálogo, una estructura puede ser o no ser un patrón arquitectónico. Buschmann et al en [Buschmann2007] presenta un catálogo de patrones arquitectónicos bastante amplio, dentro del cual se pueden observar a los siguientes:

- *Capas:* Define varios conjuntos de elementos o capas, cada capa tiene responsabilidades distintas y al interior de cada capa las responsabilidades están relacionadas entre sí. Este patrón disminuye el impacto sobre las demás capas al modificar a los elementos de una.

- *Modelo-Vista-Controlador:* Se divide la interacción con el exterior del sistema en 3 tipos desacoplados de elementos (procesamiento, entrada y salida). Este patrón permite la modificación de los mecanismos de interacción con el exterior al desacoplarlos de la

Porción del escenario	Valores posibles
Fuente	Uno de un número independiente de fuentes, posiblemente desde dentro del sistema
Estímulo	Arribo de eventos periódicos; Arribo de eventos esporádicos; Arribo de eventos estocásticos
Artefacto	Sistema
Entorno	Modo normal; Modo sobre-cargado
Respuesta	Procesa el estímulo; Cambia el nivel del servicio
Medida de la respuesta	Latencia; Flujo de datos; Jitter[1]; Tasa de pérdidas

Tabla 2.3: Ejemplo de tabla de generación de escenarios de atributos de calidad de modificabilidad [Bass2003]

capa de procesamiento.

- *Microkernel:* Este patrón permite crear versiones diferentes de un sistema mediante la extensión de forma plug-and-play de un conjunto mínimo de funcionalidades llamado núcleo. Este patrón facilita la modificación en las funcionalidades de un sistema.

2.1.5. Vistas arquitectónicas

Una arquitectura de software que se basa en patrones y tácticas arquitectónicas es una entidad muy compleja como para que se pueda representar en una sola forma [Clements2002], por ello, para representarla se emplean múltiples *vistas*. Clements et al [Clements2002] definen una vista como una representación de un conjunto coherente de elementos (no todos los elementos de la arquitectura, solo un subconjunto de ellos) y las relaciones asociadas entre estos (a esto es a lo que se refiere la definición de arquitectura de software presentada anteriormente cuando habla de estructuras). Cada vista muestra un conjunto de atributos de calidad desde diferentes perspectivas, por lo que cada vista será relevante para diferentes

involucrados; por lo tanto, las vistas permiten comunicar a los involucrados las decisiones que se tomaron para realizar la arquitectura de software.

Como se mencionó, la elección de los patrones y tácticas arquitectónicas depende de los atributos de calidad que son significativos para los involucrados. Adicionalmente, la experiencia del arquitecto que diseña la arquitectura juega un papel muy importante en la elección de los patrones y tácticas arquitectónicas. La experiencia de un arquitecto le permite discriminar entre patrones y tácticas que aparentemente satisfagan en el mismo grado a los atributos de calidad, seleccionando patrones y tácticas con las que ya tenga experiencia.

Para tratar de limitar el hecho de que el desarrollo arquitectónico se ha basado principalmente en la experiencia de los arquitectos, han aparecido recientemente metodologías de desarrollo arquitectónico que permiten realizar el análisis y diseño de una arquitectura, la siguiente sección presenta algunos de los métodos propuestos para realizar estas actividades. En el contexto de este trabajo, el concepto de metodología se emplea para hacer referencia a un conjunto de métodos, el concepto de método se emplea para hacer referencia a un conjunto de pasos realizados por un grupo de personas y cuyo resultado es uno o un conjunto de artefactos.

2.2. Métodos de análisis, diseño y evaluación arquitectónica

Existen diversos métodos de análisis y diseño arquitectónico. Hofmeister et al [Hofmeister2005] dice que los métodos tienen en común 3 etapas las cuales son:

- *Análisis arquitectónico:* Los intereses y el contexto son examinados con el fin de producir requerimientos con significado arquitectónico.
- *Síntesis arquitectónica:* Se proponen alternativas mediante diseños arquitectónicos para solucionar un subconjunto de requerimientos con significado arquitectónico.
- *Evaluación arquitectónica:* Los candidatos a solución son evaluados contra el subcon-

junto de requerimientos con significado arquitectónico. La evaluación es un proceso iterativo que lleva a definir una arquitectura válida.

A continuación se presentan diversas metodologías de desarrollo arquitectónico que son vistas a la luz de las actividades definidas por Hofmeister et al en [Hofmeister2005].

2.2.1. Metodología propuesta por el SEI

En la Figura 1.2 se muestran los métodos que integran a la metodología de desarrollo arquitectónico propuesta por el SEI y la forma en cómo trabajan para apoyar en la realización del desarrollo de la arquitectura de software. En esta metodología la etapa de análisis arquitectónico corresponde a QAW (Quality Attribute Workshop), la etapa de síntesis arquitectónica corresponde a ADD (Attribute Driven Design) y VaB (Views and Beyond), y por último la etapa de evaluación arquitectónica corresponde a ATAM (Architectural Tradeoff Analysis Method).

En las siguientes secciones son descritos cada uno de los métodos que esta metodología comprende.

2.2.1.1. QAW (Taller de Atributos de Calidad)

QAW es un método usado para obtener, priorizar y refinar escenarios de atributos de calidad antes que la arquitectura sea diseñada [Barbacci2003]. Este método depende de la participación de los involucrados en el desarrollo del taller que QAW propone.

Los pasos de QAW son:

1. *Presentación del taller e introducción:* El facilitador:

   a) Describe la motivación del taller.

   b) Explica cada paso del taller.

   c) Presenta a los involucrados y su relación con el sistema.

2. *Presentación de la misión/negocio:* Un representante de la organización cliente presenta las motivaciones de negocio/misión del sistema. Adicionalmente, presenta los requerimientos funcionales a alto nivel, contexto del sistema, las restricciones y los atributos de calidad a alto nivel.

3. *Presentación del plan arquitectónico:* El arquitecto presenta los planes y estrategias para satisfacer los principales requerimientos de la organización cliente, los requerimientos técnicos y principales limitaciones (sistemas operativos, middleware, hardware, etc.). Adicionalmente, presenta diagramas de contexto, diagramas del sistema a alto nivel y otras descripciones escritas.

4. *Identificación de motivaciones arquitectónicas:* Durante los pasos 2 y 3 el facilitador crea una lista de motivaciones arquitectónicas (requerimientos de alto nivel, intereses de negocio, objetivos de negocio, atributos de calidad y restricciones de diseño) que él considere relevantes. El facilitador presenta a los involucrados su lista de motivaciones con el fin de lograr un consenso con los demás involucrados acerca de cuáles son las motivaciones arquitectónicas del sistema.

5. *Lluvia de ideas para los escenarios:* Los involucrados generan escenarios (ver sección 2.1.2) mediante un proceso de lluvia de ideas. Cada involucrado propone escenarios con respecto a sus intereses en forma round-robin. Este proceso se realiza en al menos 2 rondas y el facilitador debe asegurar que al menos un escenario sea creado por cada motivante identificada en el paso 4.

6. *Consolidación de escenarios:* El facilitador pide a los involucrados identificar que escenarios son similares, los escenarios que así lo sean son mezclados con previo acuerdo de los involucrados.

7. *Priorización de escenarios:* Cada involucrado tiene un número de votos aproximadamente igual al 30 % del número de escenarios consolidados. La votación ocurre en 2 rondas en las cuales cada involucrado emite la mitad de sus votos. Al final los escenarios son priorizados conforme al número de votos que obtuvieron.

8. *Refinamiento de escenarios:* Los escenarios más importantes son refinados con más detalle. El número de escenarios refinados dependerá del tiempo que le reste al taller. Para cada escenario el facilitador documenta los objetivos de misión/negocio afectados por el escenario, la descripción de los atributos de calidad relevantes, las preguntas sobre los escenarios que los involucrados quisieran hacer y las partes del escenario como fueron explicadas en la sección 2.1.2.

Al final de la ejecución de QAW, se dispone de:

- Lista de motivaciones arquitectónicas
- Escenarios crudos
- Lista priorizada de escenarios crudos
- Escenarios refinados

2.2.1.2. ADD (Diseño Guiado por Atributos)

ADD es un proceso de diseño basado en la descomposición recursiva del sistema. En cada iteración se eligen patrones y tácticas arquitectónicas con el fin de satisfacer a los atributos de calidad [Wojcik2006].

Los pasos de ADD son:

1. *Confirmar que existe suficiente información de los requerimientos:* Se debe asegurar que los involucrados han priorizado los escenarios de acuerdo con los objetivos de misión/negocio y que los escenarios contienen las partes explicadas en la sección 2.1.2. El arquitecto realiza el diseño de la arquitectura en orden descendente de la priorización de escenarios.

2. *Elegir un elemento del sistema a descomponer:* Se elige un elemento del sistema sobre el cual se enfocarán lo siguientes pasos, para la primera iteración el elemento a descomponer es el sistema completo. En iteraciones subsecuentes el sistema está dividido en 2

o más elementos, la elección de cuál de ellos elegir se puede basar en la experiencia del arquitecto, riesgos, dificultad y criterios de negocio u organizacionales

3. *Identificar candidatos a intereses arquitectónicos:* En este paso los escenarios son evaluados con respecto al impacto que tienen sobre la arquitectura. Para evaluarlos es necesario considerar si el escenario generará nuevos elementos arquitectónicos. Una vez que la priorización ha terminado, los escenarios están priorizados por los involucrados y por su impacto sobre la arquitectura, de esta priorización sólo 5 o 6 de los escenarios de mayor prioridad son considerados para continuar con el método.

4. *Elegir conceptos de diseño:* En este paso se eligen los patrones y tácticas arquitectónicas que satisfacen a los escenarios producidos por el paso 3 y las restricciones de diseño seleccionadas.

5. *Instanciar módulos y asignarles responsabilidades:* En este paso se crean nuevos elementos y a cada uno, según su tipo, se le asigna una responsabilidad. Cada responsabilidad asignada al módulo padre se debe localizar en alguno de sus hijos.

6. *Definir interfaces para los módulos instanciados:* En este paso se definen los servicios y propiedades que el módulo requiere o pública. Las interfaces deben incluir la sintaxis de las operaciones (firma), semántica de las operaciones (pre y post condiciones), información de intercambio (eventos, datos globales), calidad de servicio y manejo de errores.

7. *Verificar y refinar los requerimientos y limitarlos a los módulos instanciados:* Se debe verificar que los requerimientos funcionales, atributos de calidad y limitaciones de diseño asignadas a un módulo padre estén asignadas a uno o más de sus hijos. Se deben refinar los atributos de calidad de algún módulo hijo en caso de que sea necesario.

8. Repetir de los pasos 2 a 7 para el siguiente elemento del sistema a descomponer.

El diseño resultante de ADD es documentado empleando VaB.

2.2.1.3. VaB (Vistas y más allá)

VaB permite documentar las vistas relevantes y adicionalmente, les agrega información para mantenerlas relacionadas. En cuanto al número y tipo de vistas este método no impone una restricción [Clements2002].

VaB divide el problema de crear la documentación arquitectónica en los siguientes subproblemas:

- *Elegir las vistas relevantes:* Se eligen las vistas que se crearán considerando las necesidades de información de cada involucrado. Las vistas en VaB pueden ser de 3 tipos:
  - *Módulos:* En esta vista los elementos son unidades de implementación o módulos que proporcionan una unidad de funcionalidad coherente y cuyas propiedades consisten del nombre del módulo, sus responsabilidades e información de implementación. Las relaciones en esta vista son de la forma "es submódulo de", "depende de" y "es un". Ejemplos de vistas: vista de uso, de capas, de clase/generalización.
  - *Componente y conector:* Esta vista consiste de componentes a tiempo de ejecución (unidades de procesamientos, almacenes de datos, etc.) y conectores entre estos. Cada componente debe describir su nombre, funcionalidad general y número y tipo de puertos; los conectores deben describir su nombre, naturaleza de interacción y nombre y tipo de roles. Las relaciones en esta vista son de la forma "adjunto a" (el componente en el puerto X esta adjunto al conector con rol R). Ejemplos de vistas: vista de procesos, concurrencia, repositorio y cliente/servidor.
  - *Asignación:* Esta vista consiste de elementos de software y su relación con elementos en entornos externos en donde el software es creado y/o ejecutado. Las relaciones en esta vista son de la forma "asignado a" (el elemento X es asignado al equipo E para que sea desarrollado). Ejemplo de vistas: vista de implantación, implementación y asignación de trabajo.
- *Documentar cada vista:* VaB propone una plantilla (ver Figura 2.1), explicada a continuación, para realizar la documentación arquitectónica:

  - *Guía de la documentación:* En esta sección se muestra cómo la vista está organizada, los escenarios empleados para crear la vista, dónde se pueden localizar los escenarios y para qué involucrados está dirigida la vista.
  - *Plantilla de vista:* En esta sección se encuentran el nombre de la vista, su descripción, elementos y sus relaciones (un gráfico o texto), catálogo de elementos (para cada elemento se describen sus relaciones con otros, interfaces, comportamiento y limitaciones), diagrama de contexto, guía de variación (describe puntos de variación y los mecanismos para realizar las variaciones) y los argumentos de diseño que guiaron la creación de la vista.
  - *Descripción general del sistema:* Describe al sistema, su propósito y su funcionalidad.
  - *Mapeo entre vistas:* Aunque las vistas proporcionan diferentes perspectivas del sistema no son independientes unas de otras, por lo que es útil colocar información que describe la relación que la vista guarda con alguna otra.
  - *Directorio:* Índice que muestra cada elemento, relación y propiedad definida y usada.
  - *Glosario y acrónimos:* Provee una lista de definiciones de términos y acrónimos usados en el documento.

- *Documentar la información que aplica a más de una vista:* Ésta información puede incluir:
  - La forma en que se relacionan las vistas
  - Contexto del sistema
  - Limitaciones externas, etc.

VaB no limita a un conjunto de vistas, sugiere que sean creadas las vistas necesarias para satisfacer las necesidades de comunicación y entendimiento de los diferentes involucrados en el sistema.

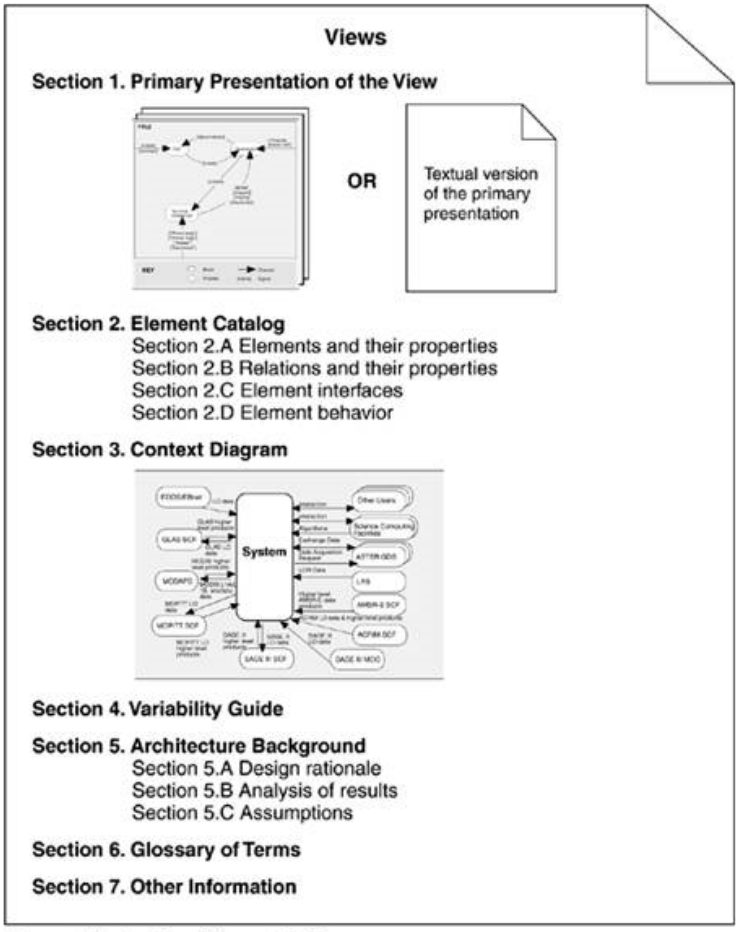


Figura 2.1: Plantilla de documentación arquitectónica usada en VaB [Bass2003]

2.2.1.4. ATAM (Método de Análisis de Compromisos Arquitectónicos)

El propósito de ATAM es evaluar las consecuencias de las decisiones arquitectónicas a la luz de los atributos de calidad y objetivos de negocio [Kazman2000]. A través de este método se puede juzgar que tan apropiada es una arquitectura. ATAM es realizado por un equipo de evaluación externo a la organización de desarrollo (un pequeño grupo de involucrados técnicos) y permite descubrir riesgos, no riesgos, compromisos entre atributos de calidad y puntos sensibles (estos conceptos se explicarán más adelante); pero no realiza un análisis preciso.

Los pasos en ATAM son:

Fase 1: Presentación

1. *Presentación de ATAM:* El equipo de evaluación realiza una presentación del método, las técnicas que emplea y los resultados que genera.

2. *Presentación de las motivaciones de negocio:* El administrador del proyecto describe las motivaciones de negocio para desarrollar el sistema, estas motivaciones incluyen: contexto del sistema, requerimientos funcionales a alto nivel y atributos de calidad a alto nivel.

3. *Arquitectura actual:* El arquitecto describe la arquitectura propuesta haciendo énfasis en cómo cubre con las motivaciones de negocio. Los demás miembros anotan lo que les parezcan riesgos del diseño presentado.

Fase 2: Investigación y análisis

4. *Identificar enfoques arquitectónicos:* Se identifican los patrones arquitectónicos predominantes. Los miembros del equipo identifican los lugares que consideran clave para el cumplimiento de los atributos de calidad.

5. *Generación del árbol de utilidad:* Los atributos de calidad más importantes son identificados, priorizados y refinados mediante la construcción de un árbol de utilidad. La Figura 2.2 muestra un ejemplo de árbol de utilidad. El árbol inicia siempre con la palabra Utilidad, en el siguiente nivel se sitúan los atributos de calidad más relevantes, debajo de cada atributo se sitúan las tácticas arquitectónicas y por último los escenarios de atributos de calidad, las letras entre paréntesis significan la priorización realizada por los involucrados y el impacto sobre la arquitectura. Estas letras pueden tomar los valores alto, medio y bajo (A, M, B).

6. *Analizar enfoques arquitectónicos:* El equipo de evaluación, basándose en los escenarios de alta prioridad, analiza y registra para cada patrón arquitectónico candidato identificado en el paso 4:

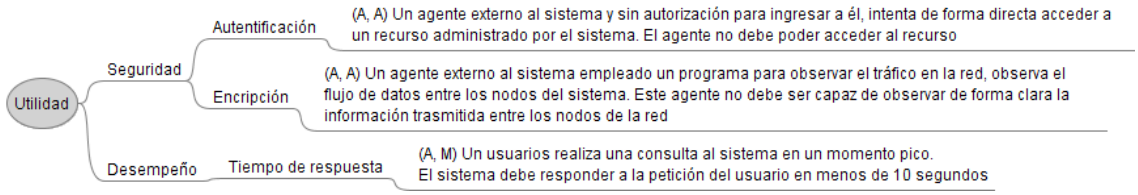


Figura 2.2: Ejemplo de árbol de utilidad

a) *Lista de riesgos:* Decisiones de diseño potencialmente problemáticas.

b) *Lista de no riesgos:* Buenas decisiones de diseño.

c) *Puntos sensibles:* Propiedades (de uno o más componentes) críticas para lograr una respuesta particular de un atributo de calidad. Ejemplo: El número de reintentos sobre una petición fallida afecta al tiempo de respuesta.

d) *Compromisos:* Propiedad que afecta a más de un atributo de calidad. Ejemplo: La liberación automática de memoria facilita el trabajo de los desarrolladores pero disminuye el desempeño del sistema.

Fase 3: Pruebas

7. *Lluvia de ideas y priorización de escenarios:* Basándose en todos los escenarios del árbol de utilidad, los miembros del equipo de pruebas realizan una votación para priorizar a los escenarios que consideren más importantes. El número de votos de cada miembro es igual al 30 % del número total de escenarios en el árbol de utilidad. En este paso es cuando en el árbol de utilidad se colocan los valores dentro del paréntesis en la coordenada “Impacto a la arquitectura”.

8. *Analizar enfoques arquitectónicos:* Este paso continúa con el análisis del paso 6, pero sólo los escenarios de más alta prioridad son considerados casos de prueba para el análisis arquitectónico. Estos casos de prueba pueden descubrir patrones arquitectónicos adicionales, riesgos, puntos sensibles y compromisos, los cuales se deben documentar.

Fase 4: Reportes

9. *Presentar resultados:* Reporte de las salidas de ATAM, éstas salidas consisten de: patrones arquitectónicos, árbol de utilidad, escenarios priorizados, riesgos, no riesgos, puntos sensibles, compromisos y temas de riesgos. Las salidas son presentadas a todos los involucrados.

2.2.2. Otras metodologías de análisis, síntesis y evaluación arquitectónica

Hofmeister et al [Hofmeister2005], además de mostrar parte de los métodos del SEI, presenta otros métodos que permiten realizar análisis, síntesis y evaluación arquitectónica. La Tabla 2.4, presenta una síntesis de los métodos encontrados en la fuente antes citada y su contraste con la metodología del SEI.

Método vs Etapa	**Análisis**	**Síntesis**	**Evaluación**
Metodología del SEI	Elegir un elemento a descomponer. Identificar candidatos a motivaciones arquitectónicas (QAW)	Elegir tácticas y patrones arquitectónicos. Instanciar módulos (asignar responsabilidad e interfaz). Tipos de vistas: Módulo, C&C y Asignación (ADD y VaB)	Verificar y refinar requerimientos (ATAM)
RUP	Casos de uso con significado arquitectónico	Vistas: Lógica, Implantación, Implementación, Proceso. Construir prototipos arquitectónicos (4 + 1 vistas)	Los prototipos permiten realizar la evaluación

Método vs Etapa	**Análisis**	**Síntesis**	**Evaluación**
Proceso de Siemens	Análisis global (tecnología existente, cualidades deseadas, restricciones organizacionales) Retos Arquitectónicos	Selección de estrategias que satisfagan los retos arquitectónicos y determinación de elementos de las vistas módulo, conceptual, ejecución y código	Evaluación global y arquitectónica
BAPO[2]	Identifica Elementos de negocio, proceso y organización	Creación de vistas cliente, aplicación, funcional, conceptual y realización	Vistas analizadas contra contexto y atributos de calidad
ASC [3]	Identificar ASR[4], clasificarlos en los segmentos del ciclo de transformación de software[5], esto permite identificar Problemas Semiseparables	Iterativamente resolver problemas en los segmentos e integrar soluciones de otros segmentos	Las decisiones son evaluadas contra los requerimientos, esto puede realizarse mediante prototipos, simulaciones, etc.

Tabla 2.4: Comparación entre métodos de desarrollo arquitectónico que comprenden análisis, síntesis y evaluación

[2]Proceso y Organización de Arquitecturas de Negocio (de sus siglas en inglés). Este método es empleado en el contexto de Líneas de Producto de Software

[3]Separación Arquitectónica de Intereses (de sus siglas en inglés)

[4]ASR: Atributos con Significado Arquitectónico (de sus siglas en inglés).

[5]Las etapas del ciclo de transformación del software son: diseño, construcción, actualización, carga y ejecución.

De las metodologías presentadas anteriormente se debe notar que la metodología de desarrollo arquitectónico propuesta por el SEI contempla métodos específicos para la realización de cada actividad en el desarrollo arquitectónico (análisis, síntesis, documentación y evaluación). Estos métodos ofrecen los siguientes beneficios en contraste con las otras metodologías:

- La obtención, documentación y priorización de los atributos de calidad se realiza junto con los clientes.
- En la identificación de elementos arquitectónicos se proponen guías para el descubrimiento y elección de los mismos.
- No limita a un cierto número de vistas como lo que propone Kruchten en su modelo 4 + 1 vistas [Kruchten95].
- La evaluación del diseño arquitectónico es realizada por un equipo de expertos externo a la organización de desarrollo.
- La metodología propuesta por el SEI es general, es decir, se creó para satisfacer las necesidades de varios tipos de organizaciones de desarrollo (al contrario de la metodología de Siemens).
- La metodología propuesta por el SEI ha sido probado en varios entornos con resultados favorables (entregas a tiempo, dentro del presupuesto y con la calidad acordada).

Por las razones previamente comentadas, para la realización de este proyecto, fue elegida la metodología de desarrollo arquitectónico propuesta por el SEI para que sea adaptada al contexto de equipos de desarrollo pequeños.

Capítulo 3

Elementos Arquitectónicos para ASOA

En la Figura 1.1 se muestra que la transición entre objetivos de negocio y el sistema pasa por una serie de elementos arquitectónicos (categorías y tablas de generación de escenarios de atributos de calidad, tácticas, patrones y marcos de trabajo). Dichos elementos arquitectónicos varían entre dominios de aplicación y antes de iniciar el desarrollo de una arquitectura es necesario realizar una identificación de los elementos que se utilizan en un dominio particular. En este capítulo se presentan los elementos arquitectónicos para una variante de SOA la cual fue llamada en este trabajo Arquitectura Orientada a Servicios a nivel Aplicación (ASOA), que es una técnica (explicada a detalle más adelante en este capítulo) que ayuda a desarrollar aplicaciones en las cuales los componentes se conectan siguiendo un enfoque de orientación a servicios, esto permite, entre otras cosas, soportar cambios dinámicos en su estructura. Ésta técnica se presenta en el contexto de este proyecto debido a que el caso de estudio seleccionado se basa en este enfoque.

3.1. Arquitectura Orientada a Servicios (SOA)

SOA ha sido una guía muy popular en las organizaciones por su facilidad de adaptación a los cambios de requerimientos. Sin embargo, esta cualidad de SOA afecta a otras áreas relacionadas con la calidad del sistema a desarrollar [OBrien2005]. Antes de entrar al detalle

de los efectos de SOA en la calidad del software, en la siguiente sección se introducirán los conceptos básicos de SOA.

3.1.1. Conceptos básicos y entidades de SOA

Aún no existe una definición clara y totalmente aceptada de SOA, pero, para el contexto de este proyecto se considerarán las expuestas por la W3C y Jammes et al, donde definen SOA como:

- *"Un conjunto de componentes que se pueden invocar y cuyas descripciones de interfaces se pueden publicar y descubrir"*. [W3C2004]
- *"Un conjunto de principios arquitectónicos para construir sistemas autónomos pero interoperantes"*. [Jammes2005]

Las definiciones anteriores involucran varios elementos, por un lado, se dice que SOA está integrado por componentes que se describen de alguna manera y que estas descripciones se pueden publicar y descubrir en algún lugar; por otro lado, se habla de sistemas autónomos que pueden interoperar entre sí. Las preguntas que surgen de inmediato por estas definiciones son: ¿Cuáles son esos componentes?, ¿Qué es lo que se describe de los componentes?, ¿En dónde se publican y descubren las descripciones de los componentes?, ¿Por qué se dice que los componentes son autónomos pero interoperantes?, ¿Cómo son invocados los componentes? Las preguntas anteriores se responden a continuación. Como se mencionó, no existe un acuerdo sobre cómo definir a SOA, pero si existe acuerdo para determinar que los componentes de SOA son denominados *Servicios*, estos representan funcionalidades compartidas y reusables de una aplicación. Según la definición de Buschmann et al [Buschmann96] sobre patrones arquitectónicos, SOA podría verse como un patrón arquitectónico debido a que describe un conjunto de elementos arquitectónicos (Servicios), sus relaciones, sus responsabilidades y establece las restricciones bajo las cuales se pueden usar.

La Figura 3.1 muestra los elementos del patrón SOA y la forma en que interactúan, a continuación se explican estos elementos:

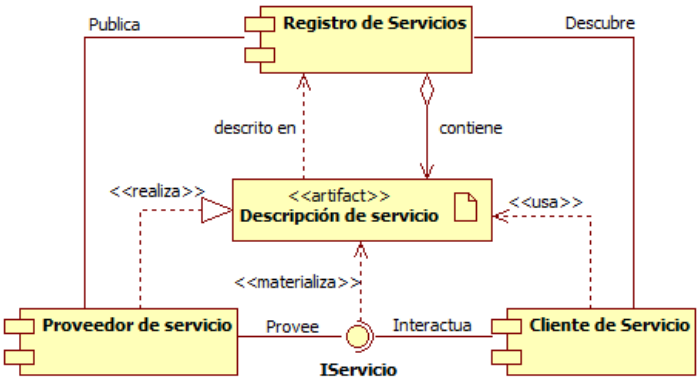


Figura 3.1: Elementos de SOA y el patrón arquitectónico en el cual participan

- *Descriptor de servicio:* Específica la forma de interactuar entre el cliente y el proveedor del servicio proporcionando información acerca de la sintaxis, semántica y aspectos de calidad del servicio. Esta descripción representa un contrato entre el cliente y el proveedor mediante el cual el proveedor se compromete a cubrir lo que se específica en ésta descripción. Este contrato puede tener una caducidad asociada.

- *Registro de servicios:* Entidad que almacena descripciones de servicio y permite que se realicen búsquedas sobre las descripciones que almacena.

- *Cliente de servicio:* Entidad que requiere de las funcionalidades provistas por algún servicio y realiza búsquedas en el registro.

- *Interfaz de servicio:* Punto a través del cual un cliente puede usar las funcionalidades de un proveedor. Esta interfaz materializa al descriptor de servicio.

- *Proveedor de servicio:* Entidad que provee un servicio mediante la implementación de una interfaz de servicio. El proveedor de servicios se encarga, además, de publicar en el registro aquellos servicios que ofrece.

En la siguiente sección se muestran las motivaciones que pueden orillar a una organización a emplear SOA.

3.1.2. Motivaciones para usar SOA

Los objetivos de negocio permean hasta los atributos de calidad, los cuales limitan la elección de un patrón arquitectónico en específico [OBrien2005]. La relación entre objetivos de negocio, atributos de calidad y patrones arquitectónicos se debe a que estos últimos se enfocan en satisfacer en mayor o menor grado ciertos atributos de calidad y los atributos de calidad son derivados de los objetivos de negocio. Por mencionar un ejemplo de lo dicho anteriormente, supóngase que una organización tiene el objetivo de adaptarse rápidamente a los cambios en las necesidades de sus clientes, esta meta organizacional implica:

- Modificabilidad: Facilidad para realizar modificaciones.
- Extensibilidad: Facilidad para que los sistemas/componentes se puedan extender.
- Escalabilidad: Asegurar que el sistema/componente escale ante una mayor demanda por parte de clientes nuevos o habituales.

En el contexto de este trabajo interesa particularmente cómo es que SOA impacta a los atributos de calidad, por este motivo la siguiente sección tratará este tema proporcionando una guía para determinar si SOA es un patrón arquitectónico adecuado para lograr los objetivos de negocio identificados por la organización de desarrollo y la organización cliente.

3.1.3. Impacto de SOA en los atributos de calidad

O'Brien et al [OBrien2005] muestra una lista completa de como SOA impacta a los atributos de calidad, por lo que esta sección se limitará a mostrar una síntesis del trabajo antes mencionado. La Tabla 3.1 muestra una síntesis, de acuerdo al autor, de los tipos de atributos de calidad y sus definiciones y la Tabla 3.2 muestra el impacto de SOA en los atributos de calidad.

Atributo de calidad	**Definición**
Interoperabilidad	Capacidad de un conjunto de entidades que pueden comunicarse para compartir información específica y operar de acuerdo con una semántica operacional acordada.
Confiabilidad	Capacidad de un sistema para mantenerse operativo con el tiempo.
Disponibilidad	Grado en el cual un sistema o componente es operacional y accesible cuando es requerido para su uso.
Usabilidad	Medida de la calidad de experiencia de usuario al interactuar con información o servicios.
Seguridad	En general es asociada a 3 principios: Confidencialidad: El acceso a información o servicios sólo se concede a sujetos autorizados. Autenticidad: Se puede confiar en que el autor/enviador indicado es el responsable de la información. Integridad: Información no corrupta.
Desempeño	En general está relacionado con cuánto tiempo le toma al servicio procesar una petición.
Escalabilidad	Capacidad de un sistema de funcionar sin degradar alguno de sus requerimientos de calidad cuando el sistema cambia en tamaño o volumen para satisfacer las necesidades de los usuarios.
Extensibilidad	La facilidad con la que las capacidades del sistema se pueden extender sin afectar a otros sistemas o partes del mismo.
Adaptabilidad	Facilidad con la que un sistema se puede modificar para ajustarse a cambios de requerimientos.
Facilidad de prueba	El grado en el cual un sistema o componente facilita el establecimiento de criterios de prueba y la realización de pruebas que determinen si los criterios se han cubierto.

Atributo de calidad	**Definición**
Facilidad para realizar auditorias	Representa el grado en que un sistema o componente mantiene registros suficientemente adecuados para soportar una o más auditorías legales o financieras específicas.
Operabilidad y facilidad de implantación	La operabilidad se refiere a la capacidad del sistema para permitirle al usuario operarlo y controlarlo. La facilidad de implantación se refiere a la facilidad con la que el sistema puede ser implantado en el entorno de producción.
Modificabilidad	Capacidad para hacer cambios a un sistema de forma rápida y rentable.

Tabla 3.1: Tipos de atributos de calidad y sus definiciones, según [OBrien2005]

Atributo de calidad	**Impacto de SOA**
Interoperabilidad	Impacto positivo. SOA le permite interactuar a aplicaciones y servicios creados en diferentes lenguajes de programación que posiblemente han sido implantados en plataformas distintas.
Confiabilidad	Falta mayor investigación sobre la forma en que SOA impacta a este atributo de calidad. Pueden ocurrir problemas al intercambiar mensajes si los servicios están distribuidos en una red.
Disponibilidad	Falta mayor investigación sobre la forma en que SOA impacta a este atributo de calidad. Se debe ofrecer alguna forma de monitoreo de la disponibilidad del servicio y proporcionar mecanismos de contingencia cuando los servicios ya no sean disponibles.

Atributo de calidad	**Impacto de SOA**
Usabilidad	Falta mayor investigación sobre la forma en que SOA impacta a este atributo de calidad. La usabilidad se disminuye si se soportan interacciones humanas complicadas (interacciones complicadas implican mayor comunicación entre servicios o mayor procesamiento).
Seguridad	El impacto de SOA en general es negativo, sobre todo si se requiere comunicación segura entre servicios distribuidos en una red.
Desempeño	El impacto de SOA puede ser negativo si los servicios se encuentran distribuidos en una red, esto debido a los retardos propios de la red.
Escalabilidad	En general el impacto de SOA es positivo, esto debido a que los clientes de servicio emplean al servicio hasta que lo encuentran en el registro, por lo que se podrían crear varias réplicas del servicio para que sean atendidas varias peticiones a la vez.
Extensibilidad	Impacto positivo. SOA soporta que fácilmente se agreguen nuevas funcionalidades o servicios. Se debe tener cuidado al diseñar las interfaces y las descripciones dado que un cambio en estas podría disminuir el impacto positivo de SOA en este atributo de calidad.
Adaptabilidad	Si las adaptaciones son manejadas apropiadamente, SOA puede tener un impacto positivo debido a que los servicios son descubiertos a tiempo de ejecución y a que un servicio se puede ligar a otro si la descripción del segundo satisface las necesidades del primero.
Facilidad de prueba	SOA tiene en general un impacto negativo en este atributo de calidad. Probar un sistema que emplea a SOA puede ser complejo, en especial si los servicios están distribuidos en una red o si no se dispone del código fuente de algún servicio.

Atributo de calidad	**Impacto de SOA**
Facilidad para realizar auditorias	El impacto de SOA en general es negativo, esto se debe a que en una composición de servicios el control de la ejecución es cedido a otro servicio y el primero no puede recolectar fácilmente los rastros necesarios para que las auditorias se realicen.
Operabilidad y facilidad de implantación	El impacto de SOA es positivo, pero hay que considerar que los entornos de ejecución de los servicios pueden ser heterogéneos, lo que puede dificultar la facilidad de administrarlos.
Modificabilidad	El impacto de SOA es positivo dado que SOA soporta la modificación de los servicios o de las aplicaciones que usan servicios, pero las interfaces de los servicios deben ser diseñadas cuidadosamente debido a que puede ser muy difícil evaluar el impacto del cambio, en especial en servicios disponibles en Internet.

Tabla 3.2: Impacto de SOA en los atributos de calidad, según [OBrien2005]

3.2. Niveles de aplicación de SOA

Gran parte de la literatura consultada para realizar este estudio ubica a SOA a un nivel organizacional (en particular con respecto a los Servicios Web), sin embargo, este no es el único contexto en donde SOA se puede aplicar o aprovechar. Para este estudio se identifican 3 niveles de aplicación SOA:

- ESOA (SOA a nivel Empresarial)
- EmSOA (SOA a nivel Embebido)
- ASOA (SOA a nivel Aplicación)

Las siguientes secciones definen las características de los niveles de aplicación mencionados. Cabe aclarar que ESOA y EmSOA sólo son presentados de forma breve pues no serán utilizados en el resto de este documento.

3.2.1. SOA a nivel Empresarial (ESOA)

SOA empresarial es una versión orientada a negocio de SOA. Establece planes detallados para desarrollar soluciones de negocio flexibles, adaptables, abiertas y basadas en servicios; esto ayuda a crear soluciones de negocio innovadoras que satisfacen las necesidades de organizaciones que se desenvuelven en entornos donde los requerimientos cambian constantemente. Las soluciones le permiten a las organizaciones afrontar posibles amenazas competitivas o atraer a nuevos clientes. Cada organización tiene objetivos diferentes, O'Brien et al [OBrien2005] menciona cuáles son los más comunes y se explica cómo SOA permite alcanzarlos.

3.2.2. SOA a nivel Embebido (EmSOA)

SOA embebido traslada la idea de SOA a nivel de dispositivos de hardware [Barisic2007]. Estos dispositivos pueden ser de tamaño mediano como ruteadores o PDA's, o bien pequeños como sensores. Algunos de estos dispositivos no logran participar en entornos SOA de forma directa (almacenando y ejecutando un servicio en su memoria), por lo que en algunos casos necesitan hacer uso de intermediarios para subcontratar carga computacional de otros dispositivos con mayores recursos. Algunas de las motivaciones para considerar a SOA como patrón arquitectónico en este contexto se mencionan en [Barisic2007] y a continuación se presentan de forma breve:

- *Hacer homogéneos entornos heterogéneos:* En redes de sensores, por ejemplo, los sensores pueden ser de diferentes fabricantes. Asignar un servicio estándar a cada sensor permite manejarlos de forma homogénea sin importar quién lo fabricó.

- *Manejar el comportamiento dinámico:* Las redes de dispositivos son típicamente dinámicas (dispositivos llegan y se van a tiempo de ejecución), SOA permite administrar el

comportamiento dinámico de forma transparente a los desarrolladores mediante el uso de middlewares.

- En SOA embebido los servicios son considerados entidades físicas autónomas, más que entidades funcionales. Las descripciones de estos servicios involucran propiedades tales como nombre del fabricante, versión, número de serie, etc.

3.2.3. SOA a nivel Aplicación (ASOA)

En el contexto de ASOA los componentes de software son los proveedores y clientes de servicios que se conectan entre ellos a tiempo de ejecución, usando el patrón SOA. Este trabajo entiende a los componentes de software según la definición de Szyperski [Szyperski98], donde los define como:

"Unidades binarias de composición con interfaces específicas y un contexto de dependencias explícito. Un componente de software se puede implantar independientemente y está sujeto a composición por terceras partes".

En ASOA se identifican 2 elementos relacionados con la descripción de los servicios (componentes) como se ejemplifica en la Figura 3.2:

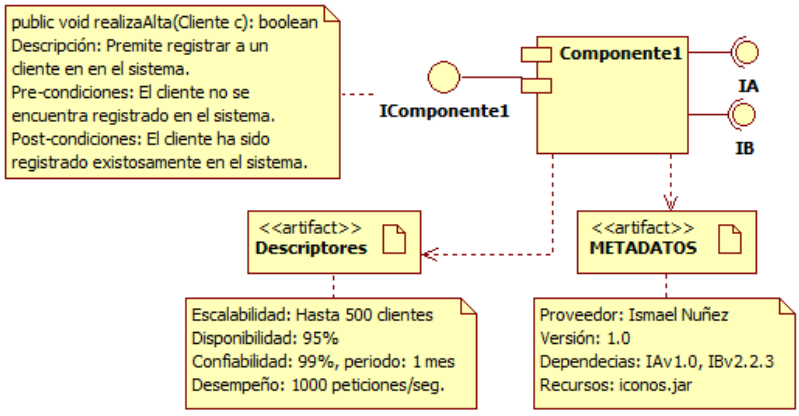


Figura 3.2: Perspectivas de la descripción de servicios

- Elementos que el cliente de servicio requiere conocer del proveedor.

- *Interfaz de servicio:* Incluye la firma y descripción de los métodos de servicio y sus pre y post condiciones.
- *Propiedades de servicio:* Éstas propiedades pertenecen al componente que provee el servicio y pueden abarcar aspectos relacionados con la calidad (escalabilidad, disponibilidad, confiabilidad, extensibilidad y desempeño) o bien atributos que permitan diferenciar a los proveedores de un servicio.

- Elementos que el proveedor de servicio requiere para proveer sus servicios.
  - Implementación del servicio, dependencias hacia otros servicios, multiplicidad de las dependencias, la versión del servicio del cual depende, la versión del servicio proveído y otros recursos de los cuales dependa el correcto funcionamiento del servicio (semántica expresada en metadatos).
  - Aspectos relacionados con la calidad.

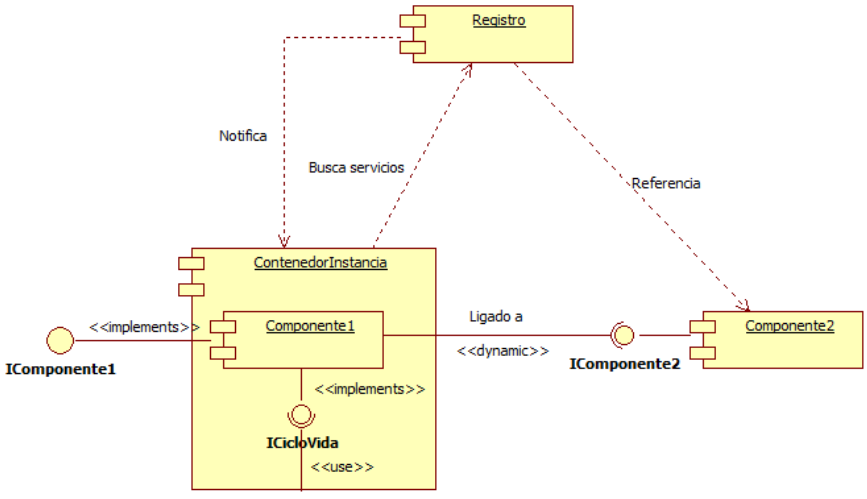


Figura 3.3: Vista a tiempo de ejecución de los componentes que soportan el comportamiento dinámico en ASOA

En ASOA los componentes de software exhiben un comportamiento dinámico, es decir, pueden llegar y partir en cualquier momento de la ejecución de la aplicación, este compor-

tamiento se debe soportar de alguna manera, por lo cual se introducen los conceptos de orientación a servicios como se puede observar en la Figura 3.3.

Elemento de comparación	**Descripción**
Nivel de abstracción	Interempresarial, intraempresarial, componentes de software o hardware, integraciones de aplicaciones, etc.
Concepto de servicio	Interpretación que cada nivel de aplicación proporciona del concepto de servicio
Elementos de la descripción del servicio	Elementos sintácticos, semánticos y de calidad de servicio que sirven para describir al servicio
Motivaciones	Requerimientos propios de la organización que la motivan a elegir un determinado nivel de aplicación SOA
Madurez de la tecnología	Estado de las herramientas, plataformas tecnológicas y estándares que apoyan el desarrollo de aplicaciones referentes al nivel de aplicación SOA
Interacciones entre servicios	Formas en que los servicios interactúan
Granularidad de la interfaz de servicio	Nivel de funcionalidades del servicio exportadas a través de su interfaz
Naturaleza de los eventos	Fuente y periodicidad con la que se producen los eventos de comunicación entre servicios o bien entre servicios y entidades externas

Tabla 3.3: Elementos de comparación entre los niveles de aplicación de SOA y sus definiciones

Con la finalidad de hacer más claras las diferencias entre los diferentes niveles de aplicación de SOA, en este estudio se proponen los elementos de comparación mostrados en la Tabla

3.3; posteriormente, en la Tabla 3.4 se muestran la síntesis de la comparación de los niveles de aplicación de SOA en el marco de los elementos de comparación propuestos.

Elem. de comp. Vs. Nivel de Aplic.	**ESOA**	**EmSOA**	**ASOA**
Nivel de abstracción	Inter/Intra empresarial	Dispositivos de HW	Componentes de SW
Concepto de servicio	Proceso/actividad de negocio	Funcionalidad del dispositivo	Funcionalidad del componente
Motivaciones	Tiempo de acceso al mercado, adaptabilidad, modificabilidad, disminución de costo de actividades del proceso de negocio (subcontratación)	Manejo de heterogeneidad de dispositivos, escalabilidad, facilidad de integración, robustez	Integración continua, facilidad de sustituir elementos de software de manera fácil y dinámica y extender las funcionalidades de la aplicación.
Madurez de la tecnología	Muy madura (Servicios Web y CORBA). Existen varios estándares ya maduros y otros en proceso de maduración	Falta más investigación en lo especial en dispositivos muy pequeños (p.e. sensores). Existen varios middlewares y plataformas enfocadas en este nivel de SOA como: Jini, DPWS, UPnP y OSGi	Existen varias plataformas como: Eclipse, OSGi/DS y COM
Interacciones entre servicios	Composiciones, orquestaciones, coreografías	Integración de aplicaciones	Realizadas a través de composiciones expresadas mediante las dependencias hacia otros componentes

Elem. de comp. Vs. Nivel de Aplic.	**ESOA**	**EmSOA**	**ASOA**
Granularidad de la interfaz de servicio	Grano grueso (actividades dentro de procesos de negocio, procesos de negocio completos, etc.)	Grano fino (nivel dispositivo)	Grano medio (funcionalidades de componentes de SW)
Naturaleza de los eventos	Socios de negocio, clientes. La periodicidad de los eventos depende de la popularidad del servicio	Sistemas de monitoreo, otros dispositivos. Los eventos pueden ser periódicos o bajo demanda	Entorno de ejecución, otros componentes de SW
Elementos de la descripción del servicio (semántica, calidad de servicio y sintáctica)	Nombre de la organización, tipo de servicio, taxonomía	Nombre del fabricante, versión, número de serie, etc.	Elementos que el cliente de servicio requiere conocer del proveedor. - Firma y descripción de los métodos de servicio y sus pre y post condiciones. - Propiedades del componente que provee el servicio. Pueden ser aspectos relacionados con la calidad o atributos que permitan diferenciar a los proveedores de servicio. Elementos que el proveedor de servicio requiere para proveer sus servicios. - Implementación del servicio, dependencias hacia otros servicios, multiplicidad de las dependencias, la versión del servicio del cual depende, la versión del servicio proveído y otros recursos de los cuales dependa el correcto funcionamiento del servicio (Semántica expresada en metadatos). - Aspectos relacionados a la calidad.

Elem. de comp. Vs. Nivel de Aplic.	ESOA	EmSOA	ASOA
	Disponibilidad, confiabilidad, desempeño del servicio	Razón de ruido en la señal, estado de credibilidad, entropía	Escalabilidad, disponibilidad, confiabilidad, extensibilidad y desempeño, los elementos de software llegan y parten a tiempo de ejecución
	Estructuras y tipos de datos, firmas de los métodos		

Tabla 3.4: Comparación entre los niveles de aplicación de SOA

3.2.4. Elementos ASOA para soporte de la metodología

En el caso particular de ASOA no se cuenta con elementos tales como tablas de generación de escenarios de atributos de calidad, catálogos de tácticas y patrones arquitectónicos, o los objetivos de negocio que satisface. Por este motivo en ésta sección se presenta la forma en que se pueden determinar los elementos anteriores en el contexto de ASOA. En la Figura 3.4 se muestra el proceso, de forma genérica, que se siguió en está sección y que se puede seguir para determinar los elementos arquitectónicos de cualquier otro patrón o estilo arquitectónico.

3.2.4.1. Estudio de plataformas que soportan ASOA

Para identificar a los atributos de calidad específicos de ASOA se analizaron las siguientes plataformas:

- *Eclipse*: Es un entorno de desarrollo integrado (IDE) para la creación de sistemas de computo y además, es un marco de trabajo (conjunto de APIs[1] y herramientas que permiten desarrollar sistemas de forma genérica) que permite desarrollar herramientas, otros IDE's y aplicaciones. La arquitectura de Eclipse, presentada en la Figura 3.5, muestra que considera a ASOA como estilo de organización por las siguientes razones:

[1] Conjunto de funciones y/o procedimientos que ofrece un componente a través de su(s) interfaz(ces) para que pueda ser usado por otro componente

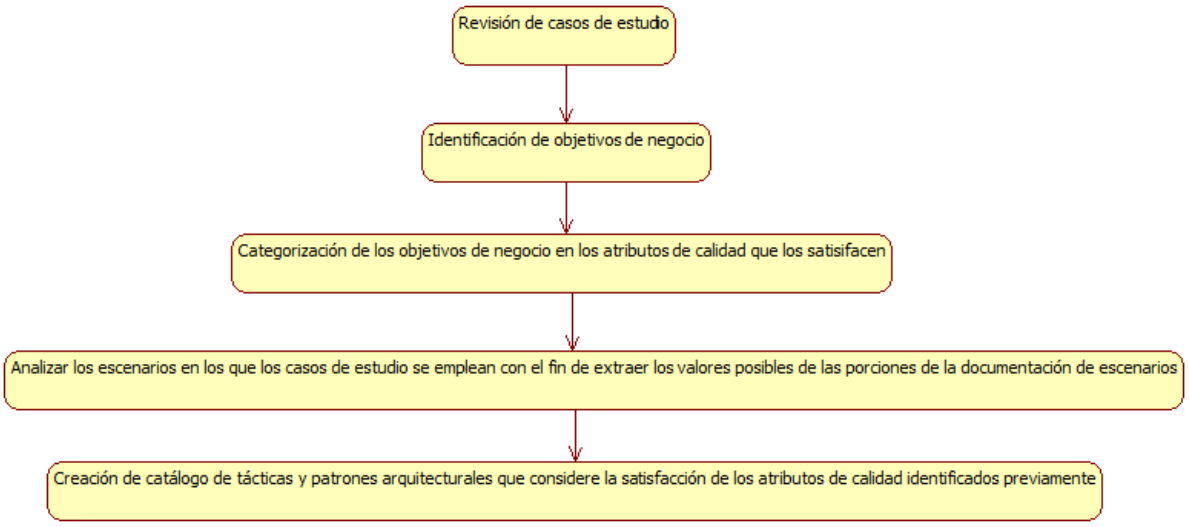


Figura 3.4: Proceso genérico para determinar a los elementos arquitectónicos

- Los *plugins* (elementos de extensión) se pueden instalar o retirar en cualquier momento, durante la ejecución de la plataforma.

- Un plugin se puede considerar como un componente de software debido a que expresa sus dependencias hacia otros plugins de forma explícita, se puede implantar de forma independiente y estar compuesto por otros elementos, no necesariamente plugins.

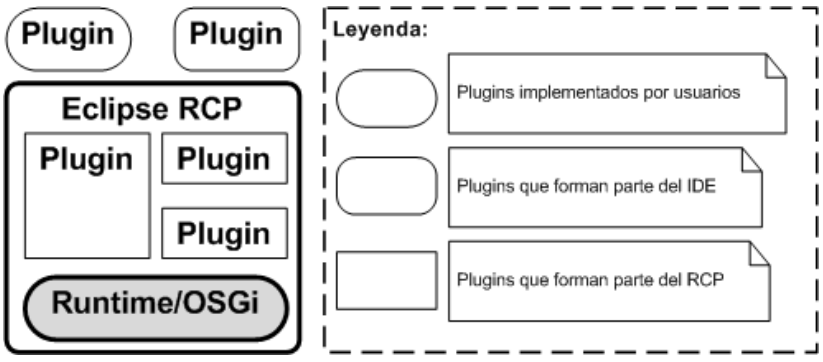


Figura 3.5: Vista de descomposición modular de la arquitectura de Eclipse (alto nivel).

- Los plugins se ligan a sus dependencias a tiempo de ejecución.
- Los plugins encuentran sus dependencias mediante el registro de plugins, que es equivalente al registro de servicios.
- Cuando los plugins se registran, se habilitan como proveedores de servicios.

- *Component Object Model (COM):* Plataforma de componentes de software introducida por Microsoft en 1993 que permite la comunicación inter-proceso y la creación dinámica de objetos en cualquier lenguaje que soporte la tecnología [wwwCOM]. La arquitectura de COM, presentada en la Figura 3.6, muestra que considera a ASOA como estilo de organización por las siguientes razones:

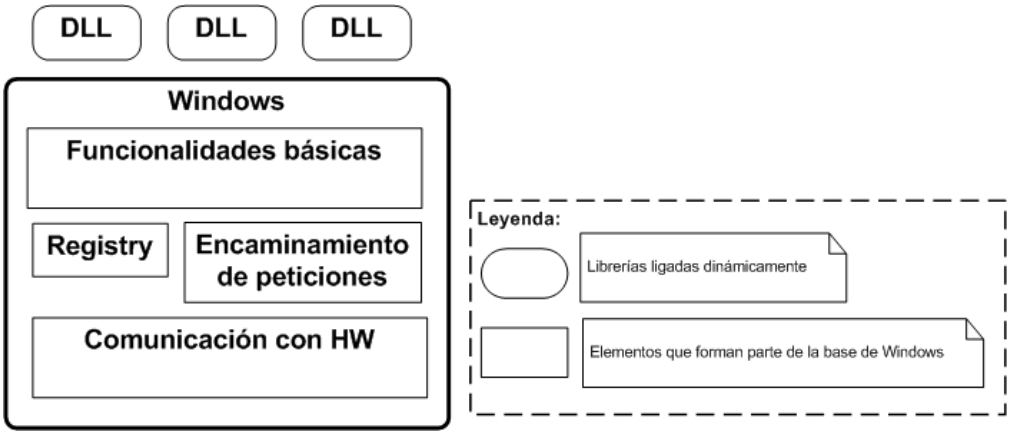


Figura 3.6: Vista de descomposición modular de la arquitectura de Windows (alto nivel)

- Las *DLL* (librerías de ligado dinámico) se pueden instalar o desinstalar en cualquier momento de la ejecución del sistema.
- Las DLL expresan sus dependencias hacia otras de forma explícita, se pueden implantar de forma independiente y pueden estar compuestas por otros elementos, lo que encaja con la definición de [Szyperski98] de componente de software.
- Las DLL se ligan a sus dependencias a tiempo de ejecución.

- Las DLL encuentran a sus dependencias en el Registry de Windows, que juega el papel de registro de servicios
- Cuando las DLL se registran, se habilitan como proveedores de servicios.

- *OSGi/Declarative Services*: OSGi es la especificación de una plataforma que permite realizar actividades de puesta en marcha en el entorno de producción (deployment) de "módulos" java a tiempo de ejecución. Ésta plataforma permite instalar o desinstalar archivos JAR[2] (llamados "bundles") y se encarga de manejar sus dependencias de código. De forma adicional, los archivos JAR pueden dar origen a componentes lógicos que se conectan entre ellos siguiendo un enfoque orientado a servicios, donde el registro de servicios está provisto por la plataforma OSGi. Cuando se desarrollan aplicaciones dentro del entorno de OSGi, es necesario escribir código para administrar los aspectos dinámicos, sin embargo, también es posible usar un enfoque declarativo (llamado Declarative Services - DS) para que el entorno de ejecución se encargue de realizar la conexión y desconexión de los componentes durante la ejecución. La arquitectura de OSGi/DS, presentada en la Figura 3.7, muestra que considera a ASOA como estilo de organización por las siguientes razones:

  - Los bundles se pueden instalar, detener o desinstalar en cualquier momento.
  - Los bundles expresan sus dependencias de forma explícita y se pueden instalar de forma independiente con lo que encajan con la definición de componentes de software de [Szyperski98].
  - Los bundles se ligan a sus dependencias a tiempo de ejecución.
  - Los bundles encuentran sus dependencias mediante el registro de servicios.
  - Cuando los bundles se registran, se habilitan como proveedores de servicios.

[2] Formato de empaquetamiento de archivos con extensión class empleado por Java, este formato permite distribuir el código ejecutable de los programas creados con Java

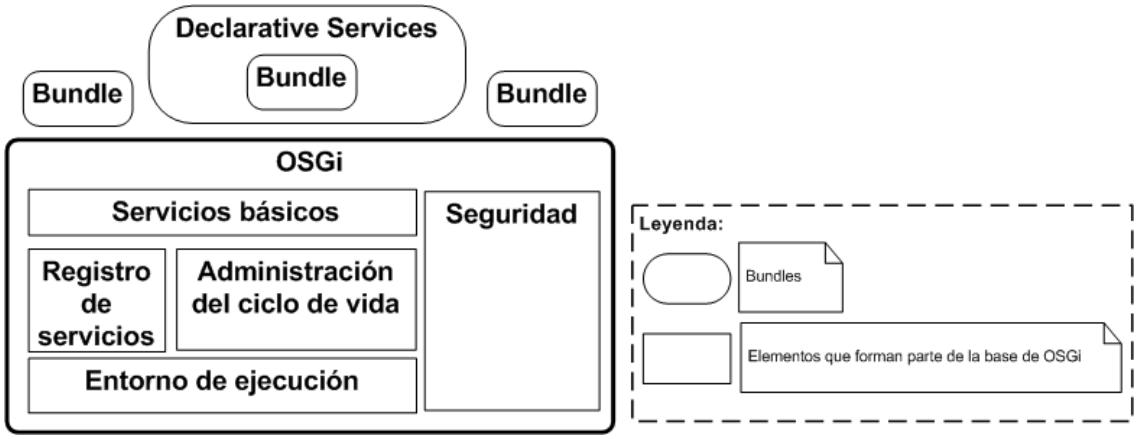


Figura 3.7: Vista de descomposición modular de la arquitectura de OSGi/DS (alto nivel)

3.2.4.2. Objetivos de negocio y atributos de calidad relacionados con ASOA

Al realizar el análisis de estas plataformas se encontró que buscan satisfacer los siguientes objetivos de negocio:

- Permitir a los usuarios finales la integración de nuevas funcionalidades después de que hayan instalado la aplicación.
- Permitir a los usuarios de la aplicación sustituir los elementos de la misma de forma fácil y dinámica.
- Facilitar la administración de elementos que pueden ingresar o partir de forma dinámica del sistema a los desarrolladores con el fin de que los sistemas sean creados de forma sencilla y en menos tiempo.
- Permitir la evolución independiente entre componentes e interfaces con el fin de soportar la compatibilidad hacia atrás.
- Permitir que desarrolladores de otras organizaciones puedan desarrollar e integrar funcionalidades a las aplicaciones.

De los objetivos de negocio antes mencionados se determinó que los siguientes atributos de calidad eran preponderantes con respecto al contexto de ASOA:

- *Modificabilidad a tiempo de ejecución:* Se refiere al costo de realizar cambios en el sistema mientras este se encuentra en ejecución [Bass2003].
- *Desempeño:* Se refiere a cuando llegan los eventos al sistema y cuándo y bajo qué circunstancias los responde [Bass2003].
- *Interoperabilidad:* Capacidad de un conjunto de entidades que pueden comunicarse para compartir información específica y operar de acuerdo con una semántica operacional acordada.
- *Facilidad de pruebas:* Se refiere a la facilidad con la cual un sistema demuestra sus fallas a través de pruebas [Bass2003].

3.2.4.3. Tablas de generación de escenarios de atributos de calidad

La generación de tablas de atributos de calidad es posible una vez que se han determinado los atributos de calidad. Como se mencionó en la sección 2.1.2, estas tablas de generación de atributos de calidad permiten apoyar en la creación de los escenarios propios de ASOA.

Las plataformas mencionadas anteriormente fueron analizadas en busca de escenarios que satisficieran a los objetivos de negocio determinados en el punto anterior, el resultado de este análisis resultó en tablas de generación de atributos de calidad para cada atributo de calidad específico a ASOA. Las tablas identificadas se muestran en: Tabla 3.5, Tabla 3.6, Tabla 3.7 y Tabla 3.8.

Porción del escenario	**Valores posibles**
Fuente	Usuarios finales; desarrolladores (internos/terceras partes); probadores; un (o varios) componente del sistema
Estímulo	Retiro/partida; actualización; incorporación/arribo; sustitución; agrega una nueva interfaz de servicio; modifican algún recurso de la aplicación
Artefacto	Sistema; componentes; interfaces; recursos de la aplicación

Porción del escenario	Valores posibles
Entorno	Tiempo de ejecución; desarrollo; reinicio de forma simultanea
Respuesta	- La funcionalidad es incorporada/retirada/actualizada/sustituida - La aplicación/funcionalidad es operacional * Especificar el tiempo en que las dependencias se satisfacen - Los elementos afectados por el cambio se adaptan de forma automática - La actualización es posible - El componente debe proveer implementaciones para varias versiones de la misma interfaz de servicio - Las modificaciones son realizadas
Medida de la respuesta	- La aplicación [no] requiere ser reiniciada - La aplicación no requiere ser recompilada para incorporar el cambio - La modificación no afecta a otros componentes - Los componentes que dependen de una versión particular de una interfaz de servicio no se ven afectados - El tiempo de reinicio no excede en un X % con respecto al tiempo antes de agregar la funcionalidad Y - La aplicación no requiere una verificación global de su integridad - La funcionalidad puede ser utilizada de inmediato - Se puede continuar con los pasos del proceso a partir del estado que dejó el componente que parte - Los recursos del sistema se mantienen consistentes después del cambio

Tabla 3.5: Tabla de generación de escenarios de modificabilidad a tiempo de ejecución

3.2.4.4. Catálogo de tácticas y patrones arquitectónicos para ASOA

Para poder satisfacer a los escenarios de atributos de calidad creados mediante las tablas antes presentadas es necesario usar catálogos de tácticas y patrones arquitectónicos, sin embargo, actualmente no es fácil encontrar catálogos que relacionen tácticas con patrones arquitectónicos. Por esta razón, en el contexto del proyecto se generó un catálogo de este tipo para el dominio específico de ASOA. La Figura 3.8 muestra el catálogo de tácticas y patrones

Porción del escenario	**Valores posibles**
Fuente	Clientes; usuarios finales
Estímulo	Realiza la función "Función X"; agrega la funcionalidad X; agregan funcionalidades a lo largo del ciclo de vida del sistema
Artefacto	Sistema
Entorno	Sobrecarga de la aplicación; la aplicación esta en ejecución
Respuesta	- El sistema ejecuta la función - La aplicación [no] requiere ser reiniciada - La funcionalidad es operacional - Las funcionalidades son incorporadas
Medida de la respuesta	- En menos de Y segundos genera la respuesta, contados a partir de [criterio de cuenta] - El tiempo de reinicio no se excede en un Y % con respecto a [criterio] - Especificar el momento en que la funcionalidad es operacional (de inmediato, después de que la aplicación se reinicia, después de satisfacer sus dependencias) - La aplicación no consume más de Y MB durante cualquier ejecución

Tabla 3.6: Tabla de generación de escenarios de desempeño

Porción del escenario	Valores posibles
Fuente	Desarrollador de prueba unitaria; integrador incremental
Estímulo	Desea realizar una prueba unitaria de caja negra/stress; probar interacción entre varios componentes
Artefacto	Un componente; un conjunto de componentes; la aplicación completa
Entorno	A tiempo de desarrollo
Respuesta	- La prueba unitaria es realizada - La prueba de integración puede realizarse al conjunto específico de componentes a ser probados
Medida de la respuesta	- No es necesario que los proveedores de servicio estén presentes - Es necesario que algunos de los proveedores de servicio estén presentes - Los componentes que se están probando interactúan conforme a [criterio] - No se conoce la implementación interna de los componentes

Tabla 3.7: Tabla de generación de escenarios de facilidad de pruebas

Porción del escenario	Valores posibles
Fuente	Un usuario final; un desarrollador
Estímulo	Desea que componentes desarrollados en lenguajes de programación distintos interactúen; que la aplicación se pueda ejecutar en varias plataformas; que componentes de aplicaciones distintas interactúen
Artefacto	El sistema; componentes
Entorno	A tiempo de diseño; a tiempo de integración
Respuesta	- Los componentes interactúan - La aplicación se ejecuta en las plataformas X, Y, Z
Medida de la respuesta	- La aplicación se mantiene operacional - Los componentes intercambian información sin producir errores de tipo [criterio]

Tabla 3.8: Tabla de generación de escenarios de interoperabilidad

arquitectónicos en el contexto específico de ASOA. En esta figura los elementos en negritas son las tácticas arquitectónicas (categorías y sub-categorías), los elementos con fuente normal son los patrones arquitectónicos y por último los elementos en itálicas son políticas (variaciones en las acciones que desarrolla un patrón arquitectónico dependiendo de la condiciones bajo las cuales se desea utilizar) que se pueden implementar en el patrón arquitectónico.

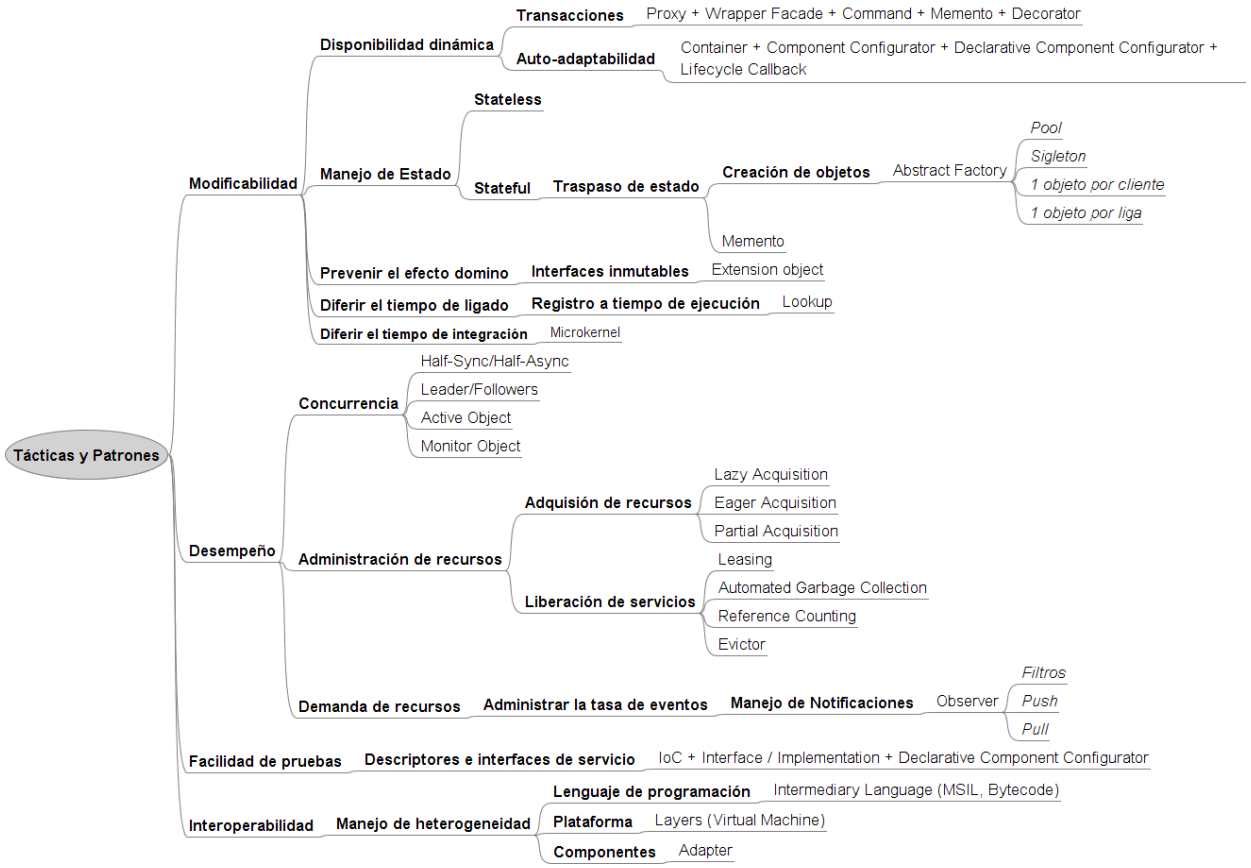


Figura 3.8: Catálogo de tácticas y patrones arquitectónicos específicos de ASOA

Los elementos presentados en la Figura 11 se explicarán a continuación, primero se introducirá el propósito general de cada patrón arquitectónico y posteriormente se explicarán las tácticas arquitectónicas (categorías y sub-categorías).

Patrones arquitectónicos:

1. Proxy [Buschmann2007]: Este patrón encapsula la funcionalidad de un componente dentro de otro llamado Proxy (intermediario), con el cual los clientes se comunican en

lugar de hacerlo con el componente real. Esto permite establecer condiciones previas y/o posteriores a los clientes antes de usar el componente real.

2. Wrapper Facade [Buschmann2007]: Este patrón evita el acceso a funcionalidades de bajo nivel directamente, envolviéndolas en grupos relacionados dentro de un API diferente. Esto facilita a los desarrolladores la programación de las aplicaciones.

3. Command [Buschmann2007]: Encapsula las peticiones en objetos Command y los habilita con una interfaz común para ejecutar las peticiones. Este patrón permite desacoplar al emisor del receptor.

4. Memento [Gamma94]: Este patrón permite, sin que la encapsulación sea violada, capturar y externalizar el estado del un objeto de forma tal que el mismo pueda ser restituido en otro momento.

5. Decorator [Gamma94]: Este patrón permite agregarle funcionalidades de forma dinámica a los objetos.

6. Container [Buschmann2007]: Este patrón provee el entorno de ejecución con la infraestructura técnica necesaria para integrar componentes en aplicaciones específicas sin atar a los componentes a las plataformas o aplicaciones. Este patrón permite desacoplar a los componentes de los detalles técnicos del entorno donde se ejecutan.

7. Component Configurator [Buschmann2007]: Este patrón desacopla las interfaces de sus implementaciones y provee un mecanismo para configurar de forma dinámica a los componentes sin la necesidad de reiniciar al sistema o apagarlo. Este patrón permite crear aplicaciones de forma dinámica.

8. Declarative Component Configurator [Buschmann2007]: Este patrón permite describir la configuración de los componentes de forma declarativa, con lo cual el entorno de ejecución, a tiempo de ejecución, le asigna las dependencias que el componente requiera.

9. Lifecycle Callback [Buschmann2007]: Este patrón permite definir una interfaz con métodos que controlan el ciclo de vida del objeto en ejecución, estos métodos son usados por el entorno de ejecución.

10. Abstract Factory [Gamma94]: Este patrón provee una interfaz que permite crear familias de objetos relacionados sin especificar una clase concreta. Las políticas de creación pueden ser:

    a) Singleton: Solo existe una instancia del componente a lo largo de todo el tiempo de ejecución de la aplicación y todos los otros componentes la comparten.

    b) Pool: Se tiene una cola con un número limitado de instancias del componente, cuando se requiere de una instancia, la fábrica regresa la instancia al frente de la cola, si la instancia no se usa en un tiempo determinado, la instancia se regresada a la cola.

    c) Un objeto por cliente: Una instancia del componente es creada para cada cliente en el sistema. Esta política podría verse como un Singleton a nivel cliente.

    d) Un objeto por liga: Una instancia nueva se creada cada que una liga nueva es solicitada. En esta política, una liga se refiere a la relación de uso que se establece cuando un elemento solicita los servicios de otro.

11. Extension Object [Gamma96]: Este patrón permite anticipar que la interfaz de un objeto necesite ser extendida en el futuro y permite que interfaces adicionales se puedan definir por extensión de objetos.

12. Lookup [Buschmann2007]: Este patrón provee un servicio de registro/desregistro de referencias de servicios. Este patrón permite encontrar a los servicios disponibles en un sistema.

13. Half-sync/Half-async [Buschmann2007]: Este patrón divide a los elementos concurrentes en 2 capas (síncronos y asíncronos) y agrega una capa entre estas 2, esta capa contiene un servicio de mensajería para que las capas síncrona y asíncrona puedan comunicarse.

14. Leaders/Followers [Buschmann2007]: Se tiene un pool de hilos de ejecución que esperan por peticiones. De este pool solo el elemento al frente de la cola (líder) espera las peticiones que llegan al sistema, cuando la petición llega, el líder actual promueve a otro hilo como líder. Este patrón permite el acceso concurrente de peticiones garantizando la escalabilidad del sistema.

15. Active Object [Buschmann2007]: Las peticiones realizadas por los clientes se ejecutan en un hilo distinto al de él, esto permite que de forma asíncrona se pueda comunicar un cliente con el objeto que atiende sus peticiones.

16. Monitor Object [Buschmann2007]: Este patrón ejecuta los métodos de un objeto compartido en el hilo de cada cliente, estos métodos son sincronizados por lo cual solo se puede ejecutar por un cliente a la vez.

17. Lazy Acquisition [Buschmann2007]: Este patrón permite adquirir los recursos que un objeto necesita hasta el momento preciso en que son necesarios en memoria.

18. Eager Acquisition [Buschmann2007]: Este patrón permite adquirir los recursos que un objeto necesita antes de que el objeto los requiera.

19. Partial Acquisition [Buschmann2007]: Este patrón permite dividir en etapas la adquisición de los recursos que un objeto necesita. En cada etapa el objeto adquiere la parte de los recursos que requiere.

20. Leasing [Buschmann2007]: Este patrón establece una duración de posesión de un componente o recurso, después del cual el componente o recurso es liberado.

21. Automated Garbage Collection [Buschmann2007]: En este patrón se define un elemento llamado recolector de basura, el cual busca en la memoria referencias de objetos que no hayan sido referenciadas por largo tiempo.

22. Reference Counting [Henney2002]: En este patrón cada componente tiene un registro. Este registro se incrementa cuando otro componente establece una referencia y dismin-

uye cuando se libera la referencia, al llegar el registro a cero, la memoria ocupada por el componente se libera.

23. Evictor [Buschmann2007]: Este patrón introduce un componente que monitorea y controla el tiempo de vida de los componentes en memoria. Los componentes que no sean utilizados por un cierto periodo son eliminados de la memoria.

24. Observer [Gamma94]: Varios objetos interesados en cambios en el estado de un objeto particular se registran en el mismo como observadores. Cuando un cambio de estado ocurre se notifica de los cambios ocurridos. Las formas en que los objetos realizan la notificación de cambio de estado se puede realizar de las siguientes formas:

    a) Pull: Los objetos interesados buscan de forma periódica cambios en el objeto de interés.

    b) Push: Cuando un cambio ocurre se lanzan notificaciones a los objetos registrados.

    c) Filtros: Cuando un cambio ocurre sólo se notifica a los objetos que se sabe de antemano que utilizaran para algún fin la notificación.

25. Inversion of Control (IoC) [Fowler2004]: En este patrón las dependencias de los objetos son inyectadas al mismo por el entorno de ejecución. Esto permite que los componentes no estén pre-cableados.

26. Interface/Implementation [Bass2003]: En este patrón los componentes exportan su funcionalidad a través de la interfaz que implementan. Los componentes cliente sólo pueden interactuar con la implementación de la funcionalidad a través de su interfaz, esto con la finalidad de poder cambiar las implementaciones de las interfaces sin alterar a los componentes que las usan.

27. Intermediary Language: En este patrón el código fuente de los componentes escritos en diferentes lenguajes de programación es traducido a un lenguaje común. Este lenguaje, a su vez, es traducido al lenguaje de la plataforma específica donde se ejecutan los

componentes. Ejemplos del uso de este patrón se pueden observar en el ByteCode de Java y en MSIL de Microsoft.

28. Layers [Buschmann2007]: Define varios conjuntos de elementos (capas), cada capa tiene responsabilidades distintas y al interior de cada capa las responsabilidades están relacionadas entre sí.

29. Adapter [Gamma94]: Este patrón permite convertir una interfaz en otra que un cliente espera utilizar.

30. Microkernel [Buschmann2007]: Este patrón permite crear diferentes versiones de una aplicación mediante la extensión de un conjunto mínimo de funcionalidades contenidas en un núcleo de forma plug-and-play. Las aplicaciones se pueden integrar mediante alguno de los siguientes enfoques:

    a) El núcleo no provee funcionalidad y requiere de los elementos de extensión para que la aplicación ofrezca alguna funcionalidad útil.

    b) El núcleo provee funcionalidad y los elementos de extensión la usan y enriquecen al núcleo.

Tácticas arquitectónicas

Modificabilidad a tiempo de ejecución

1. *Disponibilidad dinámica* [Cervantes2003]: La disponibilidad dinámica contempla que componentes de software que proporcionan un servicio pueden aparecer o desaparecer en cualquier instante a tiempo de ejecución, este comportamiento está fuera del control de la aplicación que usa los servicios [Cervantes2003]. Para tratar con componentes de software que exhiben este comportamiento se pueden usar las siguientes tácticas:

    a) Transacciones: Las transacciones pueden verse como etapas sucesivas en un proceso que se deben ejecutar de forma atómica (todo o nada) [Bass2003]. Las transacciones pueden emplearse en el contexto de ASOA para realizar rollback de operaciones en las cuales intervienen componentes que pueden partir en cualquier

instante de la ejecución de la aplicación. Esto permite garantizar que si un componente desaparece, el estado en el cual queda la aplicación es consistente.

b) Auto-adaptación: El objetivo de la auto-adaptación es construir aplicaciones que son capaces de adaptase de forma automática, a tiempo de ejecución, a la disponibilidad dinámica de sus componentes [Cervantes2004]. En ASOA los componentes de software pueden exhibir disponibilidad dinámica, por lo cual puede ser útil que los clientes que usan estos servicios se adapten de forma automática a este hecho. También, puede ser útil que cuando un nuevo servicio llegue, los clientes que pueden emplearlo se liguen a él de forma automática o decidan entre seguir usando el servicio al cual están ligados actualmente o bien usar el que acaba de llegar.

2. *Manejo de estado*: Los componentes de software pueden mantener un estado de la conversación con otros componentes (Stateful) o también podrían no hacerlo (Stateless). Como parte de los principios de diseño orientados a servicios, los servicios de ASOA deben mantener un mínimo de estado por periodos de tiempo lo más reducidos posibles, esto con el fin de facilitar el reemplazo de servicios.

   Para algunos casos puede ser útil mantener el estado de la conversación entre componentes (por ejemplo cuando se está realizando una transacción), en estos casos se puede considerar la forma en que el estado se puede transferir de un servicio a otro en caso de estos se puedan sustituir de forma dinámica. Este traspaso de estado podría realizarse de las siguientes formas:

   a) Antes de que una instancia de servicio parta, puede dejar su estado guardado en algún lugar mediante el empleo del patrón "*Memento*", de esta forma el nuevo servicio puede recuperar el estado anterior. Para realizar la transferencia de estado podría ser necesario notificar al cliente que deberá esperar algunos instantes en lo que la instancia del nuevo servicio recupera el estado.

   b) Cuando se crea una instancia de servicio se le puede inyectar un estado de con-

versación específico. Para crear las instancias de servicio puede hacerse uso de Fábricas de objetos que implementen alguna de las políticas mencionadas en "*Abstract Factory*". Cuando se usan estas políticas debe tenerse en mente las siguientes consideraciones:

1) Singleton: En este caso, cuando un cliente se desliga de la instancia única, esta continúa manteniendo su estado. Esto implica que no hay necesidad de transferir el estado ya que todos los clientes compartirán el estado de la instancia única. Esta política se podría comparar con un patrón de pizarrón, en donde todos los usuarios del pizarrón pueden escribir en él y también pueden leer de él.
2) Pool: Cuando un cliente requiere de una instancia, la fábrica regresa la instancia al frente de la cola, si el servicio no se usa en un tiempo determinado, la instancia se regresa a la cola. Cuando el cliente requiere interactuar nuevamente con el servicio, se toma la instancia de servicio al frente de la cola, y se le inyecta el estado asociado al cliente.
3) Un objeto por cliente: Cuando el cliente libera su instancia de servicio y la requiere nuevamente debe proporcionar a la fábrica una forma de identificación única para poder otorgarle la instancia apropiada.
4) Un objeto por liga: Para esta política es complicado implementar un mecanismo de transferencia de estado debido a que se podría solicitar una instancia nueva antes de que la anterior cargue el estado. Este escenario en particular es problemático debido a que puede causar inconsistencias durante la conversación

3. *Prevenir el efecto dominó* [Bass2003]: Cuando es realizado un cambio este puede afectar a componentes que no están directamente relacionados con el cambio, es decir, si es modificado el componente A, el componente B también debe serlo debido a que de alguna forma está relacionado con el componente A. En el contexto de ASOA este tipo de modificaciones que afectan a terceros puede evitarse empleando la siguiente táctica:

a) Interfaces inmutables: Los cambios en las interfaces de servicio pueden ocasionar problemas en el funcionamiento de los clientes del servicio (esto se debe a que no se conoce previamente a los clientes del servicio, ni tampoco al número de estos), por lo que se recomienda mantener a las interfaces de servicio sin cambios. Mantener a las interfaces de servicio sin cambios puede ser una actividad muy compleja debido a la evolución propia de los sistemas, para tratar de lidiar con esto es recomendable usar el patrón "*Extension Object*".

4. *Diferir el tiempo de ligado*: En ASOA las dependencias de los servicios se pueden satisfacer en 2 momentos, a tiempo de cargado de la aplicación en memoria y a tiempo de ejecución. Para realizar el ligado de las dependencias en cualquiera de estos 2 momentos se puede hacer uso de la siguiente táctica:

   a) Registro a tiempo de ejecución: Ésta táctica permite descubrir a los componentes que forman parte de la aplicación mediante el uso de un servicio de registro/desregistro. Cuando un componente requiere de otro pregunta al registro y si la dependencia se encuentra ambos son ligados en cualquiera de los momentos antes mencionados. Ésta táctica puede ser implementada usando el patrón "*Lookup*".

5. *Diferir el tiempo de integración*: En ASOA las aplicaciones pueden ser integradas por usuarios finales o desarrolladores de diversas organizaciones. Diferir el tiempo de integración se refiere al momento en el cual las dependencias de los componentes que forman a la aplicación son satisfechas y todas las funcionalidades que la aplicación ofrece están disponibles. Para lograr diferir el tiempo en que la aplicación se integra se puede hacer uso del patrón "*Microkernel*".

Desempeño

1. *Concurrencia*: Algunas de las tareas realizadas por la aplicación podrían ejecutarse de forma paralela, (con esto el tiempo de respuesta de la aplicación disminuye) estas actividades o conjuntos de ellas se pueden ejecutar en varios hilos de ejecución [Bass2003].

La introducción de la concurrencia en ASOA permite manejar más peticiones por unidad de tiempo, pero se debe considerar que un servicio que invoca a otro puede perder el flujo de control debido a que el servicio invocado se podría ejecutar en un hilo distinto.

Cuando se ejecutan las peticiones, éstas deben ser lanzadas en hilos de ejecución, para obtener el hilo en el cual la petición se ejecutará se puede hacer uso de alguno de los siguientes patrones: "*Half-Sync/Half-Async*", "*Leader/Followers*", "*Active Object*" o "*Monitor Object*".

2. *Administración de recursos*: La administración de recursos se refiere a la forma en la cual se obtienen y liberan las referencias a los recursos, para ello se puede hacer uso de las siguientes tácticas:

   a) Adquisición de recursos: La adquisición de recursos se refiere al momento en el cual se obtienen las referencias a los recursos. Estas referencias pueden obtenerse de forma anticipada ("*Eager Acquisition*"), tardía ("*Lazy Acquisition*") o por etapas ("*Partial Acquisition*").

   b) Liberación de recursos: Ésta táctica se refiere a la forma en la cual se liberan las referencias a los recursos. La liberación puede consistir en dejar libre la referencia para que otra entidad pueda usarla o la liberación de la memoria, en cuyo caso la referencia en destruida. Esta liberación puede hacerse de forma implícita ("*Leasing*", "*Automated Garbage Collection"* y "*Evictor*") o explícita ("*Reference Counting*").

3. *Demanda de recursos*: La demanda de recursos se debe al procesamiento de eventos que solicitan recursos [Bass2003]. Un incremento eventual en el número de eventos o peticiones solicitadas incrementa la latencia en la respuesta de estos eventos, con el fin de reducir la latencia se puede hacer uso de la siguiente táctica:

   a) Administrar la tasa de eventos: De ser posible se debe reducir la cantidad de

eventos que son lanzados, esto con el fin de disminuir la demanda de los recursos del sistema. En ASOA los eventos generados tienen que ver con las notificaciones que los servicios reciben del registro de servicios, para administrar este tipo de eventos se puede hacer uso de la siguiente táctica:

1) Manejo de notificaciones: El manejo de las notificaciones se refiere a la forma en la cual las notificaciones llegan a sus destinatarios, este proceso se puede realizar mediante el envío explícito del registro a los servicios (Push) o mediante un sondeo de los servicios al registro (Poll). Adicionalmente, este proceso se puede optimizar mediante el uso de filtros que permiten que las notificaciones sean enviadas o sondeadas sólo si el servicio tiene interés en ésta.

Facilidad de pruebas

1. *Descriptores e interfaces de servicios*: Los clientes de servicio no tienen acceso a la implementación del servicio, sólo pueden acceder a la interfaz del servicio y conocen la forma de interactuar con él porque tienen acceso a la descripción del servicio. Esto facilita las pruebas debido a que existe una separación entre interfaz e implementación y, adicionalmente, si se va a probar un servicio el cual está compuesto de otros se pueden crear sustitutos (Stubs) de los servicios que lo componen dado que se conoce de antemano el comportamiento de cada servicio que se está empleando. Ésta táctica se puede implementar mediante la utilización de los patrones "*IoC*", "*Interface / Implementation*", y "*Declarative Component Configurator*"

Interoperabilidad

1. *Manejo de heterogeneidad*: En SOA a nivel aplicación se pueden manejar los siguientes niveles de heterogeneidad:

   a) Lenguaje de programación: Los componentes desarrollados en distintos lenguajes de programación se integran mediante el uso de intermediarios, estos exportan el servicio provisto por el componente en un lenguaje de programación común a

todos los elementos de la aplicación. En este nivel de de heterogeneidad se puede hacer uso del patrón "*Proxy*".

b) Plataforma: En general, en ASOA los componentes residen dentro del mismo nodo, pero la aplicación se podría instalar en nodos con plataformas distintas (por ejemplo, que la aplicación deba funcionar en plataformas Windows, Linux, etc.). En este caso se debe asegurar que lo elementos puedan interoperar entre sí y con la plataforma (si lo requieren). Para lograr este objetivo se podrían usar máquinas virtuales o hacer uso de estándares comunes entre las distintas plataformas. En este nivel de heterogeneidad se puede hacer uso del patrón "*Intermediary Language*".

c) Componentes: En ASOA un posible escenario podría ser la incorporación de un componente que originalmente no estaba diseñado para ser empleado en el contexto de la aplicación. Un ejemplo podría ser añadir un plugin de Firefox como plugin de Eclipse, en este escenario se debe asegurar que el plugin de Firefox podrá interoperar en el contexto de plugins de Eclipse. Para lograr el objetivo anterior se puede hacer uso del patrón "*Adapter*".

En síntesis, si se desea realizar algún método de desarrollo arquitectónico en el contexto de algún estilo o patrón arquitectónico específico, es necesario identificar las tablas de generación de escenarios de atributos de calidad, el catálogo de tácticas y patrones arquitectónicos y se deben definir los objetivos de negocio que satisface el patrón o estilo arquitectónico previo a la realización del método, esto es lo que se hizo en el contexto de ASOA, con lo cual se facilita el diseño arquitectónico basado este nivel de aplicación.

Capítulo 4

Adaptación de las Metodologías

En los capítulos anteriores se describieron los conceptos y elementos necesarios para el planteamiento y comprensión de este proyecto. En este capítulo se explicarán las restricciones que considera esta propuesta de investigación, se presenta la propuesta de adaptación a la metodología de desarrollo arquitectónico propuesta por el SEI al contexto de equipos de desarrollo pequeños y además, se propone una integración de las adaptaciones propuestas con el Proceso de Software en Equipo a nivel Introducción (TSPi).

4.1. Restricciones en el desarrollo de las adaptaciones

Las restricciones que son consideradas en el desarrollo de las adaptaciones son:

- *Se debe considerar ASOA como una guía para organizar a los elementos arquitectónicos:* La propuesta de adaptación se evaluará creando el diseño arquitectónico de un caso de estudio de la vida real que considera objetivos de negocio que se pueden satisfecer con el uso de ASOA.

- *Equipos de desarrollo pequeños:* En las organizaciones de desarrollo de cualquier tamaño se pueden encontrar equipos de desarrollo pequeños, por lo cual se requiere de metodologías que les permitan realizar de forma eficiente el desarrollo arquitectónico.

4.2. Adaptación de la metodología del SEI

Como se mencionó anteriormente, las principales características de la metodología de desarrollo arquitectónico del SEI son:

- Para obtener un mayor beneficio, los métodos que integran esta metodología se deben utilizar juntos (a excepción de ATAM que se puede realizar opcionalmente) y,
- Los métodos arquitectónicos del SEI tienden a ser costosos (en términos de tiempo de adaptación, costo de capacitación, etc.).

En esta sección se presenta la propuesta de adaptación para que la metodología de desarrollo arquitectónico propuesta por el SEI se pueda emplear en el contexto de equipos de desarrollo pequeños y los métodos que la integran funcionen de forma coordinada, además, se propone una secuencia de realización de los métodos diferente a la propuesta por la metodología de desarrollo arquitectónico del SEI con la finalidad de reducir el re-trabajo en la documentación. De forma general la adaptación al proceso propuesto sigue el orden mostrado en la Figura 4.1 (-A indica que es una adaptación del método original). Cabe señalar que en esta propuesta se supone que todos los métodos son realizados.

La idea general de las adaptaciones a los métodos es mantener la esencia de cada método original mientras se simplifican algunas de sus actividades como se menciona a continuación:

- QAW-A: Obtener escenarios de atributos de calidad refinados y priorizados por un pequeño grupo de involucrados en el proyecto y adicionalmente, priorizar las restricciones de diseño y requerimientos a alto nivel con los mismos involucrados.
- ADD-A: Realizar un diseño arquitectónico y una documentación preliminar de este diseño de tal forma que se pueda evaluar sin realizar una documentación tan detallada como la propone VaB.
- AEM: Evaluar como el diseño propuesto satisface a las motivaciones arquitectónicas y producir una lista de riesgos, no riesgos, compromisos y puntos sensibles. La evaluación

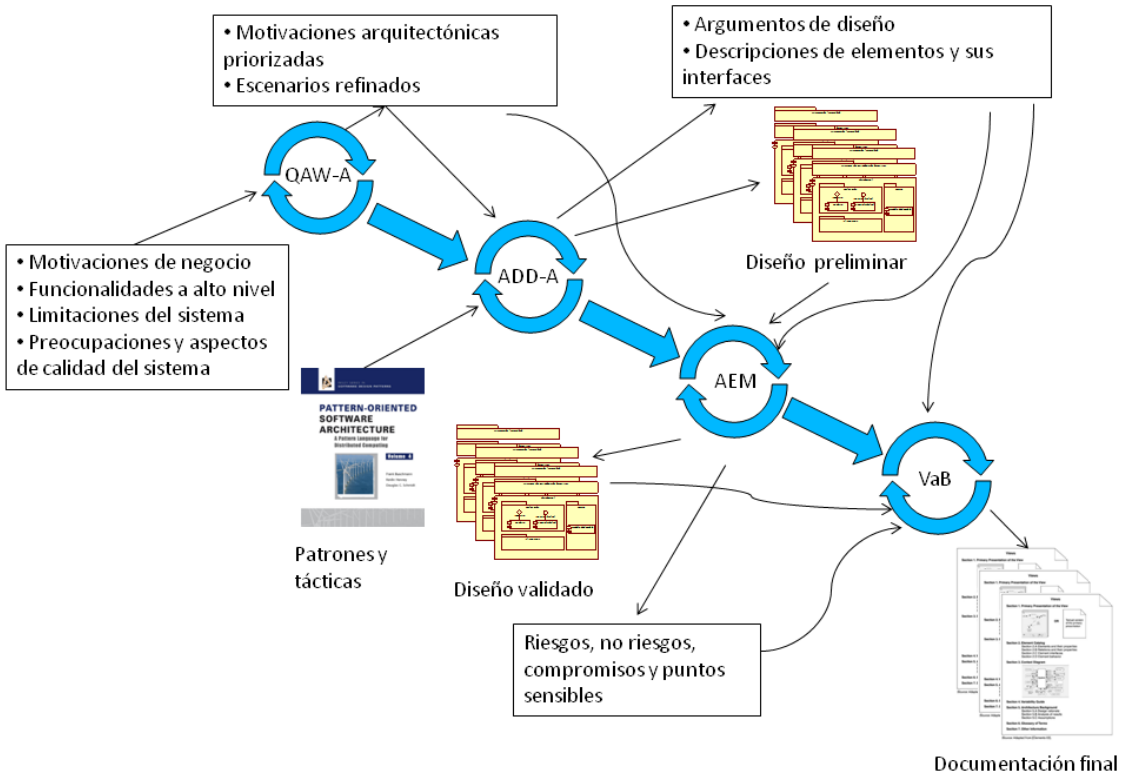


Figura 4.1: Propuesta de adaptación de la metodología de desarrollo arquitectónico del SEI

es realizada por miembros de la organización de desarrollo con una documentación preliminar.

- VaB: Documentar el diseño previamente validado de forma detallada.

Las siguientes secciones proveen más detalle de la forma en que los métodos fueron adaptados al contexto de equipos de desarrollo pequeños.

4.2.1. Adaptaciones a QAW (QAW-A)

Las adaptaciones en este método se enfocaron en reducir el número de involucrados en el taller y en agilizar los pasos que comprende.

En QAW-A se requiere de un mínimo de 4 participantes en el taller: 2 representantes de la organización cliente y 2 de la organización de desarrollo, de los cuales uno es el arquitecto, quién guiará el desarrollo del taller y el otro es el líder de proyecto quién documentará los artefactos resultantes del taller. Esta reducción en el número de participantes (en QAW original se requieren mínimo 5 y máximo 20 participantes) se debe a que, por un lado esperar a que muchos involucrados puedan participar puede retrasar la realización del taller, y por otro lado es más fácil guiar el taller con menos participantes. Adicionalmente, está reducción en el número de participantes se puede realizar dado que los equipos de desarrollo pequeños en general desarrollan sistemas de mediana a baja complejidad.

En la Tabla 4.1 se presenta la guía para la realización de QAW-A en la cual los elementos en negritas representan cambios respecto a QAW original, de estos cambios los más importantes son:

- Se contempla una actividad previa al taller en la cual el arquitecto prepara el manual de participación que se les enviará a los participantes en el mismo.
- El arquitecto, en la priorización de escenarios, puede promover como prioritarios a 1 o 2 escenarios que no hayan sido considerados así por el resto de los participantes.
- Se contempla una actividad posterior al taller en la cual los participantes priorizan las restricciones de diseño y los requerimientos funcionales a alto nivel.

La guía de QAW-A al igual que el resto de las que se presentan en este trabajo se inspiran en el formato de las guías de realización de actividades que propone el Proceso de Software Personal (PSP). La plantilla de refinamiento de escenarios se muestra en la Tabla 4.2, esta plantilla se retoma de la plantilla de refinamiento de escenarios propuesta en el QAW original y se debe crear una instancia de la misma por cada escenario refinado. Adicionalmente, se propone una lista de verificación que apoya en la revisión de la forma en la cual se documenta cada escenario refinado. La lista de verificación de escenarios refinados se puede encontrar en el Apéndice A.

Propósito	Guiar al equipo de desarrollo en realización del taller de atributos de calidad (QAW-A)

Criterios de entrada		- Motivaciones de negocio del sistema - Funcionalidad a alto nivel - Limitaciones sobre el sistema - Intereses y aspectos de calidad del sistema
Roles involucrados		- **Arquitecto:** Captura escenarios crudos, guía la priorización de los mismos, facilita las discusiones y asegura que los pasos de este método se lleven a cabo como se planearon. - **Líder de proyecto:** Captura escenarios crudos, su priorización, refinamiento y cualquier cuestión que surja durante el taller. - **Involucrados/participantes:** Personas directamente relacionadas con el proyecto que aportan información sobre sus intereses relevantes a la calidad del sistema. Estos incluyen miembros del equipo de desarrollo, miembros de la organización cliente y el arquitecto.
General		QAW-A es un método usado para obtener, priorizar y refinar escenarios de atributos de calidad antes de que la arquitectura se comience a diseñar formalmente. Este método depende de la participación de los involucrados en el proyecto. Los escenarios son utilizados por el arquitecto para analizar a la arquitectura e identificar intereses y posibles estrategias de mitigación.
Paso	Actividades	Descripción
1	**Preparación del taller**	El arquitecto escribe el manual de participación del taller adecuado al contexto del sistema, en este manual explica: - Qué son los atributos de calidad - Qué son los escenarios de atributos de calidad - Cuál es la estructura de un escenario - Quiénes son los participantes en el taller - Cuáles son los pasos del taller y el tiempo estimado para cada paso El manual debe hacer énfasis en que el taller es importante para las partes involucradas y que requiere de su compromiso para realizarlo

2	Presentación del taller e involucrados	**El arquitecto presenta a todos los participantes en el taller incluyéndose a sí mismo y debe enfatizar que la información relacionada a cada participante se encuentra en el manual de participación**. También describe brevemente los pasos del taller y los tiempos estimados para realizar cada paso.
3	Presentación de las motivaciones de negocio	Un representante de la organización cliente explica: - Las motivaciones de negocio que dan origen al sistema - La funcionalidad a alto nivel, intereses y aspectos de calidad El arquitecto captura la información que pueda dar indicios de atributos de calidad (modificabilidad, escalabilidad, seguridad, etc.).
4	Presentación del plan arquitectónico	El arquitecto presenta: - Planes y estrategias sobre como satisfacer los principales requerimientos de negocio. - Los principales requerimientos técnicos y limitaciones que guían las decisiones arquitectónicas. - Presentación de los diagramas de contexto, diagramas de sistema de alto nivel y otras descripciones que haya realizado.
5	Identificación de motivaciones arquitectónicas	Durante los pasos anteriores el arquitecto ha identificado una lista de motivaciones arquitectónicas que debe incluir: - Requerimientos a alto nivel - Intereses de negocio - Objetivos y objetivos de negocio - Atributos de calidad - Restricciones de diseño **El arquitecto otorga un receso de 10 minutos para que los participantes consoliden las notas que tomaron**. Al término de este tiempo el arquitecto expone su lista de motivaciones tratando de lograr un acuerdo con los participantes.

6	Lluvia de ideas sobre generación de escenarios	El arquitecto recuerda a los participantes la estructura de un escenario crudo (estímulo, entorno, respuesta) y presenta las tablas de generación de escenarios de atributos de calidad que sean necesarias. Todos los participantes en forma round-robin crean un escenario que exprese sus intereses respecto al sistema. - Se pueden requerir de 2 a 4 rondas - En cada ronda, cada participante crea un escenario El arquitecto debe asegurar: - Que se genere al menos un escenario por cada motivante arquitectónica - Que cada atributo de calidad tenga asociado al menos un escenario de atributo de calidad, el cuál de preferencia debe ser cuantificable. - Que se generen escenarios de los siguientes tipos: * Caso de uso * Exploratorios * Crecimiento
7	Consolidación de escenarios	El arquitecto pregunta a los participantes si hay escenarios parecidos en contenido, de haberlos los participantes se ponen de acuerdo para fusionarlos.
8	Priorización de escenarios	Todos los involucrados contaran con un número de votos igual al 30 % del total de escenarios consolidados. La votación se realiza en 2 rondas de forma round-robin - En cada ronda, cada participante emite la mitad de sus votos - Los participantes pueden asignar cualquier cantidad de votos a cualquiera de los escenarios. Al término de las rondas se cuentan los votos y de acuerdo al número de votos que reciba cada escenario se establece una prioridad. **El arquitecto puede promover 1 o 2 escenarios que crea relevantes como prioritarios**.
9	Refinamiento de escenarios	Según el tiempo restante, el líder de proyecto refina los 6 o 7 escenarios más importantes empleando el plantilla de la Tabla 4.2. **Adicionalmente, puede usar la lista de verificación mostrada en la Tabla A.3 para revisar la forma de los escenarios que este refinando**.

10	**Post-QAW**	Mientras el líder de proyecto refina los escenarios, el arquitecto prioriza la lista de requerimientos funcionales a alto nivel y las restricciones de diseño con el resto de los participantes de la siguiente forma: - El arquitecto asigna un número de votos igual al 30 % del total de requerimientos funcionales a alto nivel y restricciones de diseño a cada participante, incluyéndose él mismo. - En 2 rondas de forma round-robin cada involucrado emite sus votos. * Se emiten el 50 % de los votos en cada ronda. - Al final de las rondas el arquitecto cuenta los votos y, según el número de votos que reciba cada requerimiento o restricción establece una prioridad. - Al termino de la priorización el arquitecto le asigna a cada motivante arquitectónica una importancia relativa que puede ser Alta, Media o Baja.
Criterios de salida		- Lista priorizada de motivaciones arquitectónicas - Lista priorizada de escenarios crudos - Escenarios refinados

Tabla 4.1: Guía para la realización del QAW-A

4.2.2. Adaptaciones a ADD (ADD-A)

Las adaptaciones en este método se enfocan en agilizar el diseño arquitectónico tomando como entradas las salidas de QAW-A (lista priorizada de motivaciones arquitectónicas) y en producir una documentación preliminar que se pueda usar para evaluar el diseño propuesto (no es necesario esperar a crear todo el paquete de documentación arquitectónica realizado en VaB para realizar una evaluación del diseño). La documentación de las iteraciones realizadas en ADD-A se realiza llenando la plantilla que se presenta en la Tabla 4.4, esta plantilla se llena conforme el método es realizado (está característica no está presente en el método original).

Al final del método se agregó otra iteración, en esta última iteración se consolidan todas las decisiones arquitectónicas tomas a lo largo de las iteraciones anteriores. Esta última iteración puede servir como punto de partida de la evaluación del diseño arquitectónico. Adicionalmente, en el Apéndice A se puede encontrar la lista de verificación para revisar la

<table>
<tr><td colspan="3">Identificador: Id de escenario refinado [colocar un identificador único para el escenario]</td></tr>
<tr><td colspan="2">Escenario(s):</td><td>[Lista de escenarios crudos que contempla este refinamiento]</td></tr>
<tr><td colspan="2">Objetivos de negocio:</td><td>[Lista de objetivos de negocio que contempla este escenario refinado]</td></tr>
<tr><td colspan="2">Atributos de calidad relevantes:</td><td>[Lista de atributos de calidad que contempla este escenario refinado]</td></tr>
<tr><td rowspan="6">Componentes del escenario:</td><td>Estímulo:</td><td>[Condición que el sistema debe considerar cuando llega a él]</td></tr>
<tr><td>Fuente del estímulo:</td><td>[Entidad que genera el estímulo. Ej.: usuario, sistema externo]</td></tr>
<tr><td>Entorno:</td><td>[Condiciones bajo las cuales el estímulo ocurre]</td></tr>
<tr><td>Artefacto:</td><td>[Parte del sistema que es estimulada. Puede ser incluso todo el sistema]</td></tr>
<tr><td>Respuesta:</td><td>[Actividad realizada después de que el estímulo llega al sistema]</td></tr>
<tr><td>Medida de la respuesta:</td><td>[Forma cuantificable de medir la respuesta del sistema]</td></tr>
<tr><td colspan="2">Preguntas:</td><td>[Preguntas relacionadas con como el sistema puede lograr la respuesta]</td></tr>
<tr><td colspan="2">Cuestiones:</td><td>[Lista de cuestiones relacionadas con el escenario de atributo de calidad que preocupa a los participantes]</td></tr>
</table>

Tabla 4.2: Plantilla de refinamiento de escenarios propuesta por Len Bass et al en [Bass2003]

forma con la cual se documentan las iteraciones. Las adaptaciones al ADD se pueden ver en la Tabla 4.3 (los elementos en negritas representan cambios respecto al ADD original).

Propósito		Guiar al arquitecto en la realización del diseño guiado por atributos (ADD-A)
Roles involucrados		- Arquitecto
Criterios de entrada		Listas priorizadas y detalladas de: - Requerimientos funcionales - Restricciones de diseño - Atributos de calidad (escenarios refinados de atributos de calidad) - Documento de especificación de requerimientos completo y revisado el cuál contemple las salidas propias de QAW-A Los elementos anteriores son llamados en esta guía requerimientos de diseño
General		ADD-A es un proceso de diseño basado en la descomposición iterativa del sistema. En cada iteración son seleccionados patrones y tácticas arquitectónicas con el fin de satisfacer los atributos de calidad
Paso	Actividades	Descripción
1	Confirmar que existe información suficiente sobre los requerimientos	El arquitecto se debe asegurar que al menos se haya realizado un QAW-A y que haya arrojado los siguientes resultados: - Lista priorizada de motivaciones arquitectónicas (restricciones de diseño, funcionalidad a alto nivel, etc.) - Escenarios de atributos de calidad refinados y priorizados La priorización indicará al arquitecto el orden en el cuál el diseño se realizará

2	Elegir un elemento del sistema a descomponer	Para un desarrollo nuevo el elemento a descomponer es el sistema completo y a este se le asignan todos los requerimientos de diseño. Cuando se refina un elemento en particular: - El sistema ya se ha dividido y a cada parte se le han asignado un subconjunto de los requerimientos de diseño - Se selecciona una parte del sistema empleando alguno de los siguientes criterios: * Conocimiento de la arquitectura * Riesgos y dificultad * Criterios de negocio * Criterios organizacionales * La prioridad asignada por alguna autoridad El arquitecto elige un elemento del sistema empleando los criterios antes mencionados.
3	Identificar candidatos a motivaciones arquitectónicas principales	El arquitecto categoriza cada requerimiento por su impacto en la arquitectura y coloca la categoría en combinación de la prioridad asignada por los involucrados. Esto resulta en un nuevo orden de prioridad de los requerimientos de diseño. - Los primeros 2 o 3 requerimientos guían los siguientes pasos del diseño y se llamarán candidatos a motivaciones arquitectónicas principales

4	Seleccionar un concepto de diseño que satisfaga a las motivaciones arquitectónicas principales	El arquitecto: - Identifica los intereses de diseño asociados con cada candidato a motivante arquitectónica principal - Por cada interés de diseño se genera una lista de tácticas y patrones arquitectónicos, esta lista puede ser generada con base en: * Experiencia, conocimiento y habilidades con tácticas y patrones que satisfagan al interés * Catálogos de tácticas y patrones arquitectónicos (como por ejemplo el mostrado en la Figura 3.8) - Identifica los atributos que permiten elegir la táctica o patrón más adecuado y estima el valor de cada atributo - Elige el patrón o táctica más apropiado y registra los argumentos, en los cuáles se debe incluir: * Pros y contras de cada patrón o táctica considerada * Compromisos entre patrones o tácticas - Considera los patrones y tácticas que ha encontrado y decide la forma de combinarlos - Describe los patrones y tácticas elegidos en los diferentes tipos de vistas (descomposición, usos e implementación). - Evalúa y resuelve inconsistencias en los conceptos de diseño, esta actividad incluye: * Determinar si hay motivaciones que no se han cubierto * Encontrar los patrones o tácticas que cubran a las motivaciones faltantes * Se evalúa el diseño actual contra el diseño producto de las iteraciones previas con el fin de resolver inconsistencias

5	Instanciar elementos arquitectónicas y asignarles responsabilidades	El arquitecto: - Instancia los elementos encontrados y les asigna responsabilidades según su tipo - Asigna las responsabilidades del elemento padre entre los hijos según los argumentos definidos en el paso anterior - Crea instancias adicionales si: * Las propiedades de un atributo de calidad sobrecargan de trabajo al elemento designado para cubrirlo * Es necesario atender otros atributos de calidad en el elemento
6	Definir interfaces para los elementos instanciados	El arquitecto: - Ejercita los requerimientos funcionales de los elementos encontrados en el paso 5 - Observa la información que es producida por un elemento y consumida por otro(s) desde la perspectiva de las vistas creadas descubriendo las interfaces: * Externas * Con la infraestructura * Entre los elementos del sistema, etc. - Crea la especificación de interfaces para cada elemento En este punto el arquitecto tiene la información suficiente para llenar la plantilla mostrada en la Tabla 4.4. Adicionalmente, el arquitecto puede usar la lista de verificación mostrada en la Tabla A.4 para revisar la forma de la documentación de las iteraciones realizadas.
7	Verificar y refinar a los requerimientos	El arquitecto: - Verifica que los requerimientos asociados al elemento padre se hayan asignados a uno o más hijos - Traduce cualquier responsabilidad asignada a los elementos hijos en requerimientos expresados en la especificación de requerimientos - Refina los atributos de calidad de los elementos hijos como sea necesario
8	Repetir de los pasos 2 a 7 para el siguiente elemento	De ser necesario el arquitecto regresa al paso 2 para continuar con la descomposición de elementos del sistema

9	**Consolidación del diseño**	El arquitecto crea una instancia adicional de la plantilla de documentación de iteraciones con la consolidación de las decisiones de diseño tomadas en las iteraciones previas.
Criterios de salida		**Una documentación general de vistas dinámicas (del tipo componente y conector), estáticas (del tipo de módulos) y de asignación**.

Tabla 4.3: Guía para la realización de ADD-A

Datos de la iteración	*[Número de iteración, lista de participantes, tiempo de realización]*
Elementos a descomponer	*[Lista con los elementos que se descomponen en la iteración]*
Motivaciones arquitectónicas	*[Lista de motivaciones arquitectónicas que se consideran durante la iteración]*
Alternativas de solución	*[Lista con las alternativas de solución que satisfacen a las motivaciones del punto anterior y los atributos que permiten realizar una discriminación entre alternativas]*
Patrones y/o tácticas arquitectónicas y decisiones de diseño	*[Lista de las soluciones elegidas con base a los atributos de discriminación y las decisiones de diseño tomadas sobre sus elementos]*
Instanciación y descripción de cada elemento *[De forma gráfica o textual se muestran los elementos que resultan de la combinación de los patrones o tácticas arquitectónicas y se describen brevemente sus responsabilidades. Adicionalmente, se puede hacer uso de diagramas que ejemplifiquen la interacción o concurrencia de los elementos mostrados]*	

Interfaz de cada elemento *[Para cada elemento se describe la especificación de su interfaz. Esto puede ser realizado mediante una tabla donde se coloque el nombre de la interfaz, los elementos que la implementan y la especificación de los métodos de la interfaz]*

Tabla 4.4: Plantilla de documentación de iteraciones

4.2.3. Método de Evaluación Arquitectónica (AEM)

AEM es un método que permite evaluar el diseño arquitectónico preliminar producido en ADD-A. Este método se inspira en ATAM pero a diferencia de este AEM no es realizado por un equipo de evaluadores externo. La evaluación arquitectónica se decidió integrar de forma obligatoria al proceso propuesto en este trabajo para garantizar que los diseños propuestos satisfagan a las motivantes arquitectónicas identificadas en QAW-A. Este proceso se realiza antes de VaB para reducir el re-trabajo en la documentación en el caso de que la evaluación no sea satisfactoria, esto fue posible gracias a que en ADD-A se produce una documentación preliminar que refleja las principales decisiones arquitectónicas que fueron tomadas.

La evaluación con AEM se realiza siguiendo la guía que se muestra en la Tabla 4.6. Los resultados obtenidos con AEM se pueden documentar con la plantilla que se muestra en la Tabla 4.5, de esta plantilla se crea sólo una tabla que indique cuáles fueron los riesgos, no riesgos, compromisos y puntos sensibles identificados y una instancia de la segunda tabla para cada análisis que el diseño arquitectónico requiera.

Identificador	**Descripción**	**Responsable**
[Identificador único a este proyecto, en el identificador puede ir el tipo del elemento (Riesgo - R, no riesgo - NR, punto sensible - PS, compromiso - C)]	*[Descripción breve del elemento]*	*[Nombre del responsable en el proyecto que se hará cargo de este elemento (no aplica en el caso de un no riesgo)]*

Motivaciones arquitectónicas	*[Lista de motivaciones arquitectónicas sobre las cuales se realiza la evaluación]*
Preguntas	*[Lista de preguntas que forman parte de algún modelo de evaluación y que dan origen al elemento]*
Respuestas	*[Respuestas por parte del arquitecto]*
Referencia a elementos	*[Referencia al identificador del tipo de elemento relacionado con esta tabla]*
Decisiones arquitectónicas	*[Lista de las posibles soluciones al elemento. No aplica en caso de que el elemento sea un no riesgo]*
Argumentos de diseño	*[Mediante algún método (simulaciones, análisis empírico, modelos matemáticos, etc.) se analizan las decisiones arquitectónicas que fueron tomadas]*
Diseño: *[En caso de ser necesario se puede crear un diseño que explique las decisiones que fueron tomadas]*	

Tabla 4.5: Plantilla de documentación de resultados de AEM

Propósito		Guiar al equipo de evaluación en la evaluación del diseño contra los requerimientos no funcionales
Criterios de entrada		- Lista de motivaciones arquitectónicas - Documentación preliminar realizada en ADD-A
General		El propósito de AEM es evaluar las consecuencias de las decisiones arquitectónicas contra las motivaciones arquitectónicas. A través de este método se puede juzgar que tan apropiada es una arquitectura. AEM permite descubrir riesgos, no riesgos, compromisos y puntos sensibles pero no realiza un análisis preciso.
Paso	Actividades	Descripción
1	Preparación de AEM	El arquitecto entrega a cada miembro del equipo de evaluación una copia de las iteraciones creadas en ADD-A y explica cual es proceso de evaluación y los productos del mismo.

2	Creación del modelo de evaluación	Cada miembro del equipo de evaluación: - Crea para cada atributo de calidad un modelo de evaluación * El modelo de evaluación consiste de preguntas relacionadas con como las decisiones de diseño presentadas en las vistas satisfacen a las motivaciones arquitectónicas
3	Recorrido del diseño arquitectónico	- El arquitecto presenta los patrones y tácticas empleadas para satisfacer a cada motivante arquitectónica principal * Realiza una descripción de cómo los elementos se instancian y se relacionan para cubrir con las motivaciones arquitectónicas - El arquitecto resuelve las preguntas que cada miembro del equipo de evaluación realice durante su presentación.
4	Identificar puntos sensibles	De la sesión de preguntas y respuestas del punto anterior, el arquitecto identifica y documenta las propiedades de cada elemento y/o sus relaciones que pueden ser críticas para alcanzar una respuesta específica de un atributo de calidad.
5	Identificar compromisos	De los puntos sensibles encontrados, el arquitecto identifica y documenta las propiedades de los elementos y/o sus relaciones que pueden afectar a más de un atributo de calidad
6	Identificar riesgos	De los compromisos entre atributos de calidad detectados, el arquitecto identifica y documenta cuales decisiones de diseño podrían causar problemas.
7	Identificar no riesgos	El arquitecto documenta las decisiones de diseño que son consideradas buenas (no son puntos sensibles, riesgos o compromisos) tanto por él como por el equipo de evaluación.
8	Presentar resultados	Los resultados de AEM son documentados empleado la plantilla mostrada en la Tabla 4.5. Adicionalmente, el arquitecto puede usar la lista de verificación mostrada en la Tabla A.5 para revisar la forma en la cual se documentó la lista de riesgos, no riesgos, compromisos y puntos sensibles.
Criterios de salida		- Conjunto de preguntas basadas en atributos de calidad - Lista de riesgos, no riesgos, compromisos y puntos sensibles - Documentación preliminar verificada contra las motivaciones arquitectónicas

Tabla 4.6: Guía para la realización de AEM

4.2.4. VaB-A

La principal diferencia entre VaB y VaB-A es el momento en el cuál se realizan. VaB-A fue colocado después de la evaluación arquitectónica debido a que se consideró que evaluar el paquete de documentación arquitectónica puede representar re-trabajo en el caso de que la evaluación no sea satisfactoria.

En VaB no se encontraron adaptaciones sustanciales, por este motivo el proceso es el mismo que propone el SEI, pero, al tener una documentación preliminar validada la creación del paquete de documentación arquitectónica se simplifica; por ejemplo, los argumentos de diseño se pueden encontrar en la documentación preliminar, por lo cual sólo es necesario transferirlos en la documentación de la vista correspondiente (la plantilla de documentación de vistas propuesta en VaB se puede ver en la Figura 2.1). Por último, la Tabla 4.7 muestra la guía de realización de VaB.

Propósito		Guiar al equipo de desarrollo en la elaboración del paquete de documentación arquitectónica.
Criterios de entrada		- Lista de motivaciones arquitectónicas - Lista de argumentos de diseño - Lista de involucrados - Vistas de descomposición, uso e implementación ya evaluadas mediante AEM
General		VaB considera 3 categorías de vistas (Módulos, componente y conector y asignación) y no limita a un cierto número de estas, el número es determinado examinando las necesidades de información de los involucrados y los atributos de calidad relevantes para los mismos.
Paso	Actividades	Descripción
1	Determinar las necesidades de información	El arquitecto y el equipo de desarrollo crean una tabla de involucrados y sus necesidades de información arquitectónica. Esta información puede ser recopilada analizando que es lo que cada involucrado necesita saber de la arquitectura para que pueda tomar decisiones sobre el sistema (en el caso de arquitectos, administradores, clientes, etc.) o pueda trabajar sobre él (en el caso de programadores, probadores, etc.).
2	Estándares de diseño	El arquitecto presenta al equipo de desarrollo los estándares de diseño.

3	Identificar las vistas candidatas	El arquitecto guía al equipo de desarrollo en la creación de una lista de las vistas que se crearan. - La elección de las vistas depende de las necesidades de información de cada involucrado y de los atributos de calidad que les son relevantes
4	Identificar las vistas que se crearán	El arquitecto guía al equipo de desarrollo en la elección de las vistas a crear y del nivel de información que cada vista proporcionará. - Crean una tabla de involucrados contra vistas donde en cada celda se coloca el nivel de información que cada involucrado requiere - Observan que vistas proporcionan información general o son usadas por muy pocos involucrados - Ven si los involucrados del punto anterior pueden ser satisfechos con otras vistas que provean más información - Buscan vistas que puedan combinarse - Los pasos anteriores producen una lista reducida de vistas las cuales son priorizadas para que sean creadas por el equipo de desarrollo. * Las vistas pueden ser creadas en paralelo partiendo de información general por lo que vistas generales son prioritarias * También pueden ser priorizadas contemplando las necesidades de información de involucrados de alto rango
5	Tareas de documentación	El arquitecto esboza la estructura del paquete de documentación arquitectónica y el trabajo necesario para realizarlo.
6	Asignación de tareas	El líder de proyecto ayuda a asignar las tareas de documentación a los miembros del equipo de desarrollo y obtiene compromiso sobre la fecha en que cada miembro entregará su parte.
7	Realización de las vistas	Cada miembro del equipo de desarrollo elabora las vistas que le fueron asignadas y al terminar se las entrega al arquitecto

8	Actualización del diseño	El arquitecto crea el esbozo del paquete de documentación arquitectónica y añade la información que relaciona a las vistas. Esta información incluye: - Organización de las vistas (Catalogo de vistas y templete) - Descripción del sistema, la forma en que las vistas se relacionan, lista de elementos con su localización y glosario de términos - Contexto del sistema, limitaciones externas y argumentos de diseño (a grandes rasgos) - Estándares de diseño - Trazabilidad con el documento de especificación de requerimientos - Lista de motivaciones arquitectónicas que guiaron el diseño
Criterios de salida		Un paquete de documentación arquitectónica que satisface a los requerimientos de diseño

Tabla 4.7: Guía para la realización de VaB

4.3. Integración de los adaptaciones de los métodos con TSPi

Los métodos que contempla la metodología de desarrollo arquitectónico propuesta por el SEI no están integrados con algún proceso de desarrollo de software, en las siguientes secciones se propone una integración de las adaptaciones antes descritas con el Proceso de Software en Equipos nivel Introductorio (TSPi).

4.3.1. Introducción a TSPi

TSPi es una versión simplificada del Proceso de Software en Equipos (TSP) que está orientada hacia equipos de desarrollo pequeños, pero que a su vez, mantiene las mejores prácticas contenidas en TSP [Humphrey99]. Se eligió TSPi debido a que está orientado al mismo tipo de equipos de desarrollo que son considerados en este proyecto, y además, TSPi contiene guías que documentan claramente las actividades realizadas por cada rol involucrado

en el proceso de desarrollo. Por último, TSPi ha sido probado exitosamente en el entorno académico.

En TSPi el desarrollo se divide en ciclos, los cuales dan como resultado una versión del sistema que se puede probar. En la Figura 4.2 se muestran las actividades consideradas en cada ciclo de TSPi, estas actividades se describen de forma breve a continuación:

- *Lanzamiento:* Se revisan los objetivos del sistema y se asignan miembros al equipo de desarrollo y el rol o roles que cada miembro desempeñará.
- *Estrategia:* Se produce un diseño conceptual, se establece la estrategia de desarrollo, se realizan estimaciones de tamaño y se evalúan los riesgos del proyecto.
- *Planeación:* Se producen los planes de equipo e ingeniería para el ciclo.
- *Desarrollo de requerimientos:* Se definen e inspeccionan los requerimientos para el ciclo y se produce el plan de pruebas de sistema y materiales de soporte.
- *Diseño:* Se produce e inspecciona el diseño a alto nivel para el ciclo y se produce el plan de pruebas de integración y materiales de soporte.
- *Implementación:* Se implementa e inspecciona el producto de software y se produce el plan de pruebas unitarias y materiales de soporte.
- *Pruebas:* Se construyen las pruebas unitarias, de integración y sistema para el producto y se realiza la documentación de usuario.
- *Postmortem:* Producir los reportes de evaluación del ciclo.

Al inicio de cada ciclo se actualizan las entradas de cada actividad contemplando el trabajo realizado en el ciclo anterior.

4.3.2. Adaptaciones a TSPi

La metodología de desarrollo arquitectónico propuesta por el SEI se puede integrar en las primeras etapas de cualquier proceso de desarrollo, TSPi no es la excepción. Se identificó

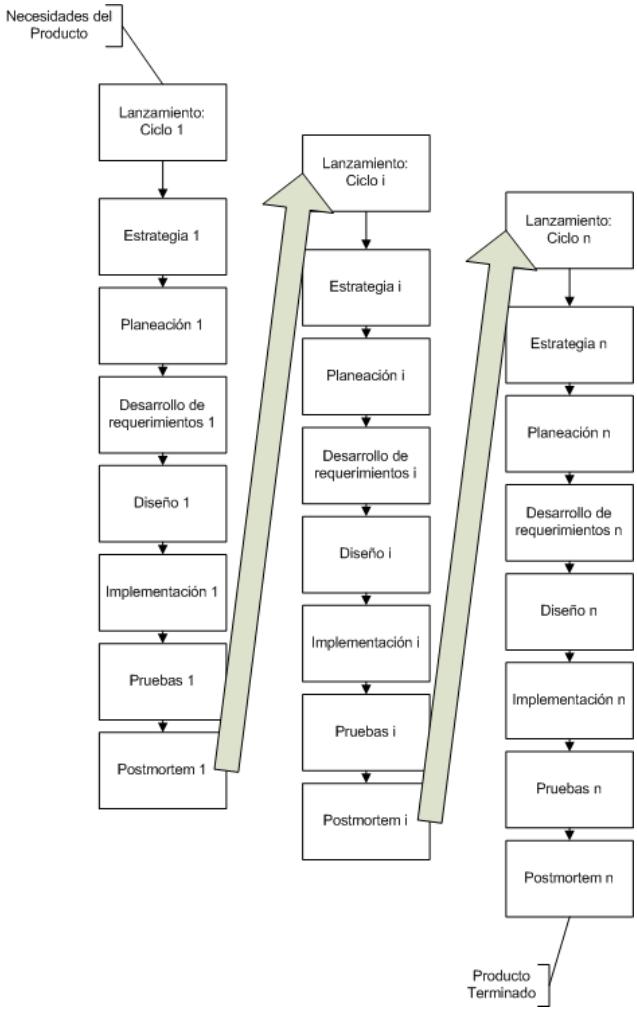


Figura 4.2: Actividades de TSPi enmarcadas dentro de ciclos

que en las actividades de Desarrollo de Requerimientos y Diseño a Alto Nivel es posible incorporar las adaptaciones de la metodología de diseño arquitectónico del SEI propuestas en este proyecto como se muestra en la Figura 4.3. Las adaptaciones consistieron de:

- *Desarrollo de requerimientos:* A la especificación de requerimientos definida por TSPi, se le agregó una sección que considera a los requerimientos no funcionales que deberá soportar el sistema. Por este motivo, se adaptó la guía de desarrollo de requerimientos para que contemplara la captura de requerimientos no funcionales mediante QAW-A. La guía adaptada de TSPi puede encontrarse en el Apéndice A (las adaptaciones realizadas se muestran en negritas). Las adaptaciones consisten de:

- Incorporar a la especificación de requerimientos una sección que considere a los requerimientos no funcionales.
- Al revisar las necesidades del producto, el arquitecto debe preguntar al instructor sobre qué requerimientos no funcionales debe soportar el sistema.
- Al dividir las tareas se debe considerar que se realizarán tareas de obtención de requerimientos funcionales y no funcionales.
- Se debe realizar la documentación de requerimientos no funcionales empleando la guía de QAW-A.
- El plan de pruebas de sistema debe considerar la realización de pruebas que muestren la forma en que se satisfacen los requerimientos no funcionales documentados con QAW-A (escenarios).

- *Diseño:* La guía de diseño no específica: la forma en la cual se eligen los elementos arquitectónicos, cómo se documentan, ni cómo se evalúa el diseño arquitectónico propuesto, por este motivo agregamos estos pasos dentro de la guía de diseño. La guía adaptada de diseño de TSPi puede encontrarse en el Apéndice A (las adaptaciones se muestran en negritas). Las adaptaciones consisten de:
  - Considerar que la especificación de requerimientos contiene las salidas de QAW-A.
  - Dentro de la especificación del diseño de software (SDS) se debe realizar un mapeo entre requerimientos no funcionales y los componentes que los soportan.
  - Se realiza y documenta un diseño arquitectónico preliminar usando la guía de ADD-A.
  - Se evalúa el diseño propuesto en ADD-A contra los requerimientos funcionales y no funcionales siguiendo la guía de AEM.
  - Se documenta el diseño ya validado mediante VaB.
  - La inspección del SDS debe contemplar a los requerimientos no funcionales.

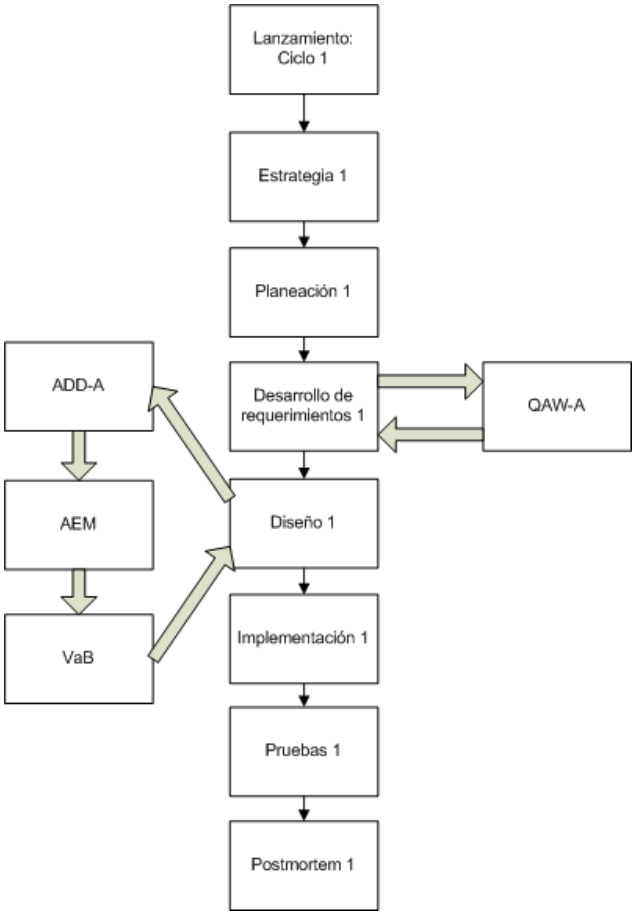


Figura 4.3: Integración de las adaptaciones propuestas con TSPi (las fechas, cualesquiera de ellas, indican el flujo en que se realiza cada una de las actividades dentro de la integración con TSPi)

Se cree que las adaptaciones propuestas en este capítulo son menos costosas que la metodología propuesta por el SEI debido a que:

- Las adaptaciones propuestas ya contemplan equipos de desarrollo pequeños y ya se han integrado con un proceso de desarrollo, por lo que las organizaciones de desarrollo no requerirán invertir recursos ni tiempo en la adaptación e integración de la metodología del SEI.

- Los métodos que integran a las adaptaciones propuestas funcionan de forma coordinada, con lo cuál se puede obtener mayor beneficio de los métodos.

- Se proveen las guías de proceso, plantillas y listas de verificación de cada uno de los métodos adaptados, con lo que el tiempo de capacitación se reduce.

- Sólo se producen los artefactos necesarios, por mencionar un ejemplo, durante el diseño no se crea un paquete de documentación arquitectónica detallado, con lo que se reduce el tiempo de diseño y el número de involucrados en esta parte de la documentación (en VaB se requiere que varias personas participen en el desarrollo del paquete de documentación).

El siguiente capítulo se trata de la realización y evaluación de las adaptaciones propuestas en este capítulo mediante la elaboración del diseño arquitectónico de un sistema real, adicionalmente, se discuten los resultados obtenidos.

Capítulo 5

Evaluación de las Adaptaciones

En este capítulo se muestran los resultados de la evaluación a las adaptaciones que fueron propuestas en el capítulo anterior, la estrategia de evaluación fue la siguiente:

- Elegir un caso de estudio basado en un sistema real.
- Identificar los objetivos de negocio del sistema elegido que se puedan satisfecer con ASOA.
- Realizar el diseño arquitectónico del caso de estudio (el cual se presenta más adelante) mediante 3 experimentos los cuales permitan evaluar a las adaptaciones propuestas. Los experimentos se limitaron a un contexto académico con estudiantes de maestría con poca experiencia en el diseño arquitectónico y sin conocimiento del dominio del problema. Los experimentos involucraron a las siguientes personas:
  - Uno de los experimentos fue realizado junto con el asesor de la tesis.
  - Los otros 2 experimentos fueron realizados junto con el asesor y un grupo de 4 estudiantes de maestría. Cabe mencionar que los 4 estudiantes formaron 2 grupos de desarrollo arquitectónico de 2 miembros, en cada grupo había un miembro con experiencia industrial, por último, a los 2 grupos se les proporcionaron las guías de las adaptaciones propuestas en este proyecto.

Los experimentos realizados con los estudiantes se efectuaron en 5 sesiones de laboratorio de 3 horas cada una en el marco de una de materia de arquitecturas de software. En estas

sesiones los estudiantes sólo siguieron las guías adaptadas de desarrollo de requerimientos y diseño de alto nivel de TSPi y como producto final elaboraron el diseño arquitectónico del sistema propuesto.

En los 3 experimentos se realizaron las siguientes actividades:

- Explicación a los estudiantes de los conceptos básicos de arquitectura de software y la visión del sistema, sin introducir los objetivos de negocio que se pueden satisfacer con ASOA (1 sesión de laboratorio). El otro equipo (el asesor y yo) ya tenía experiencia con el diseño arquitectónico y conocía el dominio del problema.

- Explicación, ejecución de las guías y llenado de las plantillas.

  - Se utilizó una sesión para la realización de QAW-A, durante esta sesión se crearon los escenarios a partir de los cuales se realizarían los diseños de los equipos de estudiantes. El otro equipo realizó un QAW-A aparte después de la realización de este.

  - Los estudiantes dispusieron de 2 sesiones para la realización y documentación del diseño con ADD-A. Cada equipo realizó de forma separada su ADD-A, incluyendo al equipo formado junto con el asesor, el cual contó con 2 semanas para la realización de ADD-A (sólo se ocuparan unas horas de estos días).

  - Se utilizó una sesión para AEM, en esta sesión los estudiantes expusieron los resultados obtenidos con ADD-A y se realizó la evaluación de los mismos. El otro equipo realizó antes su AEM de forma separada.

  - Al término de la realización de cada método se efectuaba una revisión de la "forma" de las plantillas con las listas de verificación proporcionadas.

- Análisis de los elementos relevantes en cada una de las plantillas y comparación contra las plantillas creadas junto con el asesor.

- Identificación de bondades y deficiencias de las adaptaciones.

Por motivos de tiempo no se realizaron las actividades de VaB en ninguno de los 3 experimentos. Las siguientes secciones entran en el detalle de las evaluaciones realizadas en este proyecto de investigación.

5.1. Descripción del caso de estudio (Visión del sistema)

Esta sección presenta una visión general del sistema elegido. Este consiste de un sistema de administración de dispositivos heterogéneos en red que será usado por una empresa mexicana de telecomunicaciones.

La Figura 5.1 muestra un diagrama de contexto general del sistema. En este diagrama varios clientes pueden acceder, a través de la red, a las funcionalidades provistas por los API de los dispositivos en la red a través de un gestor (el software de administración de dispositivos) albergado en un servidor (hardware), el gestor almacena la información de los dispositivos en una base de datos. Los dispositivos, a su vez, pueden informar de problemas en su funcionamiento a través del envío de notificaciones al servidor, posteriormente el servidor hará del conocimiento de los clientes estos problemas.

El sistema mostrado en la Figura 5.1 se originó, desde la perspectiva de la organización que desarrolló el sistema, a partir de los siguientes objetivos de negocio:

- La organización desea vender un sistema que:
    - Soporte agregar o quitar funcionalidades del sistema por usuarios finales mientras este se encuentra en funcionamiento
    - Soporte agregar o quitar manejadores de dispositivos (elementos de software dentro del gestor que permiten la interacción con los dispositivos administrados por el gestor) del sistema por usuarios finales mientras esté se encuentra en funcionamiento

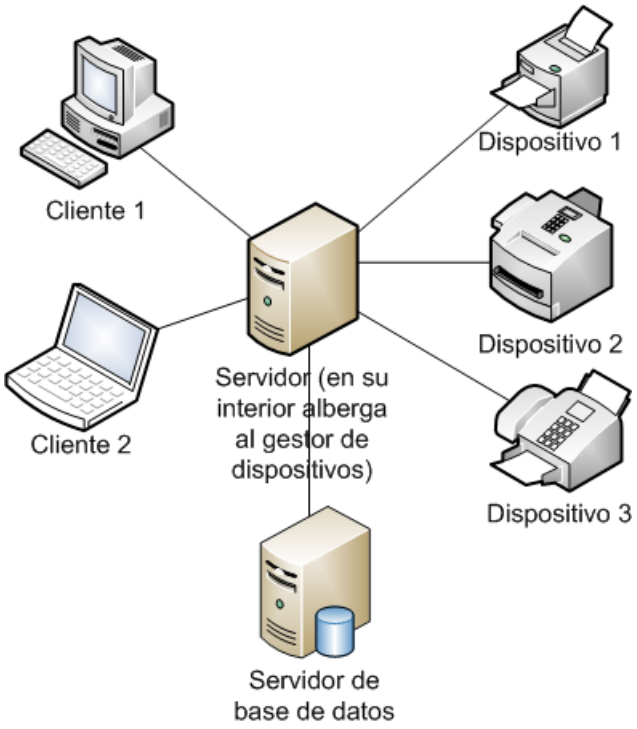


Figura 5.1: Diagrama de contexto general del sistema (las conexiones entre los dispositivos representan conexiones de red)

  - Soporte la integración de nuevas funcionalidades creadas por otras organizaciones a tiempo de desarrollo.
  - Soporte actualizaciones a las funcionalidades presentes en el sistema sin que esto afecte a funciones que dependan de versiones anteriores de la función actualizada.

- La organización requiere crear versiones del sistema adaptadas (versiones pre-configuradas con un conjunto particular de funcionalidades y manejadores de dispositivos) a las necesidades de sus clientes en tiempo limitado.

Y los siguientes objetivos de negocio vistos desde el punto de vista de la organización cliente:

- La organización desea mejorar el tiempo de respuesta de atención ante anomalías en los dispositivos.

- La organización desea que el sistema pueda administrar diferentes tipos de dispositivos.

Para satisfacer a los objetivos de negocio, el sistema debe implementar las funcionalidades a alto nivel mostradas en la Tabla 5.1 y estar sujeto a las siguientes restricciones y limitaciones técnicas:

- Restricciones de diseño:
  - Los dispositivos son administrados de forma centralizada.
  - Los usuarios finales pueden agregar/quitar funcionalidades a tiempo de ejecución.
  - Los usuarios finales pueden agregar/quitar manejadores de dispositivos a tiempo de ejecución.
- Limitaciones técnicas:
  - Los APIs de los dispositivos son provistos por los fabricantes
  - Los APIs de los dispositivos no son estables
  - Los dispositivos pueden fallar en cualquier instante
  - Los dispositivos pueden generar grandes cantidades de información (del orden de cientos de megabytes)
  - Los dispositivos pueden generar grandes cantidades de notificaciones (del orden de cientos de notificaciones por día)
  - Los protocolos de comunicación entre los dispositivos son, en principio, distintos
  - Los dispositivos que se administrarán son heterogéneos
  - El servidor no es, en principio, dedicado

Por último, el sistema debe considerar a los siguientes requerimientos no funcionales:

- El sistema se debe usar de forma fácil por los usuarios.
- El sistema se debe modificar de forma fácil para adecuarlo al tipo de organización al cual sea vendido

Requerimiento funcional a alto nivel	**Descripción**
Administrar dispositivos	Permite administrar a los dispositivos (altas, bajas y cambios de dispositivos).
Administrar usuarios	Permite administrar a los usuarios del sistema (altas, bajas y cambios de usuarios).
Administración de representación gráfica	Permite administrar los elementos gráficos que representan a los dispositivos (altas, bajas y cambios de elementos gráficos).
Interactuar con la representación gráfica	Permite interactuar con la representación gráfica de la topología de dispositivos.
Acceso de usuarios	Permite a los usuarios ingresar a las funcionalidades provistas por el sistema.
Configuración de dispositivos	Permite modificar los diferentes parámetros de configuración que ofrece cada tipo de dispositivo.
Consultar estado de dispositivos	Permite consultar la información de estado del dispositivo.
Crear reporte de estado de dispositivos	Permite crear un reporte con el estado de un dispositivo durante un periodo dado.
Recibir notificaciones de dispositivos	Permite administrar las notificaciones generadas por cada dispositivo.
Colectar datos de los dispositivos	Permite, de forma periódica, colectar información relevante de cada dispositivo.

Tabla 5.1: Funcionalidades a alto nivel

- El sistema debe soportar que sea accedido por múltiples usuarios de forma simultánea
- El sistema debe soportar un incremento en funcionalidades, usuarios y dispositivos
- El sistema debe controlar el acceso de los usuarios y debe tener una bitácora de operaciones realizadas por cada uno de ellos.

5.2. Ejecución de los métodos

Las siguientes secciones profundizaran en los resultados obtenidos en los experimentos realizados. Cabe señalar que los artefactos resultantes de las ejecuciones realizadas por los estudiantes de maestría no se presentaran en este trabajo, pero parte de los productos generados junto con el asesor se encuentran en el Apéndice B.

5.2.1. Ejecución de QAW-A

Se realizaron 2 talleres QAW-A siguiendo la guía propuesta en la sección 4.2.1, uno de los talleres se realizó junto con el asesor de este proyecto y el otro taller fue realizado por el grupo de 4 estudiantes de maestría, en el cual el asesor y yo fungimos como facilitadores. Sólo fueron necesarios 2 talleres debido a que los equipos integrados por los alumnos partirían de los mismos escenarios de atributos de calidad.

El resultado del QAW-A realizado junto con el asesor se muestra en el Apéndice B. Durante las realizaciones del taller se obtuvieron los resultados mostrados en la gráfica de la Figura 5.2 y se observó lo siguiente:

- La falta de experiencia le hizo difícil a los estudiantes identificar escenarios relevantes (que requieren decisiones de diseño significativas para satisfacerlos).
- En general, es complicado establecer métricas a las respuestas de los escenarios. Esto se debe a la falta de experiencia en el dominio del problema, y también, a que algunos escenarios de atributos de calidad son difíciles de cuantificar (por ejemplo algunos escenarios de usabilidad).

- Se observó que el método es útil para identificar y documentar escenarios de atributos de calidad. Los escenarios generados por los estudiantes fueron de calidad aceptable (se consideran así si son específicos al sistema y proporcionan una forma apropiada de medir la respuesta del sistema ante el estímulo del escenario).

- Las tablas de generación de escenarios de atributos de calidad fueron muy útiles en el QAW-A desarrollado junto con el asesor.

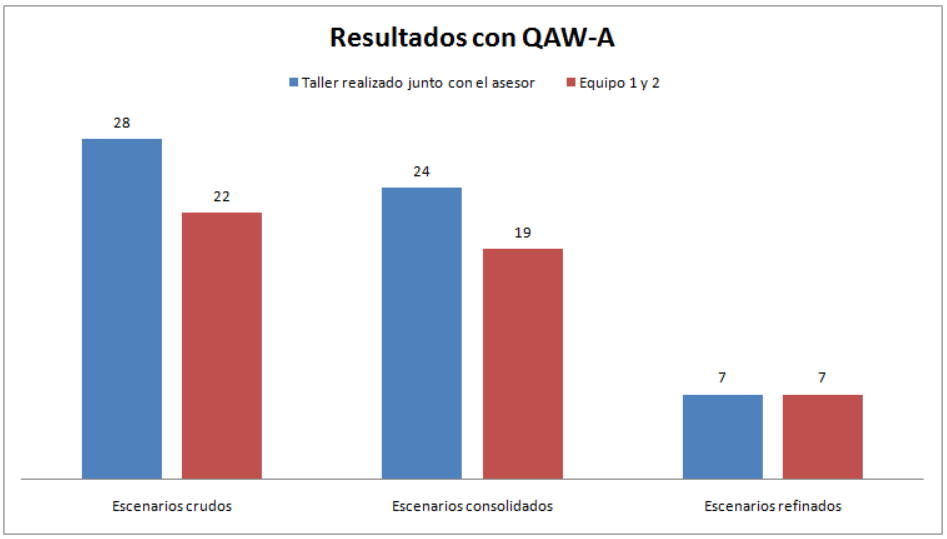


Figura 5.2: Resultados obtenidos en los 2 talleres QAW-A

Del total de 28 escenarios crudos creados en el QAW-A realizado con el asesor, 12 se obtuvieron mediante las tablas de generación de escenarios de atributos de calidad mostradas en la sección 3.2.4.3, este número de escenarios corresponde al total de escenarios crudos relacionados con ASOA.

A continuación se explican los resultados mostrados en la Figura 5.2:

- El número de escenarios crudos que fueron creados por los estudiantes de maestría fue menor debido a que en este taller no se consideraron objetivos de negocio que estuvieran relacionados con el atributo de calidad de modificabilidad a tiempo de ejecución, esto se debe a que los estudiantes no están familiarizados con los conceptos necesarios para tratar con este atributo de calidad.

- El número de escenarios consolidados, de igual forma, fue menor debido a que los estudiantes partieron de una base menor de escenarios crudos.

- En ambos QAW-A el número de escenarios refinados fue igual, esto se debe a que el SEI propone refinar a los 4 o 5 escenarios con mayor prioridad y además, en ambos QAW-A los arquitectos (el asesor y yo) promovieron 2 escenarios que consideraron relevantes.

Un análisis posterior de los resultados obtenidos en el QAW-A realizado por los estudiantes mostró que la guía de QAW-A les permitió crear escenarios crudos de calidad aceptable (la calidad se refiere a que generaron escenarios crudos relacionados con el sistema que además establecían formas realistas de medir la calidad del mismo). Los escenarios creados por los estudiantes tienen una calidad similar a los creados en el QAW-A que fue realizado junto con el asesor, esto es relevante debido a que el asesor y yo teníamos experiencia en el dominio del problema. Adicionalmente, se observó que a pesar de que el asesor y yo guiamos el QAW-A realizado por los estudiantes faltaron algunos escenarios para que los diseños elaborados por los mismos fueran de mayor calidad.

5.2.2. Ejecución de ADD-A

Se realizaron 3 ejecuciones de ADD-A, las cuales involucraron a 2 integrantes por cada ejecución. Los equipos integrados por estudiantes tuvieron como punto de partida el QAW-A realizado durante su sesión de laboratorio.

El resultado del ADD-A realizado junto con el asesor se muestra en el Apéndice B. Durante las ejecuciones del ADD-A se obtuvieron los resultados mostrados en la gráfica de la Figura 5.3 y se observó lo siguiente:

- El método proporciona una buena guía que ayuda a abordar el problema de diseño de forma gradual. Los estudiantes tenían una idea clara de por dónde empezar la descomposición del sistema.

- La plantilla de documentación preliminar demostró su utilidad para documentar las decisiones de diseño tomadas durante la ejecución del método. El hecho de que este

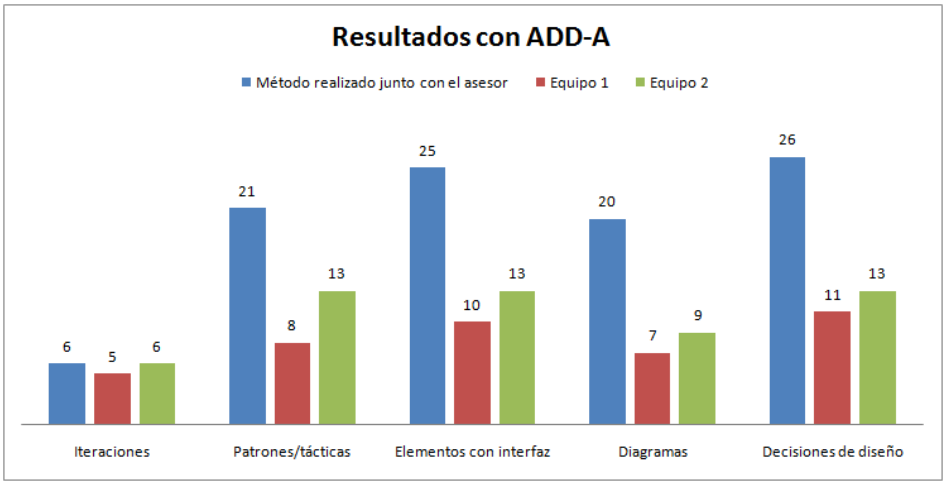


Figura 5.3: Resultados obtenidos en la ejecución de los 3 ADD-A

estructurada en el mismo orden que los pasos del método ayuda a evitar confusiones sobre la forma de llenarla.

- Tener un catálogo de patrones y tácticas arquitectónicas específico a un dominio particular, como el mostrado en la sección 3.2.4.4, facilita la creación del diseño arquitectónico de una aplicación en ese dominio.

Por situaciones de tiempo los equipos 1 y 2 no realizaron la iteración de consolidación, pero, los demás resultados mostrados en la Figura 5.3 no se ven afectados por este hecho.

En los 3 experimentos, la documentación preliminar se realizó empleando la plantilla de documentación de iteraciones que se puede encontrar en la sección 4.2.2 y por último, de los 21 patrones y/o tácticas arquitectónicas necesarias para la creación del diseño propuesto junto con el asesor 7 se tomaron del catálogo de tácticas propuesto en la sección 3.2.4.4, este número de patrones y/o tácticas corresponde al total de patrones y/o tácticas que se relacionan con ASOA.

Un análisis posterior de la documentación de las iteraciones de los estudiantes mostró que no introdujeron los suficientes patrones y tácticas arquitectónicas probablemente debido a su falta de experiencia en el uso de los mismos; esta inexperiencia tuvo 2 consecuencias:

1. Los estudiantes experimentaron problemas con la auto-evaluación de sus diseños, es de-

cir, cuando seleccionaron su conjunto de tácticas y patrones arquitectónicos y evaluaron si satisfacían a las motivaciones de la iteración, la evaluación fue positiva.

2. Los estudiantes tuvieron problemas con la representación en UML de los diseños que proponían. Aunque los estudiantes tienen los conocimientos suficientes en UML, la falta de experiencia les complicó la representación de los patrones y tácticas con esta herramienta.

5.2.3. Ejecución de AEM

Se realizaron 3 AEM, uno de ellos, el realizado junto con el asesor, se realizó en 2 etapas en 2 días distintos. La primera etapa consistió en un recorrido por el diseño propuesto y la realización de un modelo de evaluación, la segunda etapa consistió en resolver y documentar la solución de cada pregunta del modelo de evaluación. Esta forma de realizar AEM es recomendable debido a que le permite a los involucrados en la evaluación reflexionar, sin presiones de tiempo, sobre la creación de un modelo de evaluación adecuado. Los otros 2 AEM fueron realizados en una sesión de laboratorio donde participaron los 4 estudiantes, el asesor y yo. En cada uno de estos AEM el equipo de evaluación consistió del equipo que no realizaba su recorrido arquitectónico, el asesor y yo, este equipo de evaluación creo en el acto el modelo de evaluación y el equipo que presentaba su diseño lo resolvió y documento. Cabe señalar que el AEM realizado junto con asesor no contó con un equipo de evaluación debido a la inexperiencia de los estudiantes.

El resultado del AEM realizado junto con el asesor se muestra en el Apéndice B. Durante las ejecuciones del AEM se obtuvieron los resultados mostrados en la gráfica de la Figura 5.4 y se observó la utilidad de AEM para evaluar si el diseño cubre con las motivaciones arquitectónicas antes de que el paquete de documentación formal se realice, con lo cual se evita re-trabajar en la documentación de las vistas.

Un análisis de los resultados obtenidos con AEM mostró que los diseños propuestos por los estudiantes aún son incompletos, esto debido a que no satisfacen a todos los atributos de calidad de forma completa (algunos fueron satisfechos de forma parcial). Podemos suponer

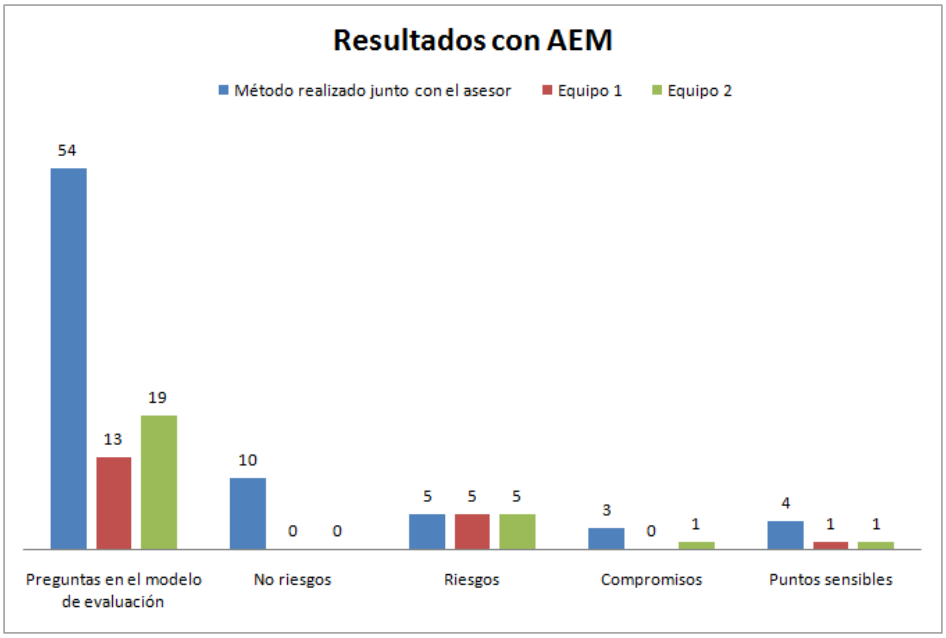


Figura 5.4: Resultados obtenidos en la ejecución de los 3 AEM

que esto se debe principalmente a su falta de experiencia en el dominio del problema, su inexperiencia en el uso de patrones y tácticas arquitectónicas, las suposiciones que hicieron sobre lo que otros involucrados deberían conocer de la arquitectura y a que faltaron algunos escenarios en QAW-A. Con respecto del AEM realizado junto con el asesor, el resultado fue similar al obtenido con los estudiantes, el diseño aún es incompleto, aunque las decisiones de diseño tomadas tienen sustento y esto permite que el diseño sea más entendible.

A continuación se explican los resultados mostrados en la Figura 5.4:

- En cuanto al tamaño del modelo de evaluación, los estudiantes experimentaron problemas debido a que no sabían que preguntar con respecto de la satisfacción de los atributos de calidad, esto suponemos que se debió a que no comprendieron por completo como los patrones y tácticas arquitectónicas satisfacen a los atributos de calidad.

- En cuanto a los no riesgos, debido a que la mayoría de las decisiones arquitectónicas tomadas por los estudiantes satisfacían de forma parcial a las motivaciones arquitectónicas no se encontraron no riesgos en sus diseños.

- En cuanto a los riesgos, creemos que se debieron en gran medida a la falta de experiencia seleccionando patrones o tácticas arquitectónicas y a que no lograron expresarse de forma adecuada con UML.

- Con respecto a los compromisos y puntos sensibles, a los estudiantes les costó trabajo entender cómo realizar su identificación, motivo por el cual su número es reducido.

5.2.4. Observaciones generales con respecto a las adaptaciones propuestas

Al término de los 3 AEM los estudiantes realizaron las siguientes observaciones con respecto de las adaptaciones propuestas:

- El diseño arquitectónico no fue realizado a prueba y error como piensan que comúnmente se realiza.

- A través del seguimiento de las guías propuestas se puede realizar un diseño arquitectónico de calidad aceptable aunque el individuo que realiza las guías no sea un arquitecto experimentado.

- ADD-A y AEM se beneficiarían con la presencia de un arquitecto experto que guíe a los estudiantes mientras se capacitan. Aunque esto podría ser complicado de implementar en una organización de desarrollo pequeña por el costo de contratar al arquitecto experto.

- Una base de conocimiento específica a un cierto dominio de problema, que contemple los patrones y tácticas arquitectónicas más utilizadas en el mismo, podría reemplazar al arquitecto con experiencia.

Capítulo 6

Discusión Crítica

Realizar la evaluación de una metodología es una actividad compleja debido a que interviene el factor humano. Este proyecto, además de lo anterior, se enfrentó la complicación de que los recursos humanos con los cuales se contó fueron estudiantes de maestría que no estaban dedicados de tiempo completo a la evaluación de las adaptaciones. Afortunadamente, se contó con el apoyo y disposición de los estudiantes, con lo cual los resultados obtenidos fueron satisfactorios dado que sus diseños satisfacían de forma parcial a los requerimientos que se les dieron, esto es una ventaja con respecto de la forma en la que tradicionalmente se realiza el diseño arquitectónico. Creemos que la parcialidad en los resultados se debe a que los estudiantes no estaban familiarizados con los conceptos relacionados con arquitectura de software.

Para tratar de complementar la evaluación realizada con los estudiantes se pensó en aplicar las adaptaciones propuestas en los siguientes 2 entornos:

- Por un lado, si se dispusiera de 2 equipos de desarrollo muy similares, 1 equipo aplicaría las adaptaciones propuestas en este trabajo y el otro aplicaría alguna otra forma de realizar el diseño arquitectónico. Esto permitiría contrastar la metodología propuesta en este trabajo con alguna otra.

- Por otro lado, se podría aplicar la metodología en varios proyectos y adicionalmente, se colectarían métricas de tiempo, defectos, etc. A estos resultados se les aplicarían análisis estadísticos para observar el comportamiento de las adaptaciones.

Cabe señalar que estos complementos de evaluación no se realizaron dado que no se disponía de los recursos humanos necesarios ni del tiempo para realizarlos.

Adicionalmente, del trabajo realizado se obtuvieron las siguientes observaciones:

- Se requiere comprender bien los conceptos relacionados con la arquitectura de software para producir diseños adecuados. Este fue uno de los principales problemas que se observaron durante la experimentación con los estudiantes, dado que tuvieron problemas al relacionar los conceptos de atributos de calidad, escenarios de atributos de calidad, tablas de generación de atributos de calidad, patrones y tácticas arquitectónicas. Este problema se observó a pesar de contar con la supervisión del asesor y del autor de este trabajo, pero también se observó que sin esta guía los diseños arquitectónicos realizados por los estudiantes no hubieran logrado satisfacer ni siquiera de forma parcial a los atributos de calidad.

- Se requiere experiencia práctica en las problemáticas relacionadas con el desarrollo de sistemas de cómputo. Fue difícil que los estudiantes imaginarán los problemas relacionados con el desarrollo de sistemas de cómputo industriales sin haber participado en alguno de estos desarrollos. Este problema se trató de solucionar incorporando en cada equipo a un estudiante con experiencia industrial. Afortunadamente, en el curso se disponía de 2 estudiantes que compartieron sus experiencias con los otros miembros.

- Se requiere cierta habilidad en la representación de los diseños arquitectónicos. Este problema se observó cuando los estudiantes expusieron sus resultados de ADD-A, un ejemplo de estos problemas se observó cuando los estudiantes representaron elementos dinámicos con representaciones estáticas.

Creemos que los puntos anteriores pueden tratarse desde el momento en que se realiza la capacitación en las adaptaciones que se proponen en este trabajo. Esta capacitación puede realizarse de forma iterativa incrementando, en cada iteración, la complejidad de los requerimientos de diseño. Pensamos que este enfoque puede ayudar a solucionar los problemas observados debido a que:

- Los involucrados en el desarrollo pueden hacerse de conocimientos teóricos en cada una de las iteraciones. Al término de cada una de las iteraciones, en AEM, se pueden realizar observaciones sobre otros conceptos además de los tratados en el diseño actual que pueden ayudarlos a satisfacer a sus motivaciones arquitectónicas de forma más adecuada.

- Los involucrados en el desarrollo pueden refinar sus conocimientos prácticos en cada una de las iteraciones. Durante una iteración, los involucrados en el desarrollo aplican de forma práctica los conceptos necesarios para realizar sus diseños, al término de una iteración los involucrados en el desarrollo son evaluados para verificar si la forma en que aplicaron esos conceptos fue la adecuada.

- Al término de cada iteración, los involucrados en el diseño arquitectónico exponen su documentación preliminar con lo cual otros involucrados les pueden realizar observaciones acerca de la forma en cómo representan a los elementos arquitectónicos que identificaron.

Adicionalmente, con respecto a los comentarios recibidos por parte de los estudiantes, se puede comentar que la creación de las guías fue un gran apoyo para el desarrollo del diseño arquitectónico y les permitió tener conocimiento de que actividades iba a desarrollar cada uno de los integrantes para de esta forma realizar una calendarización y división del trabajo. Las plantillas y listas de verificación (al igual que las guías) fueron útiles para los estudiantes y nosotros debido a que era relativamente fácil revisar el avance de cada uno de los equipos.

Es necesario mencionar que los experimentos fueron realizados sólo en el marco de las adaptaciones de las actividades de Desarrollo de requerimientos y Diseño a alto nivel de TSPi. Las actividades previas y posteriores a estas 2 no fueron realizadas por motivos de tiempo.

Por último, los resultados obtenidos nos permiten pensar que los métodos planteados en este trabajo se podrían transferir a las organizaciones de desarrollo con equipos de desarrollo pequeños debido a que se pueden realizar por personal con conocimientos básicos de procesos y es proporcionado el material de apoyo suficiente para su realización.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Síntesis

En este proyecto de investigación se propone una adaptación a los métodos de desarrollo arquitectónico del SEI al contexto de equipos de desarrollo pequeños. Para está adaptación se crearon las guías, plantillas y listas de verificación de apoyo al proceso propuesto, el formato en el cual se documentaron las guías se inspira en la forma de documentación de actividades que propone el Proceso de Software Personal (PSP). Además, estas adaptaciones se validaron mediante la realización del diseño arquitectónico de un sistema real cuyos objetivos de negocio se satisfacían con el uso de ASOA. Adicionalmente, ASOA fue caracterizado y diferenciado de los otros 2 niveles de aplicación SOA encontrados en la literatura consultada, identificando para ASOA los elementos que sirven como apoyo en la realización de cualquier proceso de desarrollo arquitectónico basado en escenarios (tablas de generación de atributos de calidad, catálogo de tácticas y patrones arquitectónicos y los objetivos de negocio que motivan su uso).

Finalmente, se propuso una integración de las adaptaciones propuestas en este trabajo con TSPi, esto con la finalidad de brindar un proceso de desarrollo completo orientado a equipos de desarrollo pequeños.

7.2. Conclusiones

En este proyecto se definió un conjunto de adaptaciones a los métodos de desarrollo arquitectónico del SEI para que estos funcionen de forma coordinada y, de forma breve, se presentó como estas adaptaciones se pueden incorporar en un proceso de desarrollo orientado a equipos de desarrollo pequeños (TSPi). En estas adaptaciones se logró mantener el espíritu de cada uno de los métodos aligerando los artefactos que produce y las actividades que cada uno comprende. Estas adaptaciones se realizaron considerando que equipos de desarrollo pequeños realizan sistemas de complejidad media a baja y sin riesgos que involucren pérdidas financieras o de vidas humanas (en caso contrario es recomendable realizar análisis minuciosos).

Para realizar la evaluación de las adaptaciones propuestas se eligió un caso de estudio real cuyos objetivos de negocio se satisfacían con el uso de ASOA. Como apoyo para la evaluación de las adaptaciones se creó un conjunto de elementos arquitectónicos (catálogo de tácticas y patrones arquitectónicos y tablas de generación de escenarios de atributos de calidad) que son necesarios para soportar la ejecución de los métodos en el contexto de la creación de aplicaciones basadas en el enfoque ASOA, además, se definió un proceso genérico que permite definir estos elementos arquitectónicos para otros contextos diferentes de ASOA.

Finalmente, se describió una evaluación preliminar con estudiantes de maestría, está evaluación arrojó resultados que permiten pensar que es altamente factible que gente con poca experiencia se beneficie de realizar las adaptaciones propuestas en este trabajo dado que les permite crear diseños arquitectónicos de calidad aceptable desde sus primeros intentos.

7.3. Trabajo Futuro

La madurez de cualquier metodología está relacionada con las métricas que se puedan colectar de la misma, en este sentido en la experimentación que se realizó para evaluar a las adaptaciones propuestas no se lograron colectar las métricas suficientes. En un futuro se considera que colectar las siguientes métricas puede ser útil para mejorar a las adaptaciones propuestas en este trabajo:

- Facilidad con la que las adaptaciones se implantan en un entorno de desarrollo.
- Tiempo de capacitación del personal en el proceso.
- Repetitividad de los resultados.
- Tiempo tomado en la realización del proceso.
- Previsibilidad en el tiempo de realización de las actividades de la metodología.

Cabe señalar que en esta etapa del proyecto aún no se establece la forma en que las métricas anteriores se colectarán, pero una vez que se obtenga esta información se podrá establecer una estrategia para mejorar a la metodología propuesta en este proyecto.

También sería interesante observar cómo funciona la integración de las adaptaciones propuestas con TSPi en un proyecto que considere a todas las actividades de este proceso de desarrollo. Un siguiente paso sería implantar estas adaptaciones en organizaciones de desarrollo con equipos de desarrollo pequeños, esto se piensa factible debido a que las adaptaciones se hicieron pensando en una versión simplificada de TSP (TSPi se emplea poco en el ámbito organizacional) y en los elementos que típicamente se pueden encontrar en el Proceso de Software Personal (PSP).

Por último, se podría realizar la identificación de los elementos arquitectónicos encontrados para ASOA (catálogo de tácticas y patrones arquitectónicos, objetivos de negocio y tablas de generación de escenarios de atributos de calidad) para el contexto de EmSOA y ESOA.

Apéndice A

Integración con TSPi y listas de verificación

Propósito		Guiar a un equipo a través del desarrollo e inspección de requerimientos para el ciclo 1 de un proyecto de desarrollo en equipo.
Criterios de entrada		- El equipo ha desarrollado una estrategia y plan de desarrollo. - Los estudiantes han leído el capítulo 6, las secciones de prueba del capítulo 9 y las necesidades del sistema.
General		El proceso de desarrollo de requerimientos produce la Especificación de Requerimientos de Software, la cual define: - Las funciones que el producto debe realizar. - **Los requerimientos no funcionales que el producto debe considerar.** - Las descripciones de los casos de uso considerando flujos normales y alternativos. El equipo debe ser cuidadoso al expandir los requerimientos - Sin experiencia con aplicaciones similares, funciones aparentemente simples pueden tomar más trabajo de lo que se espera. - Es recomendable agregar funcionalidad en pequeños incrementos - Si sobra tiempo, se agrega la funcionalidad faltante.
Paso	Actividades	Descripción.

1	Visión general del proceso de requerimientos	El instructor describe el proceso de requerimientos y sus productos - Cómo se realiza el proceso de requerimientos - Cómo se conduce y reporta la inspección de requerimientos
2	Revisión de las necesidades	El arquitecto conduce al equipo en la revisión de las necesidades del producto y le formula al instructor preguntas acerca de: - Las funciones que son realizadas por las diversas versiones del producto. - Cómo las funciones van a ser utilizadas. - **Los requerimientos no funcionales a alto nivel que son soportadas por las diferentes versiones del producto.**
3	Clarificación de las necesidades	El arquitecto realiza preguntas consolidadas al instructor, quien discute las respuestas con el equipo.
4	Tareas de requerimientos	El arquitecto conduce al equipo a través de: - El esbozo del documento de especificación de requerimientos y el trabajo para producirlo
5	Asignación de tareas	**El líder de proyecto ayuda a asignar las tareas de documentación de requerimientos funcionales y no funcionales a los miembros del equipo y** obtiene compromisos sobre la fecha en la cual completarán estas tareas.
6	Documentación de requerimientos funcionales	Cada miembro del equipo: - Produce y revisa su porción del documento de especificación de requerimientos. - Proporciona su parte al arquitecto.

7	**Documentación de requerimientos no funcionales**	**El arquitecto guía al equipo de desarrollo en la realización de un QAW-A para la captura de escenarios de atributos de calidad.** **- Al término del taller se entregan las salidas de QAW-A al arquitecto.** El arquitecto produce el borrador del documento de especificación de requerimientos.
8	Planeación de pruebas de sistema	El arquitecto guía al equipo en la producción y revisión del plan de pruebas de sistema. - **El plan debe basarse en como el sistema satisface a los escenarios de atributos de calidad capturados mediante QAW-A, además de cómo cubre con los requerimientos funcionales.**
9	Requerimientos e inspección del plan de pruebas de sistema	El administrador de proceso/calidad conduce al equipo a través de: - Inspección del borrador del documento de especificación de requerimientos y del plan de pruebas de sistema. - Identificación de preguntas y problemas. - Definir quiénes responderán cada pregunta y problema y cuándo lo harán. - Documentar la inspección en la forma INS.
10	Actualización de requerimientos	El arquitecto obtiene las secciones actualizadas del documento de especificación de requerimientos y: - Las combina en la versión final del documento de especificación de requerimientos. - Verifica la trazabilidad hacia las necesidades u otras fuentes.

11	Revisión del documento de especificación de requerimientos por el usuario	- El arquitecto provee una copia del documento de especificación de requerimientos final al instructor (o usuario) para su aprobación. - Después de la aprobación, el equipo arregla cualquier problema identificado.
12	Línea de base de los requerimientos	El administrador de soporte coloca la línea de base del documento de especificación de requerimientos.
Criterios de salida		- **Un documento de especificación de requerimientos completo e inspeccionado que contempla las salidas de QAW-A.** - Un plan de pruebas de sistema. - Una forma INS completa para la inspección de requerimientos. - Datos de tiempo, defectos y tamaño capturados en el sistema de soporte de TSPi. - Libreta del proyecto actualizada.

Tabla A.1: Integración de QAW-A con la guía REQ de TSPi [1]

Propósito	Guiar a un equipo a través del desarrollo e inspección de la especificación del diseño de software para un proyecto de desarrollo en equipo.
Criterios de entrada	- Un plan y estrategia de desarrollo - **Un documento de especificación de requerimientos completo e inspeccionado que contempla las salidas de QAW-A** - Los estudiantes deben leer el capítulo 7

[1] Los elementos en negritas representan modificaciones con respecto de la guía original de TSPi de desarrollo de requerimientos

General		El proceso de diseño produce la especificación del diseño de software (SDS), el cual define la estructura general del producto para el ciclo 1. - Componentes mayores del producto y su especificación de interfaz. - La asignación de casos de uso a componentes. - **La asignación de requerimientos no funcionales a componentes.** El SDS también específica: - Estándares para archivos y mensajes, definiciones y convenciones de nombrado. - Notaciones de diseño y estándares.
Paso	Actividades	Descripción.
1	Visión general del proceso de diseño	El instructor describe el proceso de diseño y sus productos. - Cómo se realiza el proceso de diseño y un ejemplo de SDS. - Cómo se conduce y reporta la inspección de diseño. - Estándares de diseño y convenciones.
2	**Elaboración del diseño**	**El arquitecto elabora el diseño del producto siguiendo la guía de ADD-A.**
3	**Evaluación del diseño**	**El arquitecto guía al equipo de desarrollo en la evaluación del diseño contra los requerimientos no funcionales mediante la guía de AEM.**
4	**Documentación del diseño**	**El arquitecto guía al equipo de desarrollo en la elaboración del SDS mediante la guía de VaB.**
5	Plan de pruebas de integración	El arquitecto guía al equipo en la producción y revisión del plan de pruebas de integración.

6	Diseño e inspección del plan de pruebas de integración	El administrador de proceso/calidad conduce al equipo a través de la inspección del SDS y del plan de pruebas de integración, de forma que: - Se cubre cada caso de uso y se referencia en el diseño **- Cada requerimiento no funcional es cubierto por el diseño** **- El SDS es consistente** *** Los elementos de diseño son empleados como se definen.** *** Las responsabilidades de los elementos no están dispersas en el documento.** - El diseño es completo y correcto. - El plan de pruebas de integración es el adecuado. - Cada problema es registrado y la responsabilidad de arreglarlo es asignada a un miembro del equipo. La inspección es documentada en la forma INS y los defectos son registrados en el LOGD.
7	Actualización de la línea de base	El administrador de soporte coloca la línea de base del SDS.
Criterios de salida		**- Un documento SDS completo, evaluado e inspeccionado.** **- Un plan de pruebas de integración.** - Los estándares de diseño y el glosario de nombres. - Las formas SUMP y SUMQ actualizadas. - La forma de inspección INS. - La libreta del proyecto actualizada.

Tabla A.2: Integración de ADD-A con la guía HLD de TSPi [2]

[2]Los elementos en negritas representan modificaciones con respecto de la guía original de TSPi de diseño

Propósito	Conducir de forma eficiente la creación de la documentación de iteraciones de ADD-A.	
General	- Verificar entrada por entrada en esta lista de verificación, no se recomienda verificar varias entradas de forma simultánea. - Cuando se complete la revisión de cada entrada, ésta debe ser marcada.	
Datos de la iteración	- Se presenta un número progresivo como identificador de iteración. - Se indica quiénes participan en el desarrollo de la iteración. - Se indica la hora de inicio y fin de la iteración * En su caso se indica la duración de las interrupciones.	
Elementos a descomponer	Se presenta una lista de al menos un elemento que corresponda al sistema en cuestión.	
Motivaciones arquitectónicas	Se presenta una lista de al menos 2 motivaciones arquitectónicas respetivas al proyecto en cuestión.	
Alternativas de solución	- Se presenta una lista que comprende tácticas y patrones arquitectónicos. - En cada patrón o táctica se definen atributos que permitan elegir cuál es la mejor opción.	
Patrones y/o tácticas arquitectónicas y decisiones de diseño	- Se presenta una lista de patrones y tácticas arquitectónicas que se emplean en el sistema en cuestión. - Para cada patrón o táctica se presenta una lista de decisiones de diseño que fueron tomadas.	

Instanciación y descripción de cada elemento	- Se presenta de forma gráfica o textual los elementos que resultaron de la descomposición en la iteración. - Se describe brevemente cuáles son las responsabilidades de cada elemento. - En caso de ser necesario se presentan diagramas que apoyen la comprensión de los elementos producto de la iteración.	
Interfaz de cada elemento	- Los elementos que así lo requieran, tienen asociada una lista con los nombres de las interfaces que implementan. - Las interfaces tienen asociados métodos con firma. - Cada método tiene asociado una descripción.	

Tabla A.4: Lista de verificación de documentación de iteraciones

Propósito	Conducir de forma eficiente el refinamiento de los escenarios más importantes encontrados con QAW-A.	
General	- Verificar entrada por entrada en esta lista de verificación, no se recomienda verificar varias entradas de forma simultánea. - Cuando se complete la revisión de cada entrada, ésta debe ser marcada.	
Identificador	Cada escenario refinado tiene asociado un identificador único en el proyecto.	
Escenarios	- Se toma en cuenta al menos un escenario crudo para ser refinado. - Cada escenario crudo es expresado en términos de estímulo, entorno y respuesta.	
Objetivos de negocio	Se tiene una lista de objetivos de negocio relacionadas con el escenario que se está refinando.	
Atributos de calidad	Se tiene una lista con los atributos de calidad relacionados con el escenario que se está refinando.	
Componentes de escenario	- El estímulo corresponde a una condición que el sistema debe considerar cuando llega a este. - La fuente del estímulo corresponde a la entidad que lo genera. - El entorno corresponde a las condiciones bajo las cuales el estímulo ocurre. - El artefacto corresponde a una entidad (o la totalidad) del sistema. - La respuesta corresponde a una actividad realizada después de que este llega al sistema. - La medida de la respuesta expresa una forma (de preferencia cuantificable) de medir la respuesta del sistema	
Preguntas y cuestiones	Se expresan inquietudes por parte de los participantes con respecto del escenario que se está refinando	

Tabla A.3: Lista de verificación para la plantilla de refinamiento de escenarios

Propósito	Conducir de forma eficiente la presentación de los resultados generados por AEM.	
General	- Verificar entrada por entrada en esta lista de verificación, no se recomienda verificar varias entradas de forma simultánea. - Cuando se complete la revisión de cada entrada, ésta debe ser marcada.	
Identificador	- Cada elemento tiene asociado un identificador único para el proyecto. - Mediante el identificador se puede determinar qué tipo de elemento es: * Riesgo (R) * No riesgo (NR) * Compromiso (C) * Punto sensible (PS)	
Descripción	Cada elemento tiene asociada una descripción conforme a su tipo	
Responsable	- En caso de no ser un no riesgo, el elemento tiene asociado a un responsable del seguimiento del elemento. - El responsable es miembro del equipo de desarrollo.	
Motivaciones arquitectónicas	Se presenta una lista de motivaciones arquitectónicas sobre las cuales se realiza la evaluación.	
Preguntas	- Se tiene una lista con al menos una pregunta. - La pregunta pertenece a algún modelo de evaluación.	
Respuesta	Se muestra una lista de respuestas a las preguntas anteriores.	
Referencia a elementos	Se muestra una lista de identificadores que referencian a elementos que estén relacionados con las preguntas que se realizan.	
Decisiones arquitectónicas	Se describen las decisiones de diseño involucradas en el escenario.	
Argumentos	Los análisis en los argumentos de diseño están relacionados con los elementos identificados en AEM.	
Diseño	En caso de ser necesario se incluye un bosquejo del diseño propuesto como solución.	

Tabla A.5: Lista de verificación de resultados de AEM

Apéndice B

Realización de las adaptaciones de los métodos del SEI

B.1. Documentación de algunos de resultados de QAW-A

Paso 7: Escenarios consolidados

Seguridad

1. SEC-1: Un usuario realiza una operación errónea o exitosa con el sistema después de haberse autentificado, se le agrega una entrada a la bitácora sin impactar al tiempo de respuesta del sistema.

2. SEC-2: Un usuario no autentificado ingresa 3 veces combinaciones de login y password erróneas. El sistema bloquea el acceso desde esa dirección IP por 15 min y se registra el intento de acceso en la bitácora.

3. SEC-3: Un usuario ingresa una combinación de login y password correcta al sistema, durante el envío de la petición de inicio de sesión una persona ajena al sistema mediante el uso de un sniffer intercepta la información del usuario. La información que la persona intercepta no se debe descifrar.

Desempeño

1. DES-1: Un usuario autentificado envía una petición al sistema en un momento de sobrecarga, el sistema procesa la petición en menos de 10 segundos.

2. DES-2: Un usuario desea graficar datos de un dispositivo para su análisis en un momento normal de operación, la gráfica se mostrará en un tiempo no mayor a 30 segundos.

3. DES-3: Dos usuarios desean modificar un elemento de la topología de forma simultánea. El sistema atiende la primera petición que recibe y envía una notificación al otro usuario para avisarle que el elemento está siendo usado en ese momento.

4. DES-4: Un administrador (o un desarrollador) agrega una nueva funcionalidad al sistema. Por cada funcionalidad incorporada al sistema el tiempo de reinicio del mismo no aumentará en más de 2 %.

Disponibilidad

1. DIS-1: Un dispositivo se avería en un momento normal de operación, la representación que tienen los clientes de la topología reflejará la falla del dispositivo en un tiempo no mayor a 20 segundos.

2. DIS-2: Un usuario intenta ingresar al sistema en un momento en que hay otros 50 usuarios interactuando con el mismo. La petición del usuario es rechazada.

3. DIS-3: El sistema intenta enviar una petición de configuración a uno de los dispositivos en un momento de operación normal, pero ocurre una falla de red entre el dispositivo y el sistema antes del envío. El sistema realiza en total 3 intentos de envío de la configuración y si sigue obteniendo errores, marca al dispositivo como inaccesible y notifica al usuario.

4. DIS-4: El cliente envía una petición al sistema en un momento en que ocurre una falla de red entre el cliente y el sistema. Se hacen 3 intentos de envío de la petición y si fallan se notifica al usuario de un error, además, se envía una notificación al administrador del sistema.

5. DIS-5: Un error que origina la caída del sistema ocurre mientras el mismo procesa una petición de usuario de modificación de la topología. Cuando el sistema se recupera, la topología se encuentra en un estado coherente pero la petición del usuario se pierde.

6. DIS-6: Se reinicia el sistema en un momento en que han ocurrido cambios a nivel de los dispositivos reflejados en la topología. El sistema actualiza el estado de la topología al iniciar.

Modificabilidad

1. MOD-1: Un administrador agrega un dispositivo que usa un protocolo soportado por el sistema mientras este se encuentra en ejecución. El sistema no es reiniciado y el dispositivo es incorporado al sistema.

2. MOD-2: Un desarrollador distinto al fabricante del sistema desea crear una nueva funcionalidad para el mismo. La funcionalidad se debe poder incorporar al sistema sin necesidad de recompilación ni de conocer el código fuente de la base.

3. MOD-3: Un desarrollador crea una nueva versión de la interfaz (API) de un componente de la aplicación. Al introducir el componente al sistema, los componentes que dependían de la versión previa no se ven afectados por este cambio.

4. MOD-4: Un administrador agrega una funcionalidad/manejador de dispositivo al sistema mientras este se encuentra en ejecución. El sistema no requiere ser reiniciado y la funcionalidad/manejador es incorporada y operacional.

5. MOD-5: Un administrador quita una funcionalidad/manejador de dispositivo del sistema mientras este se encuentra en ejecución. El sistema no requiere ser reiniciado y los elementos que dependían del elemento/manejador eliminado se adaptan de forma automática a este hecho.

Usabilidad

1. USA-1: Un usuario autorizado desea realizar la baja de un dispositivo de la red (u otra operación de modificación de la topología). El sistema debe solicitar una confirmación de la operación antes de su realización.

2. USA-2: Un usuario desea ubicar dentro de la topología un dispositivo, momentos después de que el dispositivo falla. El usuario ubica al dispositivo siguiendo un código de colores que indica la ruta del dispositivo con fallas en la topología.

3. USA-3: Un usuario desea conocer la acción que realiza una funcionalidad del sistema. El sistema provee una descripción breve de la acción realizada por la función.

4. USA-4: Un usuario registrado en el sistema olvida su password cuando intenta ingresar al mismo. El sistema ofrece una opción para enviar su password a la cuenta de correo electrónico que dio de alta al momento de registrarse en el sistema.

Facilidad de pruebas

1. PRU-1: Un desarrollador de prueba unitaria desea probar un elemento del sistema con dependencia hacia otros elementos. La prueba se puede crear sin la necesidad de que los elementos reales de los cuales depende el elemento estén presentes y sin conocer su implementación interna.

2. PRU-2: Un probador desea realizar pruebas de integración sobre los componentes de la aplicación base del sistema. Estas pruebas se pueden realizar sin la necesidad de que se utilicen extensiones de la aplicación.

Paso 9: Refinamiento de escenarios

<table>
<tr><td colspan="3">Identificador: DESREF-1</td></tr>
<tr><td colspan="2">Escenario(s):</td><td>Un administrador (o un desarrollador) agrega una nueva funcionalidad al sistema. Por cada funcionalidad incorporada al sistema el tiempo de reinicio del mismo no aumentará en más de 2 %.</td></tr>
<tr><td colspan="2">Objetivos de negocio:</td><td>- El sistema debe permitir crear en tiempo limitado versiones de sí mismo adaptadas a las necesidades de diversos clientes
- El sistema debe permitir agregar/quitar funcionalidades del sistema a los usuarios finales mientras este se encuentra en funcionamiento</td></tr>
<tr><td colspan="2">Atributos de calidad relevantes:</td><td>Desempeño</td></tr>
<tr><td rowspan="6">Componentes del escenario:</td><td>Estímulo:</td><td>Agrega una funcionalidad al sistema</td></tr>
<tr><td>Fuente del estímulo:</td><td>Un administrador o un desarrollador</td></tr>
<tr><td>Entorno:</td><td>El sistema es reiniciado</td></tr>
<tr><td>Artefacto:</td><td>Sistema</td></tr>
<tr><td>Respuesta:</td><td>Cuando el administrador (o desarrollador) reinicia el sistema</td></tr>
<tr><td>Medida de la respuesta:</td><td>Por cada funcionalidad incorporada al sistema el tiempo de reinicio del mismo no aumentará en más de 2 %</td></tr>
<tr><td colspan="2">Preguntas:</td><td>- ¿Cuánto es el tiempo máximo de reinicio que un usuario del sistema tolera?
- ¿Será necesario notificar al usuario que tiene funcionalidades que ya no usa y que podría eliminar con el fin de optimizar el tiempo de reinicio?</td></tr>
<tr><td colspan="2">Cuestiones:</td><td></td></tr>
</table>

Identificador: MODREF-1		
Escenario(s):		Un administrador agrega un dispositivo que usa un protocolo soportado por el sistema mientras este se encuentra en ejecución. El sistema no es reiniciado y el dispositivo es incorporado al sistema
Objetivos de negocio:		- El sistema debe permitir agregar/quitar manejadores de dispositivos a los usuarios finales mientras el sistema se encuentra en funcionamiento - El sistema debe permitir crear en tiempo limitado versiones de sí mismo adaptadas a las necesidades de diversos clientes
Atributos de calidad relevantes:		Modificabilidad
Componentes del escenario:	Estímulo:	Agrega un dispositivo que usa un protocolo soportado por el sistema
	Fuente del estímulo:	Un administrador
	Entorno:	A tiempo de ejecución
	Artefacto:	Sistema
	Respuesta:	El dispositivo es incorporado
	Medida de la respuesta:	El sistema no requiere que sea reiniciado
Preguntas:		- ¿Qué acción realizar cuando se intenta agregar el dispositivo número 201?
Cuestiones:		

Identificador: MODREF-2		
Escenario(s):		Un desarrollador distinto al fabricante del sistema desea crear una nueva funcionalidad para el sistema. La funcionalidad se debe poder incorporar al sistema sin necesidad de recompilación ni de conocer el código fuente de la base
Objetivos de negocio:		- El sistema debe permitir agregar nuevas funcionalidades creadas por otras organizaciones - El sistema debe permitir crear en tiempo limitado versiones de sí mismo adaptadas a las necesidades de diversos clientes
Atributos de calidad relevantes:		Modificabilidad
Componentes del escenario:	Estímulo:	Desea crear una nueva funcionalidad para la aplicación
	Fuente del estímulo:	Un desarrollador distinto al fabricante
	Entorno:	A tiempo de desarrollo
	Artefacto:	El sistema
	Respuesta:	La funcionalidad se debe poder incorporar
	Medida de la respuesta:	Sin la necesidad de recompilar al sistema ni conocer el código de la base
Preguntas:		
Cuestiones:		

Identificador: MODREF-3		
Escenario(s):		Un desarrollador crea una nueva versión de la interfaz (API) de un componente de la aplicación. Al introducir el componente al sistema, los componentes que dependían de la versión previa no se ven afectados por este cambio
Objetivos de negocio:		- El sistema debe permitir agregar nuevas funcionalidades creadas por otras organizaciones - El sistema debe permitir crear en tiempo limitado versiones de sí mismo adaptadas a las necesidades de diversos clientes - El sistema permitirá actualizar el API de las funcionalidades que lo integran sin afectar a funciones que dependan de versiones anteriores del API actualizado
Atributos de calidad relevantes:		Modificabilidad
Componentes del escenario:	Estímulo:	Se crea una nueva versión del API
	Fuente del estímulo:	Un desarrollador
	Entorno:	A tiempo de desarrollo
	Artefacto:	La interfaz de un componente
	Respuesta:	La actualización de la interfaz es posible
	Medida de la respuesta:	Sin afectar a los componentes que dependían de la versión anterior de la interfaz
Preguntas:		- ¿Cuánto tiempo se dará soporte a las versiones anteriores de una interfaz?
Cuestiones:		

Identificador: MODREF-4		
Escenario(s):		Un administrador agrega una funcionalidad/manejador de dispositivo del sistema mientras este se encuentra en ejecución. El sistema no requiere que sea reiniciado y la funcionalidad/manejador es incorporada y operacional
Objetivos de negocio:		El sistema debe permitir agregar/quitar funcionalidades del sistema a los usuarios finales mientras este se encuentra en funcionamiento
Atributos de calidad relevantes:		Modificabilidad
Componentes del escenario:	Estímulo:	Se agrega una funcionalidad/manejador de dispositivo
	Fuente del estímulo:	Un administrador
	Entorno:	A tiempo de ejecución
	Artefacto:	Sistema
	Respuesta:	La funcionalidad/manejador de dispositivo es incorporada al sistema
	Medida de la respuesta:	La funcionalidad/manejador de dispositivo es operacional sin que se requiera que el sistema se reinicie
Preguntas:		- ¿Qué acción se debe tomar en caso de que las dependencias de la funcionalidad/manejador no se puedan satisfacer? - ¿Qué acción realizar cuando las dependencias sean satisfechas?
Cuestiones:		

Identificador: MODREF-5		
Escenario(s):		Un administrador quita una funcionalidad/manejador de dispositivo del sistema mientras este se encuentra en ejecución. El sistema no requiere que sea reiniciado y los elementos que dependían del elemento/manejador eliminado se adaptan de forma automática a este hecho
Objetivos de negocio:		- El sistema debe permitir crear en tiempo limitado versiones de sí mismo adaptadas a las necesidades de diversos clientes - El sistema debe permitir agregar/quitar funcionalidades del sistema a los usuarios finales mientras este se encuentra en funcionamiento
Atributos de calidad relevantes:		Modificabilidad
Componentes del escenario:	Estímulo:	Se elimina una funcionalidad/manejador de dispositivo
	Fuente del estímulo:	Un administrador
	Entorno:	A tiempo de ejecución
	Artefacto:	Sistema
	Respuesta:	La funcionalidad/manejador de dispositivo se retira del sistema
	Medida de la respuesta:	Los componentes que dependían de la funcionalidad/manejador de dispositivo se adaptan de forma automática a su partida sin la necesidad de que el sistema sea reiniciado
Preguntas:		
Cuestiones:		

Identificador: PRUREF-1		
Escenario(s):		Un probador desea realizar pruebas de integración sobre los componentes de la aplicación base del sistema. Estas pruebas se pueden realizar sin la necesidad de que se utilicen extensiones de la aplicación.
Objetivos de negocio:		
Atributos de calidad relevantes:		Facilidad de pruebas
Componentes del escenario:	Estímulo:	Se desea realizar pruebas de integración
	Fuente del estímulo:	Un probador
	Entorno:	A tiempo de desarrollo
	Artefacto:	La aplicación base
	Respuesta:	Las pruebas se pueden realizar
	Medida de la respuesta:	Sin la necesidad de que estén presentes las extensiones de la aplicación base
Preguntas:		
Cuestiones:		

Paso 10: Priorización de motivaciones arquitectónicas

Prioridad Alta

- Administrar dispositivos
- Configuración de dispositivos
- Consultar estado de dispositivos
- Crear reporte de estado de dispositivos
- Recibir notificaciones de dispositivos
- Colectar datos de los dispositivos
- Acceso de usuarios
- Los dispositivos son administrados de forma centralizada
- Los usuarios finales pueden agregar/quitar funcionalidades a tiempo de ejecución
- Los usuarios finales pueden agregar/quitar manejadores de dispositivos a tiempo de ejecución
- MODREF-1
- MODREF-3
- MODREF-4
- MODREF-5
- PRUREF-1

Prioridad Media

- Administración de la representación gráfica
- DESREF-1
- MODREF-2

B.2. Algunos ejemplos de documentación de iteraciones de ADD-A

Datos de iteración:

Numero de iteración:	3
Participantes:	Ismael Nuñez

Elemento(s) a descomponer:

1. Extensiones de funcionalidad/dispositivos
2. Elementos del microkernel

Lista de motivaciones arquitectónicas y restricciones considerados para la iteración:

1. DESREF-1
2. MODREF-3
3. PRUREF-1

Tabla de alternativas de solución

Alternativa de solución	**Motivante(s) que satisface**	**Elementos de discriminación**
Objetos de extensión / extensión de interfaz	- MODREF-3	Este patrón permite que un componente implemente distintas versiones de una misma interfaz con lo que actualizaciones futuras a está no afectan a otros componentes que usan versiones previas.

Adaptadores	- MODREF-3	Este patrón permite adaptar una versión de una interfaz en otra versión, con el inconveniente de la necesidad de mantener otro objeto
Adquisición tardía	- DESREF-1	Este patrón permite solo cargar la información mínima necesaria para emplear una funcionalidad, con lo que el tiempo de reinicio de la aplicación se reduce. Cuando la información es requerida se carga en memoria
Adquisición parcial	- DESREF-1	Este patrón carga por etapas las partes de los recursos hasta que el recurso se cargue en memoria.
Separación de interface implementación	- PRUREF-1	Esta táctica permite realizar pruebas a solo un conjunto específico de componentes de la aplicación base.
Mantener la coherencia semántica	- PRUREF-1	En esta táctica las operaciones realizadas por la aplicación base se deben relacionar solo con las funcionalidades comunes de todas las distribuciones posibles y no depender de un conjunto particular de extensiones.
Manejo de excepciones	- PRUREF-1	En esta táctica la aplicación base debe manejar las excepciones lanzadas por ella y por las extensiones para que de esta forma se evite que la aplicación se caiga (Programación defensiva).

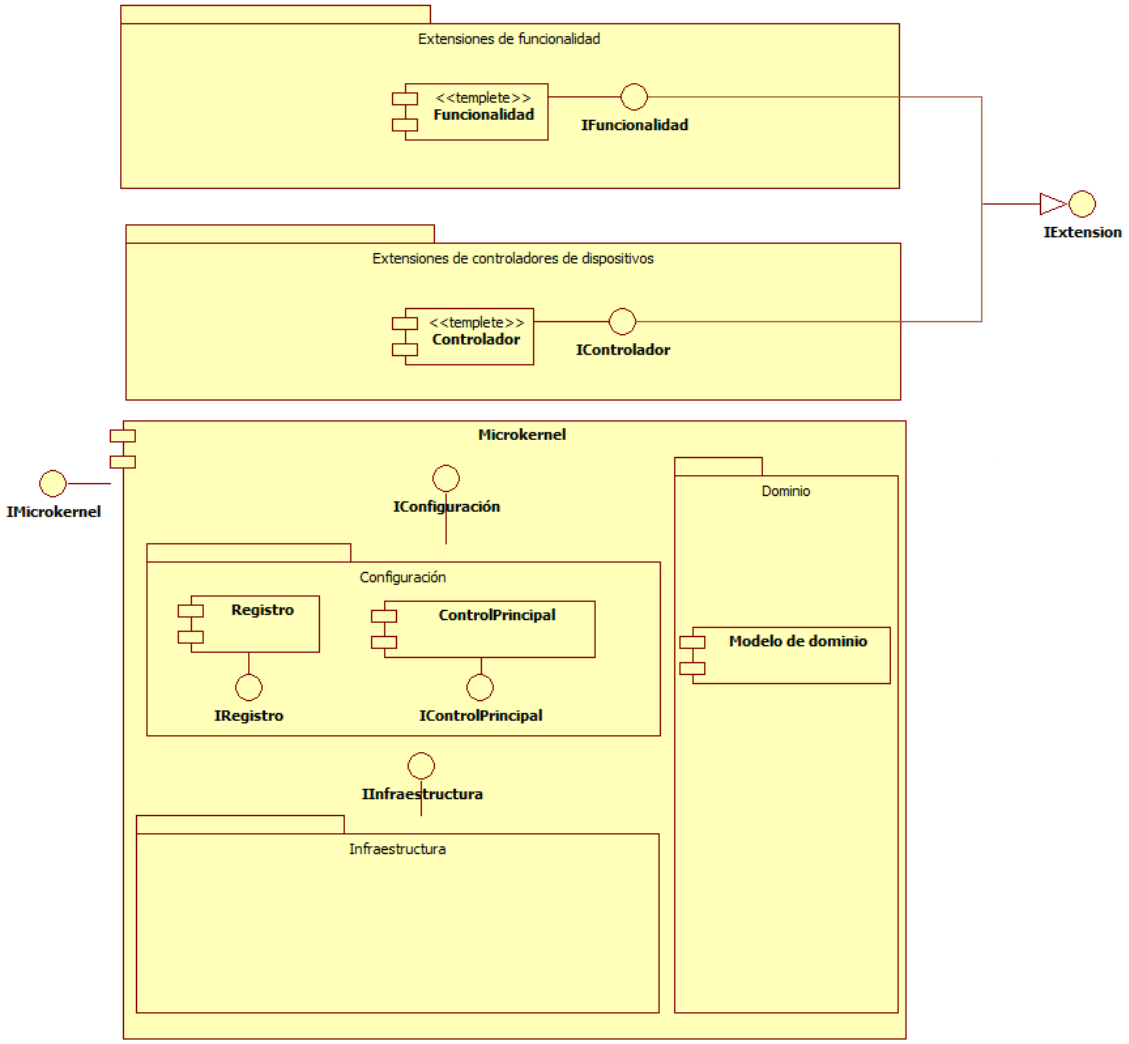
Descriptores e interfaces de servicio	- PRUREF-1	Esta táctica permite conocer de antemano como deben interactuar el cliente y el proveedor de servicio.

Tabla de patrones arquitectónicos:

Nombre del patrón / táctica	**Decisiones de diseño**
Objetos de extensión/Interfaces de extensión	- Las interfaces de las extensiones extenderán a la interfaz IExtensión la cual tendrá el siguiente método: * GetExtensión(Int versión): Retorna la interfaz con la versión dada
Adquisición tardía	Del lado cliente solo se mostrarán el nombre, icono y una breve descripción de las funcionalidades, cuando una funcionalidad sea lanzada se creará un comando que le indicara al servidor que acción realizar. Del lado servidor las funcionalidades se instancian hasta que se ejecute el comando que las activa. En caso de que la funcionalidad ya haya sido instanciada solo ejecutará el comando del cliente.
Comando	Permite desacoplar al emisor del comportamiento del receptor y además evita el efecto dominio hacia los clientes cuando las interfaces de servicio cambian
Separación de interface e implementación	Cada componente del servidor, exceptuando a los elementos del modelo de dominio, deberá proveer una interface.
Mantener la coherencia semántica	Cada componente del sistema estará dedicado a realizar solo un conjunto relacionado de actividades

Manejo de excepciones	La aplicación base deberá validar cada tipo de excepción que las extensiones puedan lanzar, con el fin de que la aplicación base no se caiga durante la ejecución
Descriptores e interfaces de servicio	Todos los proveedores de servicio deben proveer una descripción de la forma en que los clientes interactúan con ellos y materializar esta descripción por medio de una interfaz

Instanciación de elementos:

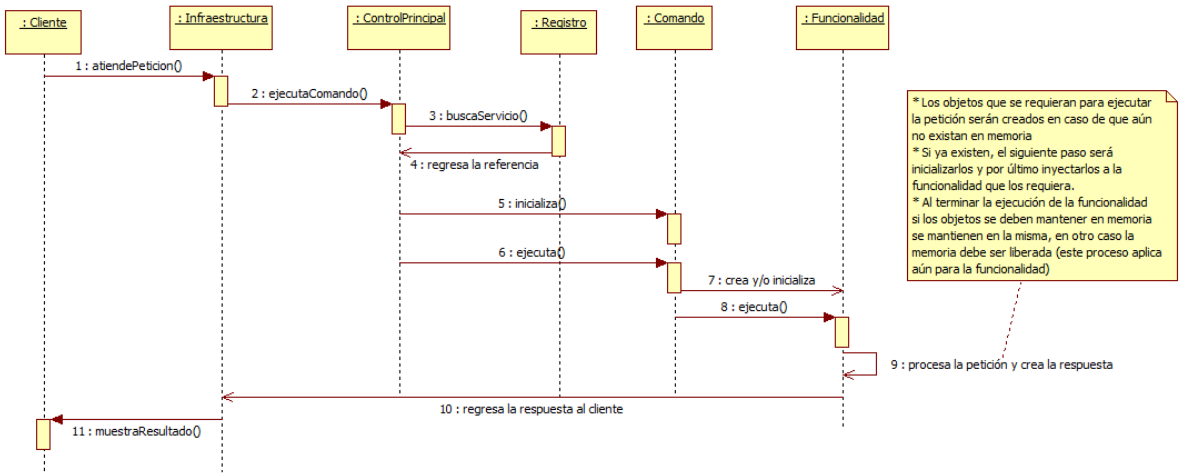


1. IExtensión: Permite conocer las diferentes versiones de la interfaz que implementa una extensión.

2. Microkernel: Cada elemento del Microkernel se deberá programar empleando un enfoque defensivo, esto con el fin de evitar que una extensión pueda hacerlo fallar.

3. Cada elemento del sistema (microkernel y extensiones) deben realizar solo un conjunto relacionado de acciones.

4. El patrón de adquisición tardía se muestra en el siguiente diagrama:



En este caso cuando el usuario selecciona la funcionalidad que requiere, el cliente envía la petición al servidor junto con la información necesaria para su ejecución. En el servidor la capa de infraestructura recibe la petición y se crea y ejecuta un comando, el ControlPrincipal busca en el registro a la o las funcionalidades que requiere el comando, las crea en caso de ser necesario y posteriormente las inyecta en el comando. El comando inicializa a la funcionalidad y por último, la activa para que procese la información. En la funcionalidad los objetos que requiere son creados en caso de ser necesario, después son inyectados a la funcionalidad. La funcionalidad procesa la petición y crea la respuesta, la cual se envía a la capa de infraestructura para que esta la envié al cliente. Cuando la funcionalidad termina su ejecución libera a todos los recursos que haya empleado (memoria, en el caso de que se

requiera crear una nueva funcionalidad por cada petición y otros objetos, estos objetos de igual forma podrán liberar la memoria que emplean o seguir en memoria si es necesario).

Interfaces de los elementos hijo:

Nombre de interfaz	**Componente al que corresponde**	**Métodos principales**
IExtensión	Todas las extensiones	- GetExtensión(int versión): Este método regresa la versión específica de la interfaz elegida, en caso de que la versión no exista lanzará una excepción.
IFuncionalidad	Todas las extensiones de funcionalidad	- Inicia(): Inicializa a la extensión - Ejecuta(Comando c): Realiza el procesamiento de la solicitud y regresa una respuesta. La funcionalidad debe liberar todos los recursos que ya no utilice con el fin de que otros elementos puedan usarlos.
IControlador	Todas las extensiones de control de dispositivo	- Inicia(): Inicializa al componente - PruebaConexión(): Prueba si el dispositivo es disponible o no. Este método es útil cuando se requiere saber si el dispositivo está disponible para realizar operaciones sobre él. - Ejecuta(Comando c): Realiza la ejecución de la petición sobre el dispositivo. El controlador debe liberar todos los recursos que ya no utilice con el fin de que otros elementos puedan usarlos.
IMicrokernel, IInfraestructura e IConfiguración	Elementos del microkernel	- Estas interfaces representan el hecho de que los elementos del microkernel deberán exponer una interfaz para poder ser usados

Datos de iteración:

En esta iteración se consolidarán todas las decisiones de diseño tomadas durante ADD-A y se mostrarán los bosquejos de las vistas arquitectónicas.

Numero de iteración:	6
Participantes:	Ismael Nuñez

Elemento(s) a descomponer:

- Todos los elementos ya han sido descompuestos

Lista de motivaciones arquitectónicas y restricciones considerados para la iteración:

- Todos

Tabla de alternativas de solución:

En esta iteración ya no son consideradas alternativas debido a que solo se consolidan las iteraciones anteriores

Tabla de patrones arquitectónicos:

En la siguiente tabla los elementos en negrita fueron obtenidos del catálogo de tácticas y patrones arquitectónicos propuesto en este trabajo.

Nombre del patrón / táctica + Referencia	**Decisiones de diseño**
Cliente-Servidor	El cliente y el servidor son entidades separadas que se comunicaran a través de una red de computadoras, el servidor a su vez es la única entrada a través de la cual los usuarios tendrán acceso a los dispositivos en la red.
Cliente ligero	El usuario accede al sistema a través de una navegador web. La incorporación de nuevas funcionalidades a tiempo de ejecución se limita a registrar la URL de la funcionalidad.

Modelo de Dominio [Buschmann2007]	Las abstracciones que se consideraran en el modelo de dominio de este sistema son Regiones, Nodos, Notificaciones y Usuarios
Microkernel [Buschmann2007]	- Existirán 2 tipos de extensiones: * Extensiones de funcionalidad: Estas extensiones permitirán añadirle al microkernel nuevas funcionalidades * Extensiones de controladores de dispositivo: Estas extensiones permitirán añadirle al microkernel nuevos protocolos de conexión con dispositivos.
Lookup [Buschmann2007]	Este patrón permitirá descubrir los servicios registrados en el microkernel
Auto-adaptabilidad	Esta táctica permite a las extensiones adaptarse de forma automática a la partida o registro de otras extensiones de las cuales se puede depender
Manejo de la tasa de notificaciones	El registro solo le enviará a las extensiones las notificaciones que le sean relevantes
Objetos de extensión/Interfaces de extensión	Las interfaces de las funcionalidades extenderán la interfaz IExtensión la cual tendrá el siguiente método: * GetExtensión(Int versión): Retorna la interfaz con la versión dada
Adquisición tardía [Buschmann2007]	Del lado cliente solo se mostrara el nombre, icono y una breve descripción de las funcionalidades, cuando una funcionalidad sea lanzada se creara un comando que le indicará al servidor que acción realizar. Del lado servidor las funcionalidades se instancian hasta que llegue el comando que las activa. En caso de que la funcionalidad ya haya sido instanciada solo se ejecutará el comando del cliente.
Separación de interface e implementación [Bass2003]	Cada componente del servidor, exceptuando a los elementos del modelo de dominio, deberá implementar una interface.

Mantener la coherencia semántica [Bass2003]	Cada componente del sistema estará dedicado a realizar solo un conjunto relacionado de actividades.
Manejo de excepciones	La aplicación base deberá validar cada tipo de excepción que las extensiones puedan lanzar, con el fin de que la aplicación base no se caiga durante la ejecución.
Descriptores e interfaces de servicio	Todos los proveedores de servicio deben proveer una descripción de la forma en que los clientes interactúan con ellos y materializar esta descripción por medio de una interfaz.
Objetos de dominio [Buschmann2007]	Todos los componentes del sistema realizan un conjunto coherente de actividades
Transactions scripts	- En las extensiones se colocará la lógica de negocio que modifica al modelo de dominio. - Estos objetos se inicializarán con los objetos de los cuales dependa. - Las reglas de negocio se delegarán a los objetos de dominio del problema.
DAO [Buschmann2007]	La capa de acceso a datos está dividida en la aplicación base y las funcionalidades, esto con el fin de que se pueda soportar el almacenamiento de atributos específicos de los dispositivos que se agreguen en el futuro.
Sincronización [Buschmann2007]	Se debe asegurar el acceso concurrente a los objetos del modelo de dominio y la administración de las tareas de tiempo que ejecutarán algunas de las extensiones o aplicación base por medio de candados de acceso

MVC [Buschmann2007]	- Las vistas se encuentran en las extensiones de funcionalidad. - Los controladores de vistas se encuentran en las extensiones de funcionalidad. - El modelo de dominio se encuentra dividido en la aplicación base y las extensiones de funcionalidad.
Autenticación [Bass2003]	Los usuarios tendrán asignado un nombre de usuario, tipo y password mediante los cuales podrán o no tener acceso a las funcionalidades y extensiones de la aplicación.
Comando [Buschmann2007]	Permite desacoplar al emisor del comportamiento del receptor y además evita el efecto dominó hacia los clientes cuando las interfaces de servicio cambian.
Singleton	Este patrón evita que existan múltiples instancias de una clase o componente, con lo que se optimiza el consumo de memoria y se facilita el manejo de los hilos de ejecución.

Instanciación de elementos:

Vistas de Módulos

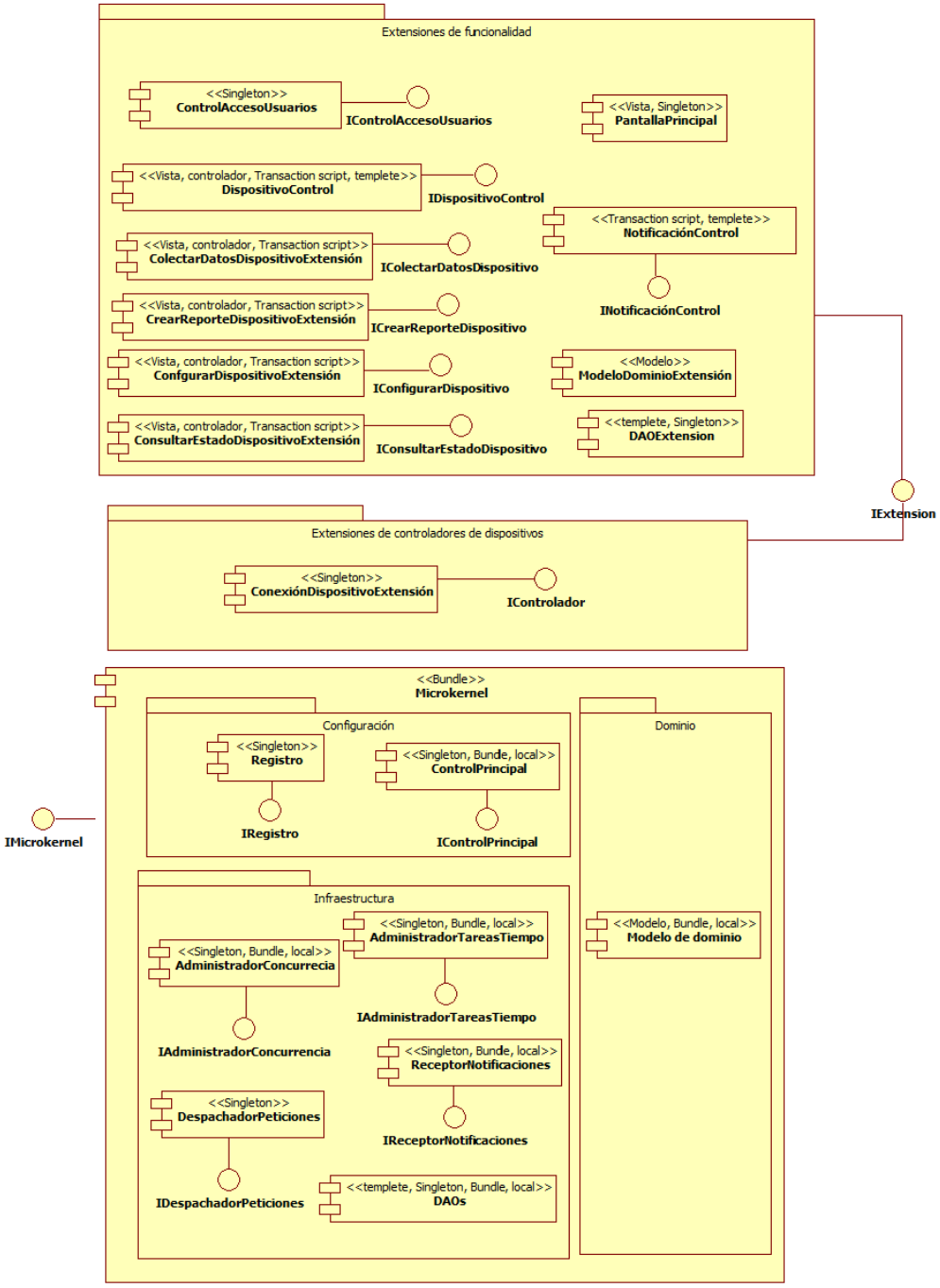


Figura B.1: Módulos de la arquitectura

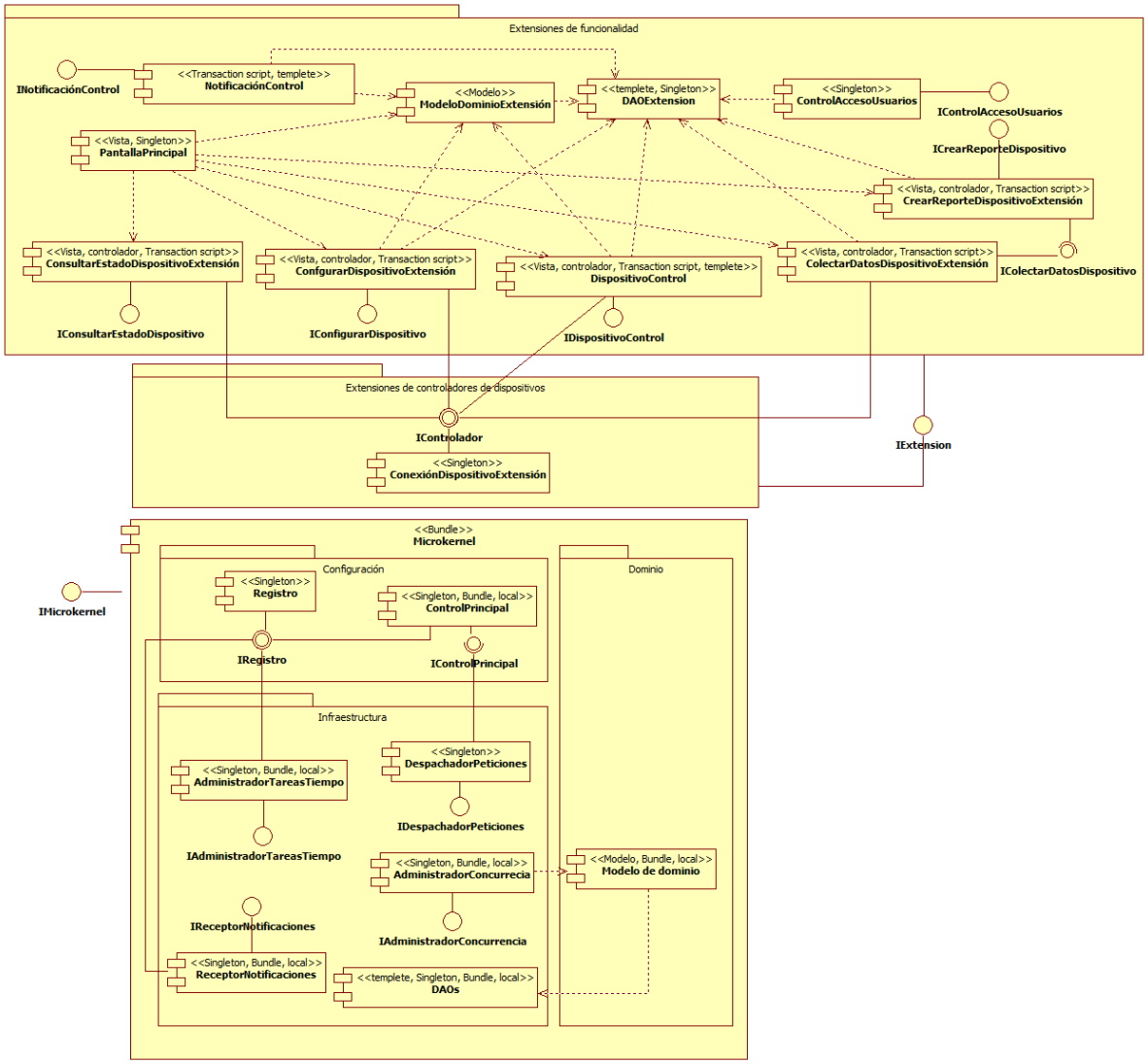


Figura B.2: Relaciones de uso entre los módulos de la arquitectura (entre extensiones o aplicación base)

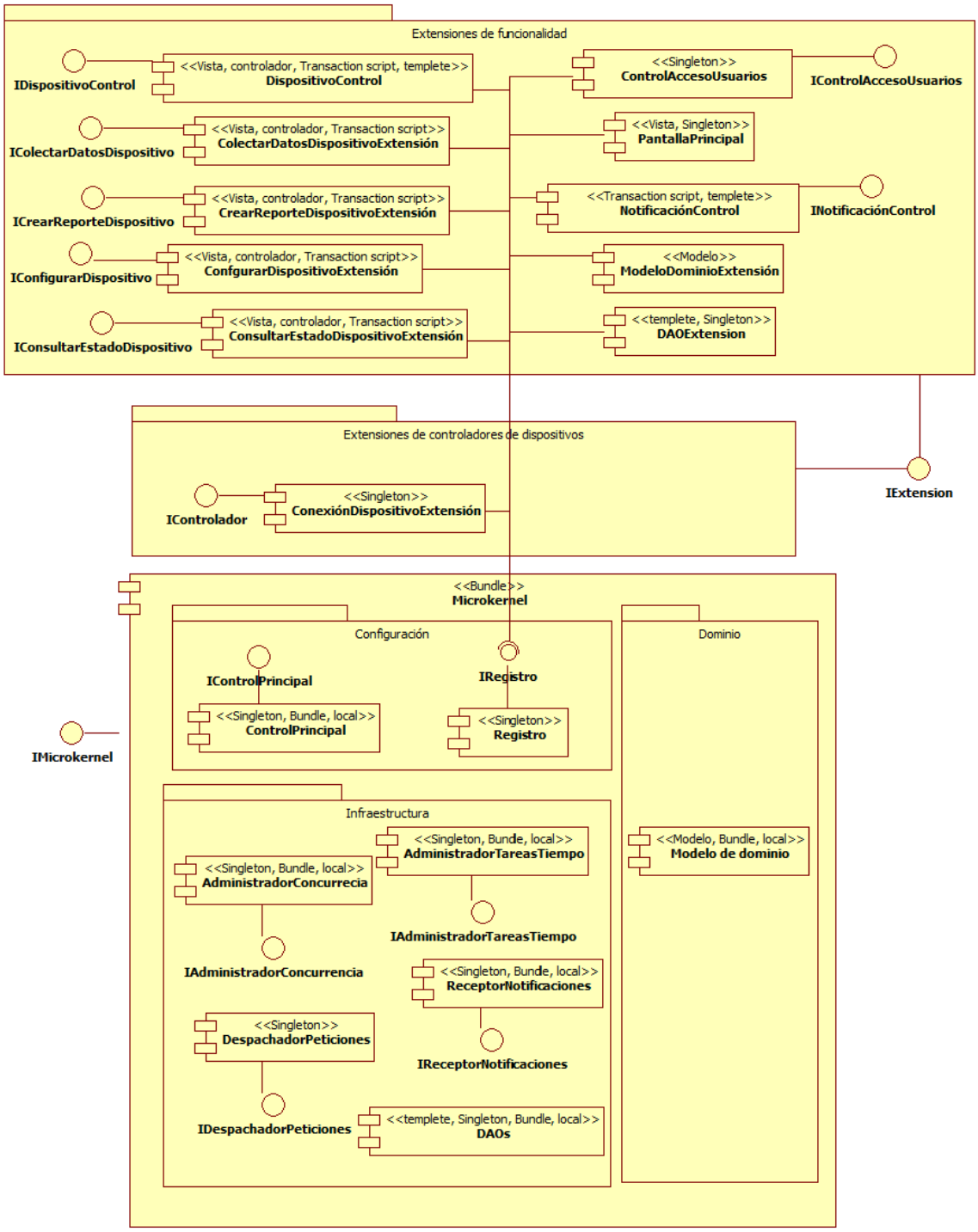


Figura B.3: Relaciones de uso entre las extensiones y el registro

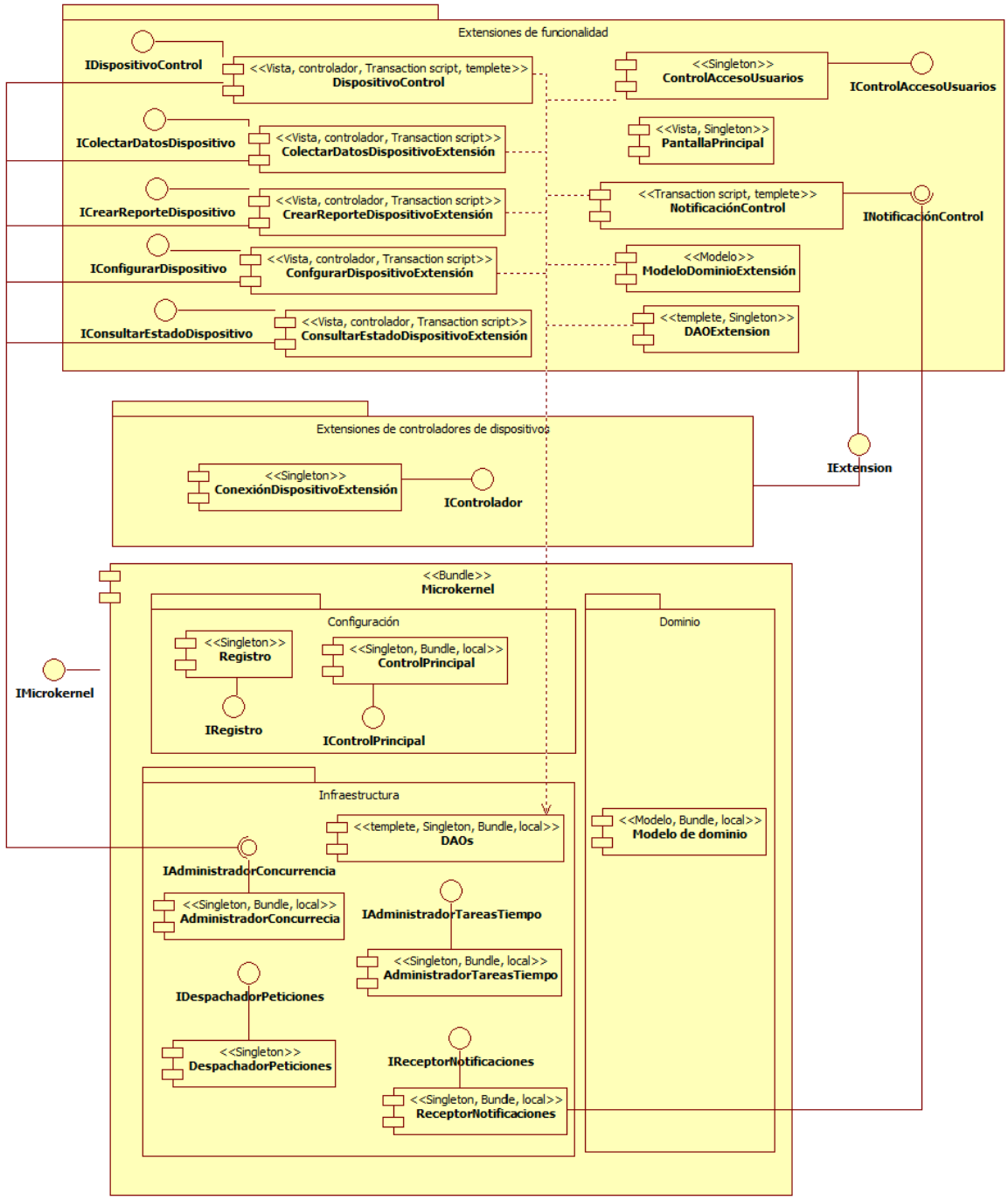


Figura B.4: Relaciones de uso entre las extensiones de funcionalidad, administrador de peticiones, DAOs y Receptor de notificaciones

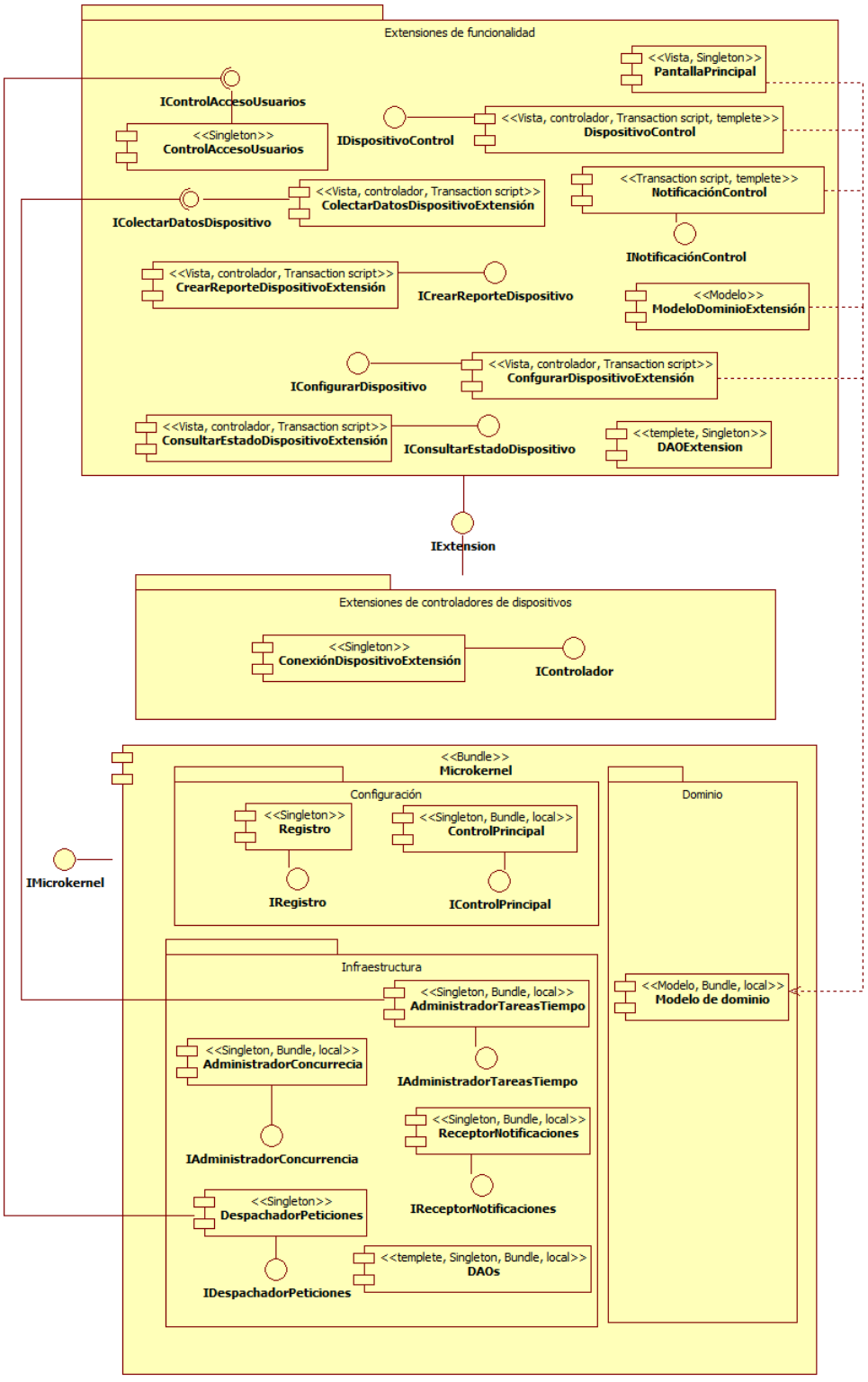


Figura B.5: Relaciones de uso entre las extensiones de funcionalidad, administrador de tareas de tiempo y modelo de dominio.

Vistas de asignación

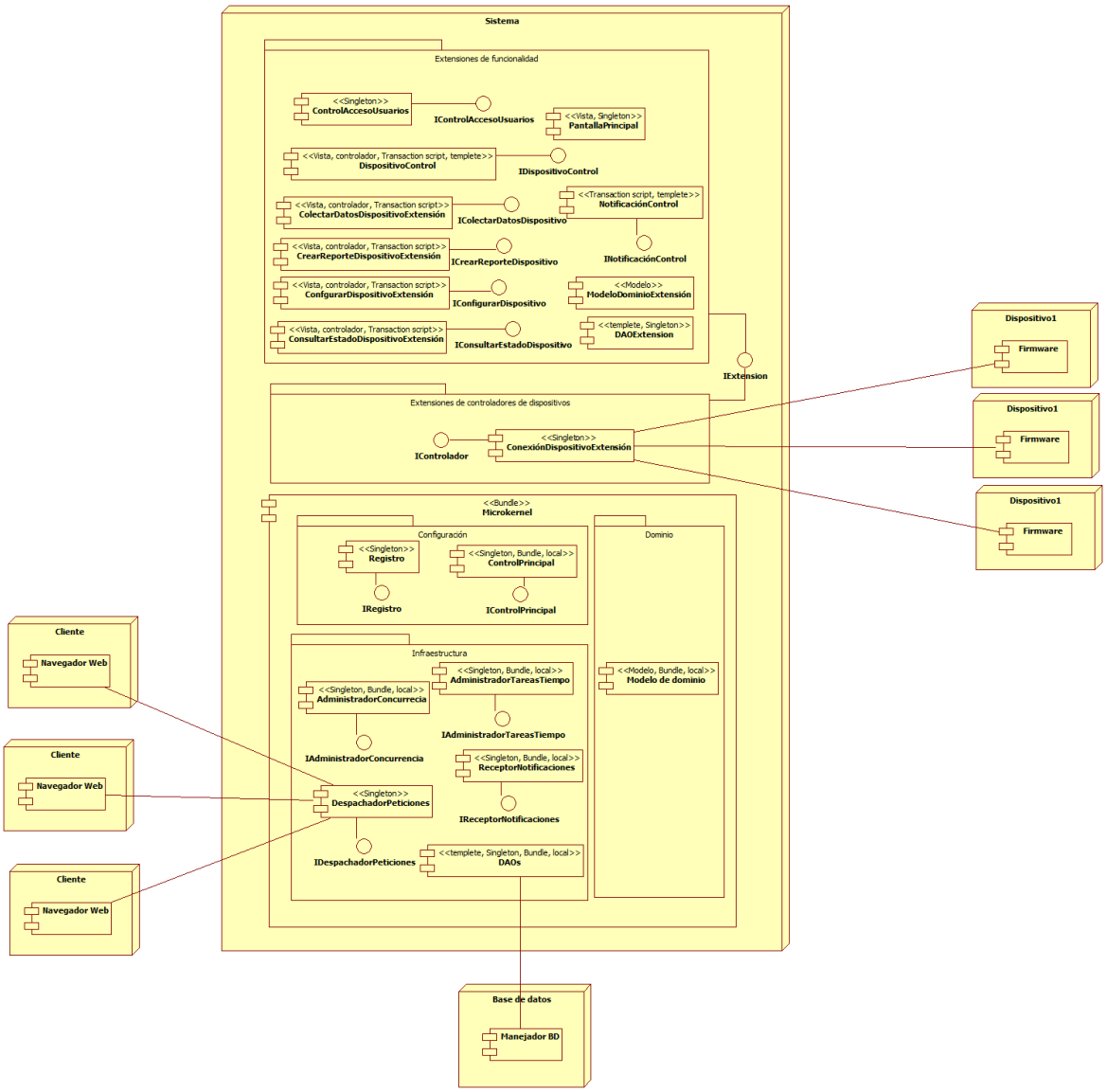


Figura B.6: Implantación en el entorno de producción de los elementos.

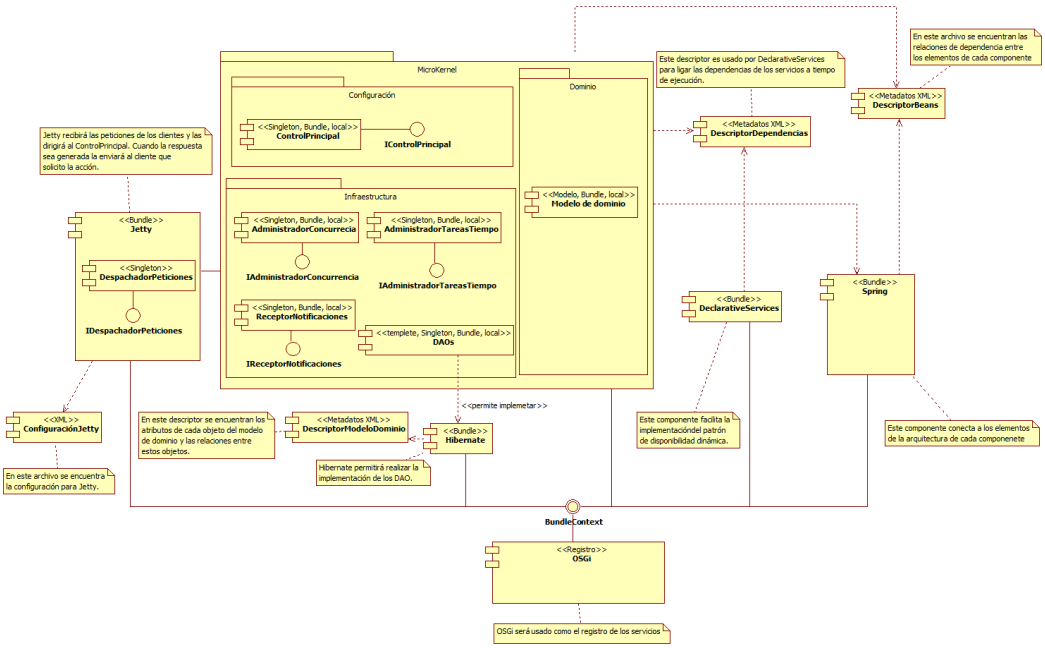


Figura B.7: Vista de implementación (primera parte).

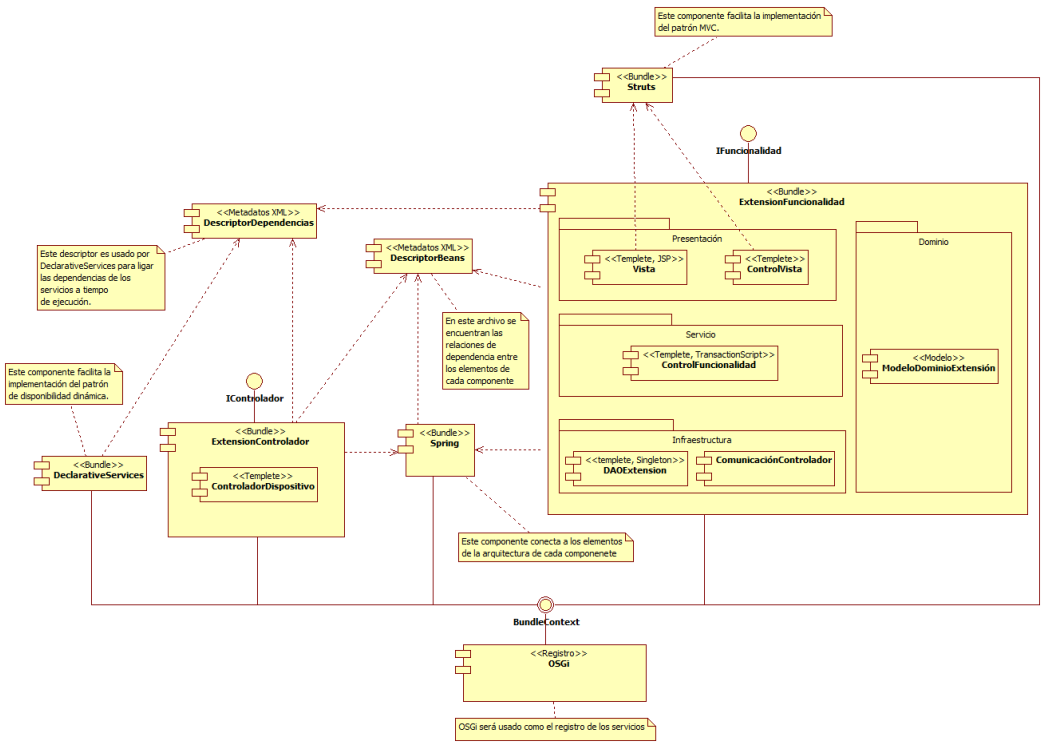


Figura B.8: Vista de implementación (segunda parte).

Vistas de componentes y conectores

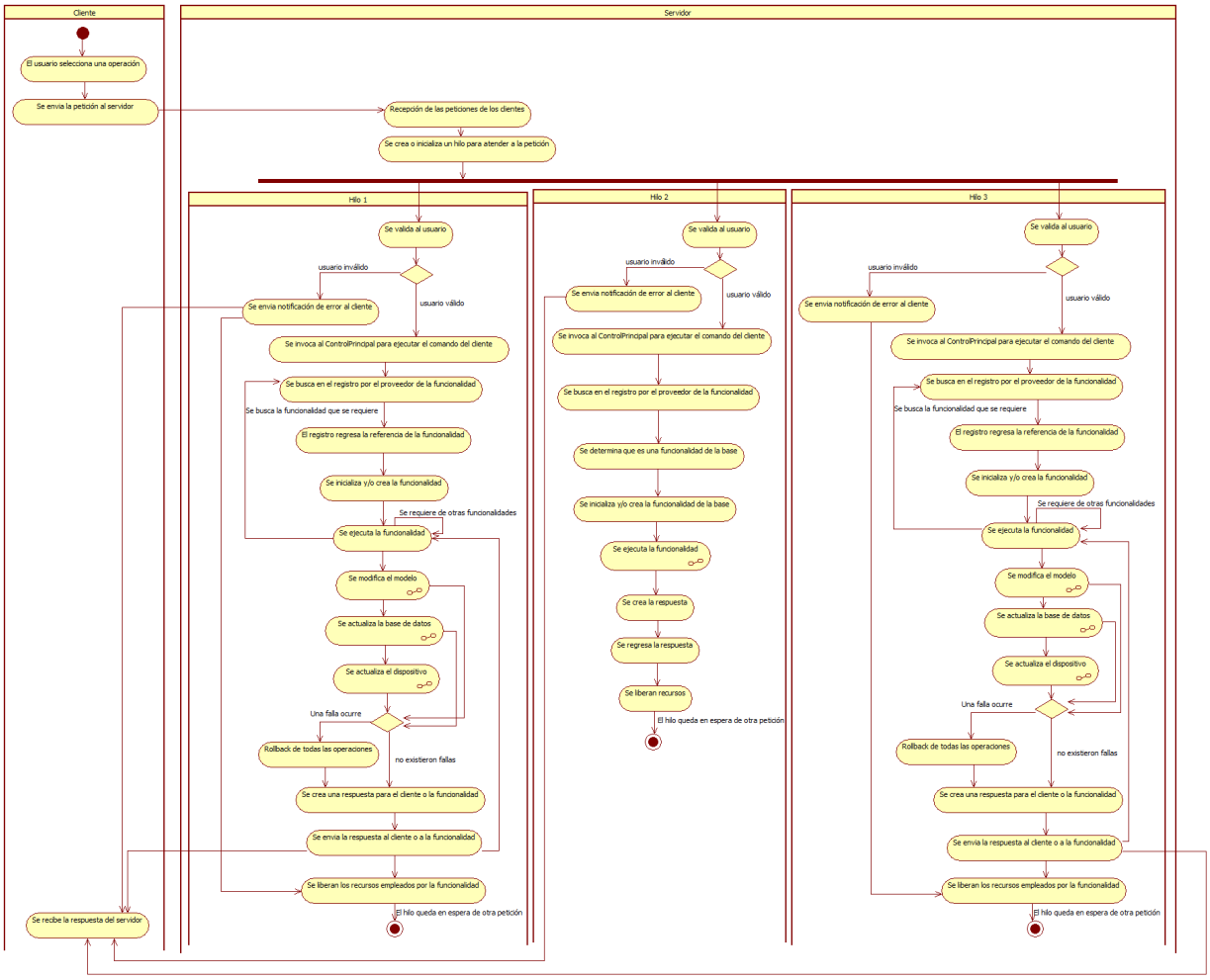


Figura B.9: Diagrama de actividades con hilos de atención de peticiones.

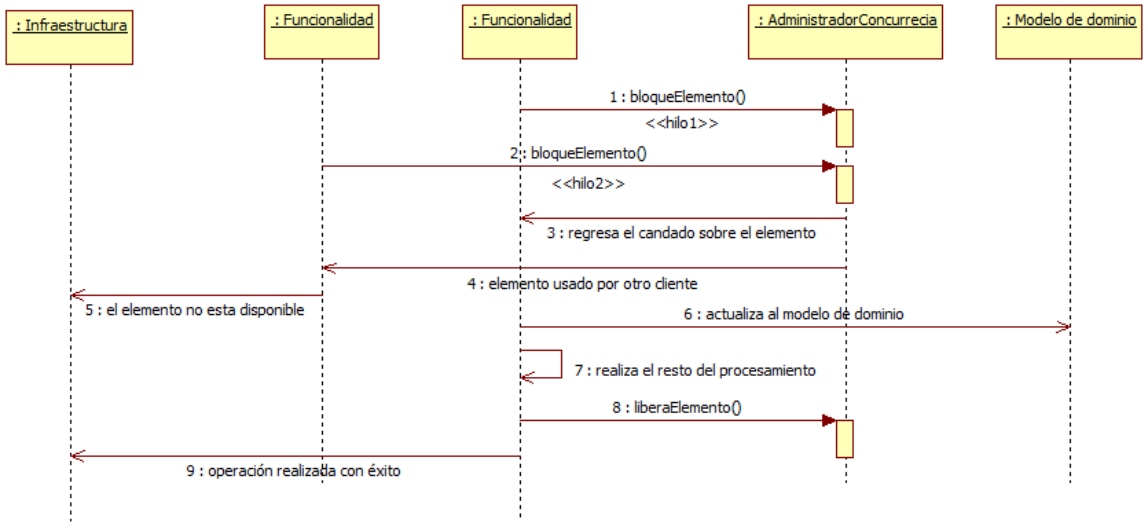


Figura B.10: Accesos concurrentes al modelo de dominio

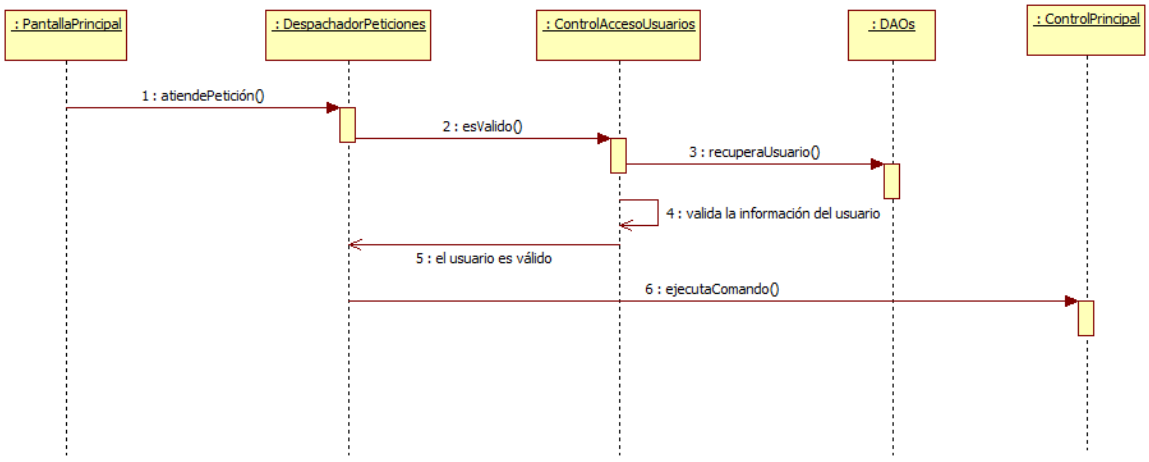


Figura B.11: Elementos que participan en la autentificación de usuarios

Atención de peticiones desde la pantalla principal

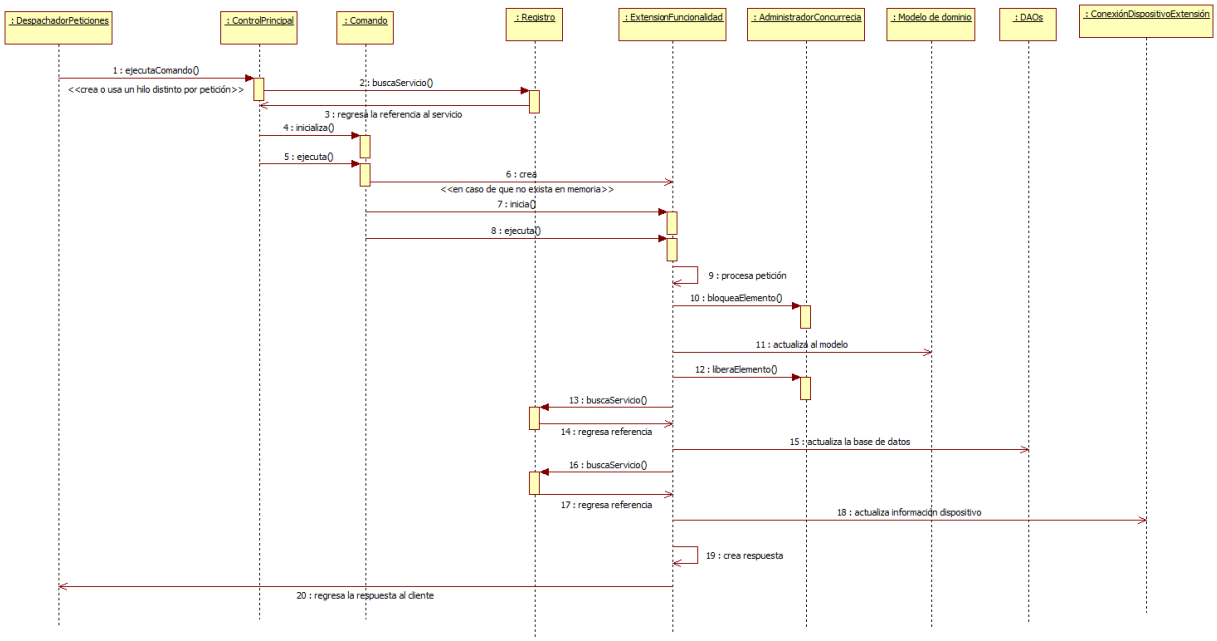


Figura B.12: Diagrama de secuencia para atender una petición (sin fallas)

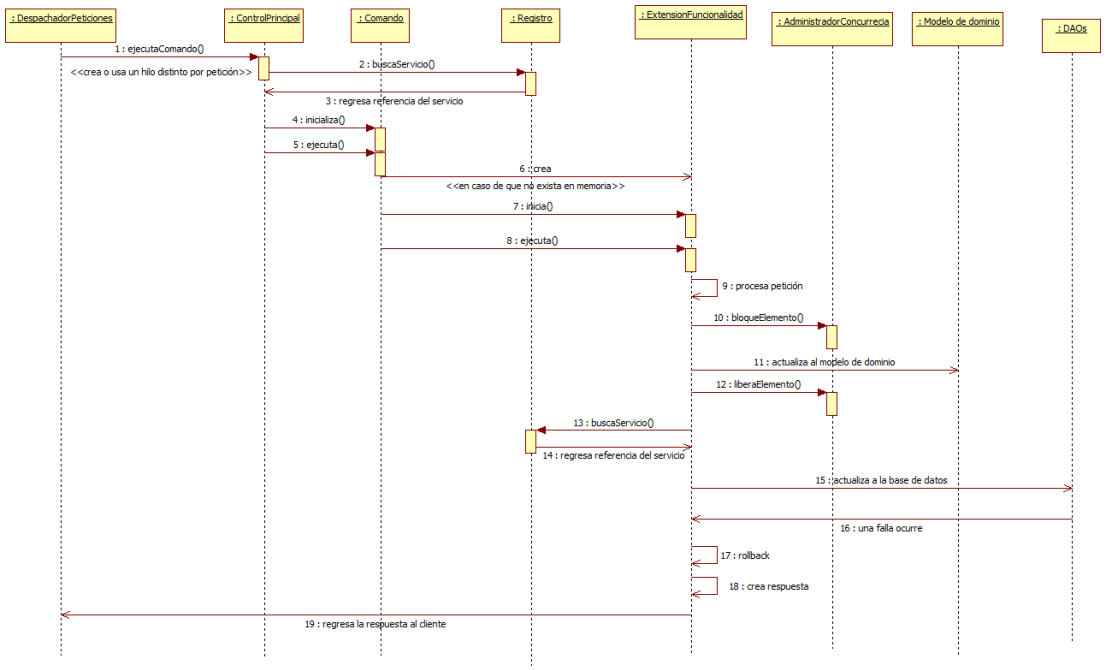


Figura B.13: Diagrama de secuencia para atender una petición (con fallas)

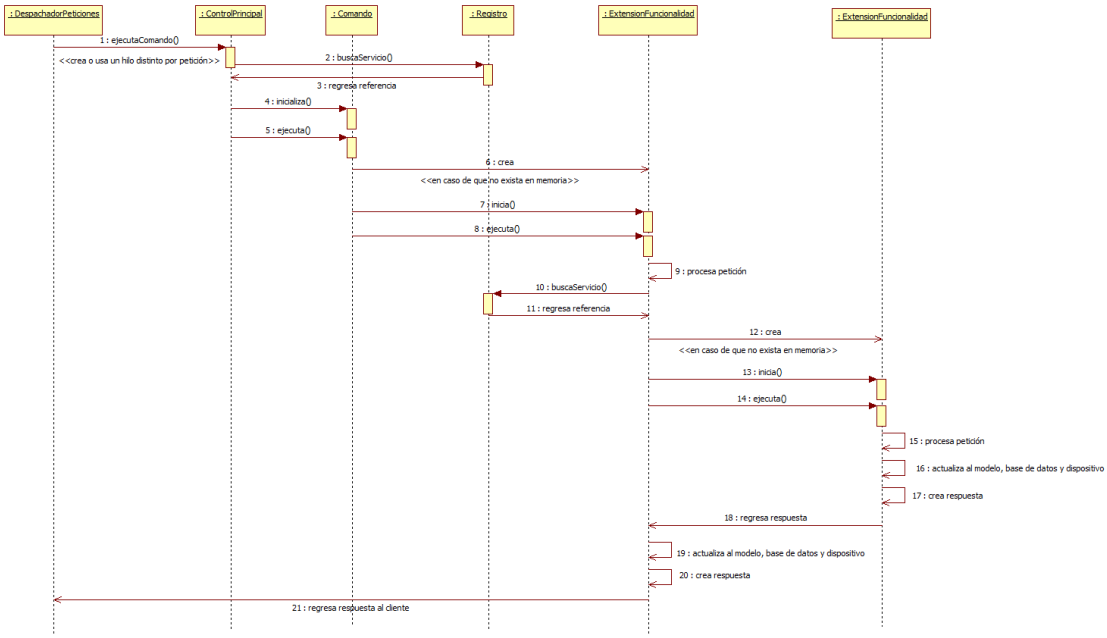


Figura B.14: Diagrama de secuencia con composición de servicios

Atención de peticiones desde la pantalla de una extensión

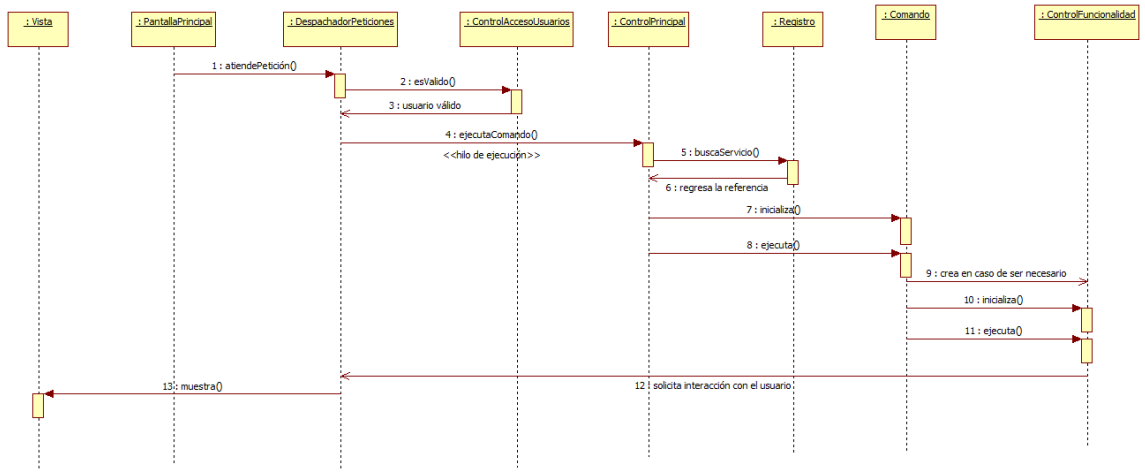


Figura B.15: Se lanza la pantalla de una funcionalidad

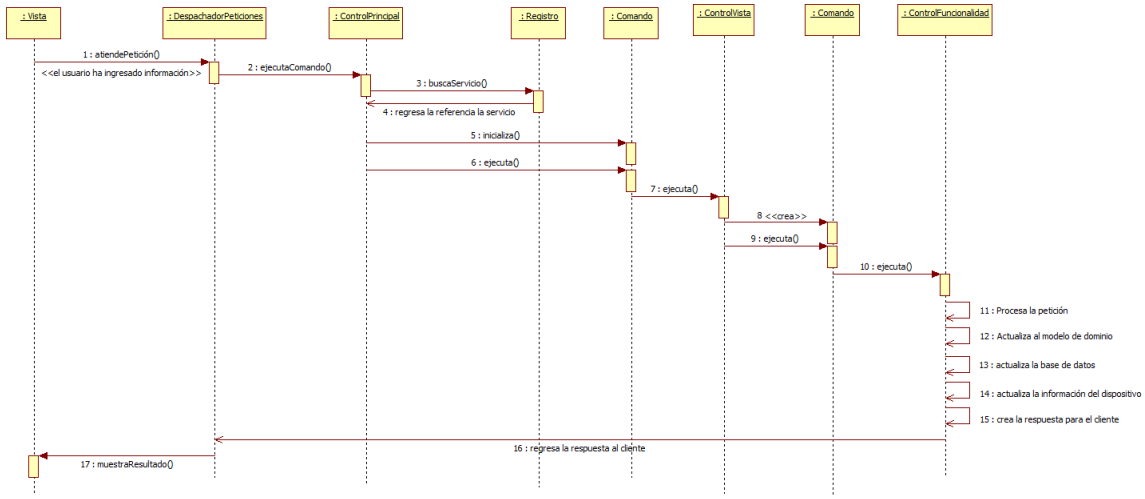


Figura B.16: Despacho de una petición lanzada desde la ventana de una funcionalidad

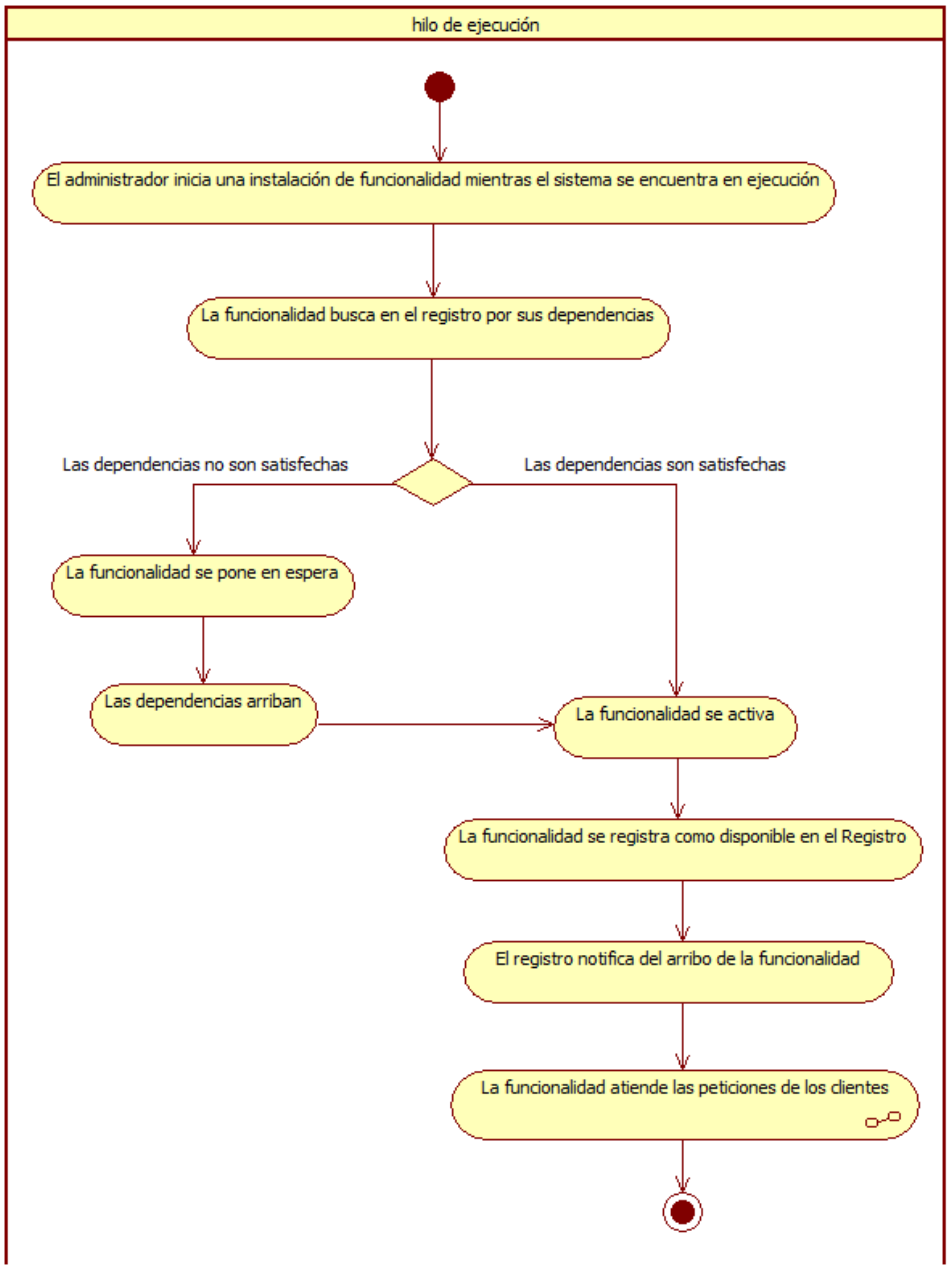


Figura B.17: Diagrama de actividades de la instalación de una nueva funcionalidad

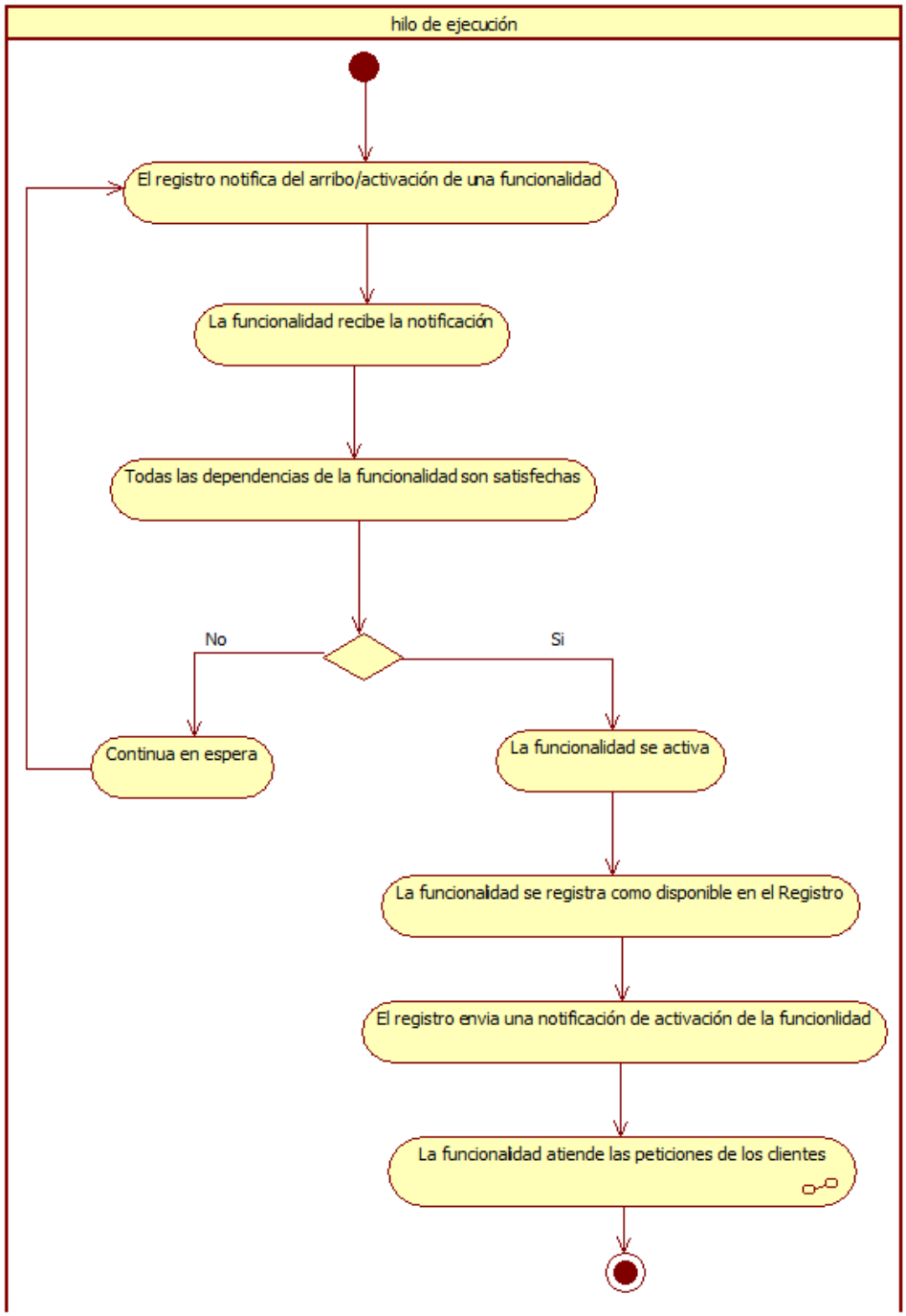


Figura B.18: Diagrama de actividades de arribo de dependencia.

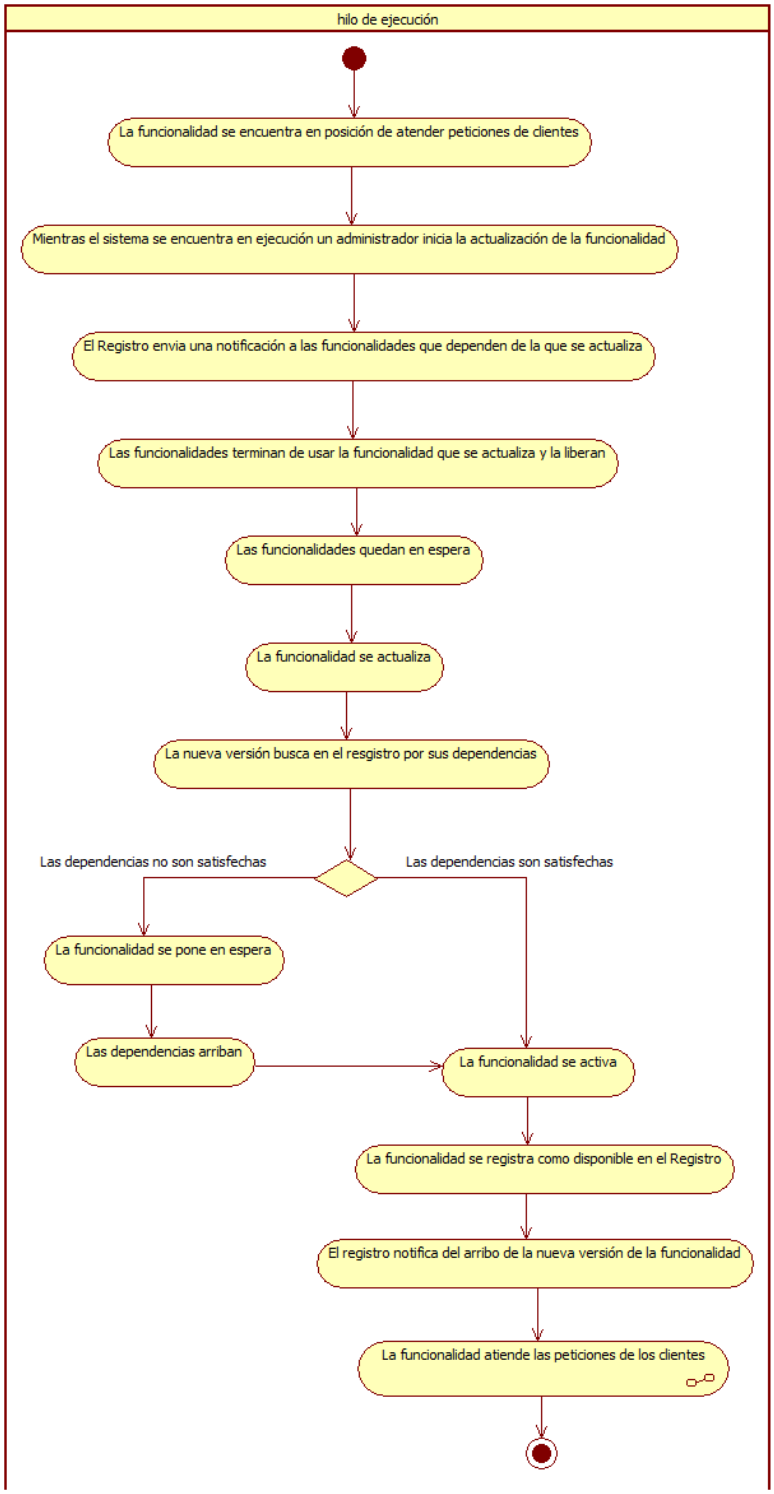


Figura B.19: Diagrama de actividades de la actualización de una funcionalidad.

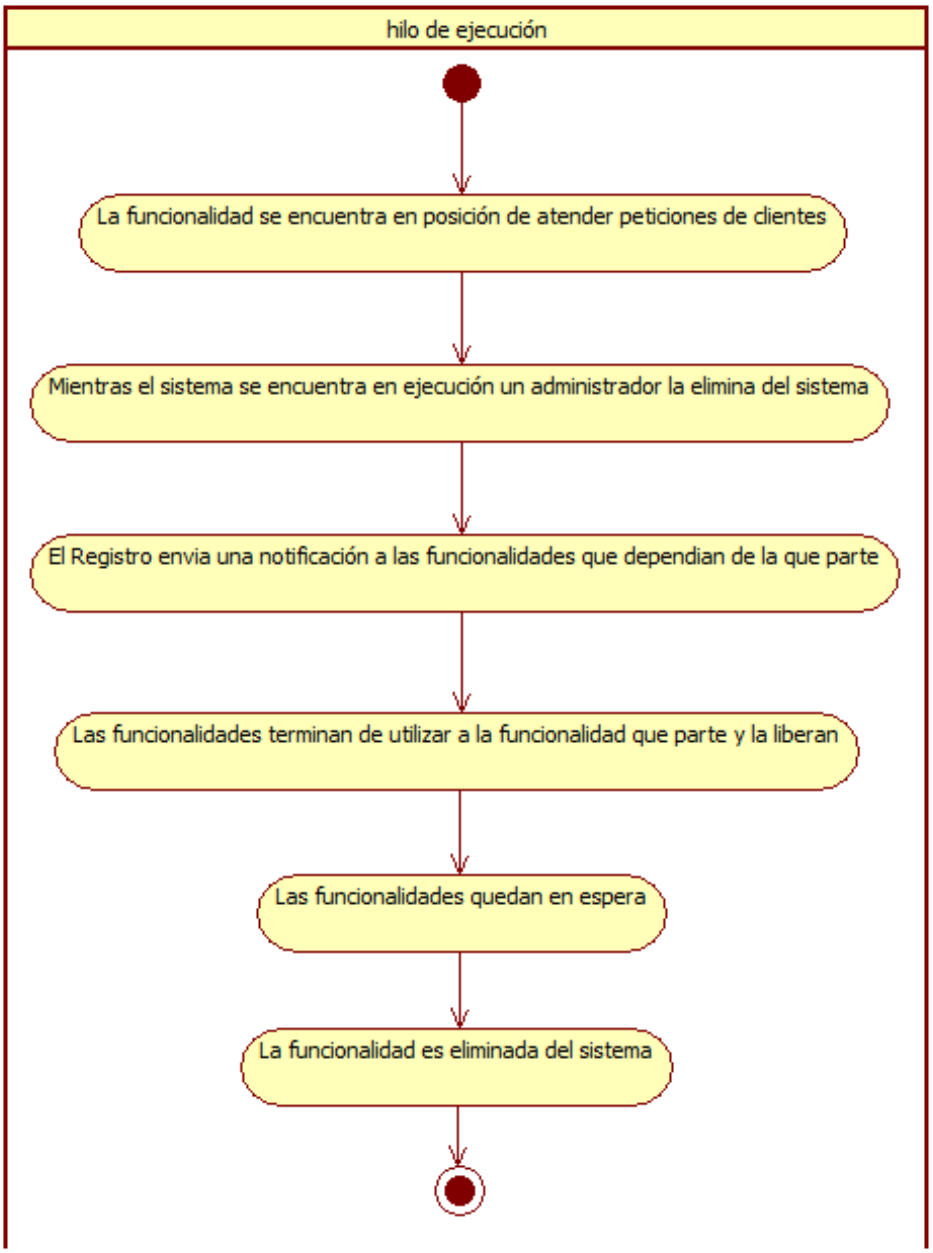


Figura B.20: Diagrama de actividades de la partida de una funcionalidad.

Descripción de los elementos:

1. Una configuración específica para una organización combinará elementos en las siguientes capas:

   a) Capa de Extensiones de funcionalidad: Contienen la implementación de la lógica de una funcionalidad para la aplicación base. Las extensiones pueden incluir lógica de presentación, negocio, datos e infraestructura.

   b) Capa de Extensiones de controladores de dispositivo: Contienen la lógica necesaria para poder establecer la comunicación con un dispositivo específico.

2. Capa de Microkernel: Está capa a su vez se descompone en las siguientes capas y elementos:

   a) El modelo de dominio contendrá a las siguientes abstracciones:

      1) Región: Abstracción de la forma de representación jerárquica del modelo. Solo existe una región raíz y todas las regiones hijo deben tener asignado a un solo padre.

      2) Nodo: Abstracción de un dispositivo genérico. Cada nodo estará asociado a una región

      3) Notificación: Abstracción de una notificación genérica generada por un nodo genérico. Las notificaciones deben estar asociadas a un nodo genérico.

      4) Usuario: Abstracción de los diferentes tipos de usuario que el sistema soporta. Cada usuario tiene asociado un conjunto de regiones a las cuales tiene permiso de ingresar.

      5) Las abstracciones Nodo y Notificación se deben extender por cada extensión de funcionalidad para incorporar las características específicas de cada dispositivo.

b) Capa de Infraestructura: Soporta la portabilidad de los niveles superiores (por ejemplo: cambio de manejador de base de datos, cambio de middleware, etc.) y soporta la administración de los recursos de la aplicación base.

c) Capa de Configuración: Soporta los mecanismos para configurar los servicios de la aplicación base así como el encaminamiento de las peticiones de los servicios externos.

1) Registro: Permite a los servicios internos y externos saber cuáles proveedores de servicio están disponibles (implementa el patrón Lookup)

3. DispositivoControl: Este componente es una plantilla de las tareas administrativas sobre los dispositivos (altas, bajas, consultas y actualizaciones de dispositivos)

4. ColectarDatosDispositivoExtensión: Realiza la función de recolectar los datos de un dispositivo en particular

5. CrearReporteDispositivoExtensión: Realiza la función de creación del reporte de un dispositivo particular

6. ConfigurarDispositivoExtensión: Realiza la función de configuración de los atributos de un dispositivo particular, esta función puede ser muy compleja debido a la cantidad de atributos de configuración de cada dispositivo

7. ColsultarEstadoDispositivoExtensión: Realiza la función de consulta de estado de un dispositivo particular

8. NotificaciónControl: Este componente es una plantilla de las tareas administrativas sobre las notificaciones que emite un dispositivo particular

9. ModeloDominioExtensión: Las extensiones requerirán de extender el modelo de dominio para incorporar las particularidades de cada dispositivo

10. DAOExtensión: Este componente es una plantilla que permite realizar la persistencia de los objetos de dominio particulares de cada dispositivo

11. Los componentes de extensión (componentes con terminación Extensión por estándar) pueden necesitar la implementación de lógica de presentación, por lo cual cada componente puede estar dividido en 2 capas, la primera capa contiene lógica de presentación y la segunda capa contiene lógica de negocio.

12. ConexiónDispositivoExtensión: Permite la interacción con cada dispositivo en su protocolo de comunicación

13. AdministradorConcurrencia: Este componente debe asegurar los accesos concurrentes a los objetos del modelo de dominio para evitar que el modelo quede en un estado inconsistente.

14. DespachadorPeticiones: Escucha las peticiones de los clientes y las dirige al ControlPrincipal para que decida que extensión la ejecuta. Internamente este componente hace uso de un pool de hilos de escucha de peticiones.

15. AdministradorTareasTiempo: Este componente es responsable de configurar y ejecutar a las extensiones y funcionalidades de la aplicación base que dependan del tiempo

16. ReceptorNotificaciones: Este componente se encarga de recibir todas las notificaciones y de solicitar al ControlPrincipal una extensión que administre a la notificación.

17. DAO: Este componente es una plantilla que define la especificación y algunas implementaciones del mecanismo de persistencia de los objetos del modelo de dominio

18. ControlAccesoUsuarios: Contiene la lógica que permite decidir si un usuarios es válido o no, con lo cual se permite o no su ingreso al sistema, solo existirá 1 elemento de este tipo en toda la aplicación.

19. PantallaPrincipal: Contiene a todos los elementos gráficos que son compartidos por todas las extensiones y funcionalidades de la aplicación, solo existirá 1 elemento de este tipo en toda la aplicación.

20. MVC:

a) El modelo se encuentra dividido en 2 lugares, la parte genérica se encuentra en la aplicación base y las partes específicas a cada dispositivo se encuentran en las extensiones

b) Los controladores se encuentran en las extensiones y permiten actualizar a las vistas

c) Las vistas muestran la información de cada tipo particular de dispositivo o bien colectan la información necesaria para configurar un dispositivo particular.

Descripción de las interfaces de los elementos:

Nombre de interfaz	**Componente al que corresponde**	**Métodos principales**
IRegistro	Registro	- Registra(Servicio s): Permite registrar a un nuevo servicio que se integra a la aplicación base - Desregistra(Servicio s): Permite eliminar a un servicio del registro para que los potenciales clientes del mismo ya no lo puedan usar - Busca(Descripción d): Permite buscar en el registro por un tipo de servicio dado
IControlPrincipal	ControlPrincipal	- Ejecuta(Comando c): Permite ejecutar un comando enviado desde el cliente, el comando debe incluir que tipo de servicio puede atender la solicitud

IExternsión	Todas las extensiones	- GetExtensión(int versión): Este método regresa una versión específica de la interfaz elegida, en caso de que la versión no exista se lanzará una excepción.
IFuncionalidad	Todas las extensiones de funcionalidad	- Inicia(): Inicializa a la extensión - Ejecuta(Comando c): Realiza el procesamiento de la solicitud y regresa una respuesta. La funcionalidad debe liberar todos los recursos que ya no utilice con el fin de que otros elementos puedan usarlos.

IControlador	Todas las extensiones de control de dispositivo	- Inicia(): Inicializa al componente - PruebaConexión(): Prueba si el dispositivo está disponible o no. Este método es útil cuando se requiere saber si el dispositivo está disponible para realizar operaciones sobre él. - Ejecuta(Comando c): Realiza la ejecución de la petición de la extensión sobre el dispositivo. El controlador debe liberar todos los recursos que ya no utilice con el fin de que otros elementos puedan usarlos.
IDispositivoControl, IColectarDatosDispositivo, ICrearReporteDispositivo, IConfigurarDispositivo, IConsultarEstadoDispositivo, INotificaciónControl	DispositivoControl, ColectarDatosDispositivoExtension, CrearReporteDispositivoExtension, ConfigurarDispositivoExtension, ConsultarEstadoDispositivoExtension, NotificaciónControl	Extienden a la interfaz IFuncionalidad y definen la forma en que se inicializará la extensión
IConexiónDispositivo	ConexiónDispositivoExtensión	Extiende a la interfaz IControlador y define la forma en que inicializará la extensión

IAdministradorConcurrencia	AdministradorConcurrencia	- BloqueaElemento(Elemento e): Obtiene el candado para manejar de forma exclusiva al elemento - LiberaElemento(Elemento e): Libera el candado sobre el elemento y permite que otro usuario pueda hacer uso de él
IDespachadorPeticiones	DespachadorPeticiones	atiendePetición(Petición p): Realiza la escucha de peticiones
IAdministradorPetciones	AdministradoPeticiones	- Configura(): Configura a todas las tareas de tiempo que no han sido configuradas - Inicia(): Arranca a todas las tareas de tiempo que no han sido iniciadas - Busca(): Busca en el registro a las tareas que dependen del tiempo
IReceptorNotificaciones	ReceptorNotificaciones	recibeNotificación(String notificación): Recibe a las notificaciones generadas por los dispositivos
IControlAccesoUsuarios	ControlAccesoUsuarios	EsValido(Usuario u): Permite el acceso o no de un usuario al sistema

B.3. Documentación de algunos resultados de AEM

Motivaciones arquitectónicas	Administrar dispositivos
Preguntas	Cuando se actualiza una funcionalidad, ¿Qué pasa con su estado?
Respuestas	Las funcionalidades son stateless, pero se debe asegurar que la funcionalidad que será actualizada no esté en ejecución cuando se vaya a actualizar para que no se afecte el estado compartido por todas las funcionalidades
Referencias a elementos	NR-2
Decisiones arquitectónicas	
Argumentos	
Diseño:	

Motivaciones arquitectónicas	Administración de la representación gráfica
Preguntas	¿Cómo se obtienen las pantallas específicas a un tipo de nodo particular?
Respuestas	Ver el diseño propuesto
Referencias a elementos	C-1
Decisiones arquitectónicas	Cada extensión de un dispositivo particular presenta su propia interfaz gráfica cuando esta es solicitada por un cliente
Argumentos	El hecho de usar un cliente ligero permite que se re-dirijan las peticiones de interfaz gráfica de forma sencilla, con solo agregar elementos a la petición HTTP
Diseño: ver Figura B.15	

Motivaciones arquitectónicas	Administrar dispositivos
Preguntas	¿Qué sucede si se bloquea la ejecución por falta de respuesta de un dispositivo?
Respuestas	Las solicitudes de información hacia los dispositivos tendrán un timeout
Referencias a elementos	PS-1
Decisiones arquitectónicas	En los componentes de comunicación con dispositivos se colocaran timeouts configurables mediante una archivo de configuración
Argumentos	Los timeouts y el número de reintentos de comunicación permitirán que el sistema no se bloque a causa de un que dispositivo falle. El tiempo de respuesta puede ser igual a: Tiempo de solicitud + Retardo de la red en la solicitud + tiempo de procesamiento de la solicitud + (Número de intentos de comunicación * timeout) + Tiempo de generación de respuesta + Retardo de la red en envío de respuesta
Diseño:	

Motivaciones arquitectónicas	Administración de la representación gráfica
Preguntas	¿Dónde se realiza el cálculo del estado de cada región?
Respuestas	El cálculo del estado se realiza en el servidor cuando una notificación que cambie el estado de algún nodo llegue al sistema
Referencias a elementos	R-1, R-4
Decisiones arquitectónicas	Se debe tener 1 hilo que atienda las actualizaciones de estado de los elementos en el modelo de dominio, el Administrador de concurrencia debe permitirle de forma prioritaria a este hilo el acceso al modelo de dominio
Argumentos	Con un hilo de actualización se pueden realizar tareas de actualización del modelo de dominio, pero se tienen problemas cuando los dispositivos generan gran cantidad de notificaciones debido a que estas tendrán prioridad sobre las peticiones de los clientes. Esto podría implicar que las peticiones de los clientes sean rechazadas constantemente.
Diseño:	

ID	Descripción	Responsable
NR-1	Los filtros en el registro permiten administrar de forma adecuada la generación de notificaciones a los componentes.	
NR-2	El estado de las funcionalidades es stateless con lo que su actualización se facilita.	
PS-1	El timeout y los reintentos sobre las peticiones hacia los dispositivos afectan al tiempo de respuesta hacia los clientes.	Ismael Nuñez
R-1	El servidor podría estar siempre ocupado en el caso de que los dispositivos generen gran cantidad de notificaciones que cambien el estado de algún nodo.	Ismael Nuñez
C-1	Mostrar las funcionalidades específicas puede impactar en la usabilidad del sistema y en el desempeño del servidor.	Ismael Nuñez
R-2	Las notificaciones son enviadas a los clientes hasta que los clientes las solicitan o realizan una operación que no se pueda realizar, esto podría dejar al modelo del cliente inconsistente.	Ismael Nuñez
NR-3	El componente ConfiguraDispositivoExtensión puede estar compuesto de elementos que permitan realizar configuraciones específicas y el servidor las despacha como si fueran genéricas.	
R-3	Cuando se realizan muchas peticiones de creación de reportes el desempeño del sistema podría verse afectado.	Ismael Nuñez
R-4	Si los dispositivos generan muchas notificaciones los clientes podrían nunca tener acceso para modificar al modelo de dominio.	Ismael Nuñez
PS-2	El número de hilos de escucha, el tamaño de la cola de espera y el número de hilos de despacho de peticiones afectan a la disponibilidad.	Ismael Nuñez
NR-4	Si fuera creado un componente de atención de notificaciones por cada notificación creada por los dispositivos la memoria del sistema podría ser consumida en su totalidad.	

C-2	El tiempo que se almacenan los datos de los dispositivos en el sistema afecta al desempeño y a la escalabilidad.	Ismael Nuñez
PS-3	El intervalo de tiempo en que los datos de los dispositivos son colectados afecta al desempeño del sistema.	Ismael Nuñez
PS-4	El número de hilos que colecten datos de los dispositivos afecta al desempeño del sistema.	Ismael Nuñez
NR-5	Las respuestas de los clientes que fallan después de enviar su petición se desechan, pero se conservan los cambios en el modelo de dominio	
NR-6	Se crean tablas genéricas y particulares para las notificaciones y nodos.	
NR-7	La política de rechazo permite no emplear otro componente con otra cola de espera.	
C-2	Incrementar el tamaño de alguno de los pools afecta al desempeño (se pueden recibir más notificaciones) y a la disponibilidad (el sistema puede caer ante un eventual consumo de memoria excesivo).	Ismael Nuñez
NR-8	La adaptación activa de funcionalidades permite que las funcionalidades inactivas se vuelvan activas en el momento en que sus dependencias se vuelven activas para de esta forma proveer el servicio que realizan.	
NR-9	El registro sabe que funcionalidad o dispositivo va a ser eliminado o actualizado y deja de enviarle trabajo para que pueda serlo.	
C-3	El uso de políticas automáticas de recolección de basura y liberación de servicios facilita el trabajo al desarrollador, pero decrementan el desempeño.	Ismael Nuñez
R-5	£Cómo evitar que las extensiones se ciclen?	Ismael Nuñez
NR-10	A los elementos arquitectónicos que serán eliminados ya no se les envían peticiones.	

Tabla B.6: Tabla de riesgos, no riesgos, compromisos y puntos sensibles.

Referencias

[Ali-Babar2004] Ali Babar Muhammad et al, "*A Framework for Classifying and Comparing Software Architecture Evaluation Methods,"* Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04), 2004.

[Barbacci2003] Barbacci Mario R. et al, "*Quality Attribute Workshops (QAW) Third Edition,"* Reporte Técnico del CMU-SEI, 2003.

[Barisic2007] Barisic Daniel et al, "*Making Embedded Software Development More Efficient with SOA,"* 21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07), IEEE, 2007.

[Bass2003] Bass Len et al, "*Software Architecture in Practice,"* Addison Wesley, ISBN 0-321-15495-9, 2003.

[Buschmann96] Buschmann Frank et al, "*Pattern-Oriented Software Architecture: A System of Patterns Volume 1,"* Wiley, ISBN 0 471 95889 7, 1996.

[Buschmann2007] Buschmann Frank et al, "*Pattern-Oriented Software Architecture: A Pattern Languaje for Distributed Computing Volume 4,"* Wiley, ISBN 978-0-470-05902-9, 2007.

[Cervantes2003] Cervantes Humberto y Hall Richard S., "*Automating Service Dependency Management in a Service-Oriented Component Model,"* Proceedings of the 6th International Workshop on Component-Based Software Engineering - (CBSE), 2003.

[Cervantes2004] Cervantes Humberto y Hall Richard S., "*Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model,"* International Conference on Software Engineering (ICSE), 2004.

[Clements2002] Clements Paul et al, "*Documenting Software Architectures: Views and Beyond,"* Addison Wesley, ISBN 0-201-70372-6, 2002.

[Dobrica2002] Dobrica Liliana y Niemela Eila, "*A Survey on Software Architecture Analysis Methods,"* IEEE Transaction on Software Engineering, Vol. 28 Num. 7, 2002.

[Fowler2004] Página web de Matin Fowler. Consultada el 29/03/2009, "*Inversion of Control,"* martinfowler.com/articles/injection.html.

[Gamma94] Gamma Erick et al, "*Design Patterns: Elements of Reusable Object-Oriented Software,"* Addison-Wesley Professional Computing Series, ISBN 0-201-63361-2, 1994.

[Gamma96] Gamma Erich, "*The Extension Objects Pattern,"* Reporte técnico, Washington University, wucs-97-07. 1996. (PLoP '96).

[Henney2002] Henney K., "*C++ Patterns: Reference Accounting,"* Proceedings of the Sixth European Conference on Pattern Languages of Programming, EuroPLoP 2001, 2002.

[Hofmeister2005] Hofmeister Christine et al, "*Generalizing a Model of Software Architecture Design from Five Industrial Approaches,"* Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA5), 2005.

[Humphrey99] Humphrey Watts S., "*Introduction to the Team Software Process,"* Addison-Wesley Professional, ISBN-10: 020147719X, 1999.

[Jammes2005] Jammes F. et al, "*Service-Oriented Architecture for Devices - the SIRENA View,"* In Proc. 3rd IEEE Int. Conf. on Industrial Informatics (INDIN), 2005.

[Kazman2000] Kazman Rick et al, "*ATAM: Method for Architecture Evaluation,"* Reporte Técnico del CMU-SEI, 2000.

[Kazman2004] Kazman Rick et al, "*Integrating Software Architecture-Centric Methods into the Rational Unified Process,"* Reporte técnico del CMU-SEI, 2004.

[Kruchten95] Kruchten P., "*The 4+1 View Model of Architecture,"* IEEE Software, vol. 12, no. 6, 1995.

[Land2002] Land Rikard, "*A Brief Survey of Software Architecture,"* Reporte Técnico Mälardalen University, 2002.

[Lattanze2005] Lattanze Anthony J., "*The Architecture Centric Development Method,"* Reporte Técnico de la Escuela de Ciencias de la Computación Carnegie Mellon University, 2005.

[Nord2004] Nord Robert L. et al, "*Integrating the Quality Attribute Workshop (QAW) and the Attribute-Driven Design (ADD) Method,"* Reporte técnico del CMU-SEI, 2004.

[OBrien2005] O'Brien Liam et al, "*Quality Attributes and Service-Oriented Architectures,"* Reporte Técnico CMU-SEI, 2005.

[wwwCOM] Página web de Microsoft sobre COM. Consultada el 27/10/2008. http://www.microsoft.com/com/default.mspx

[wwwSEI] Página web de arquitectura de software del SEI. Consultada el 03/07/2008. http://www.sei.cmu.edu/architecture/

[W3C2004] Worldwide Web Consortium (W3C). Consultada el 06/12/2008. "Web Services Glossary,"http://www.w3.org/TR/ws-gloss/.

[Szyperski98] Szyperski, C., "*Component software: beyond object-oriented programming,*" ACM Press/Addison-Wesley Publishing Co., 1998.

[Wojcik2006] Wojcik Rob et al, "*Attribute-Driven Design (ADD), Version 2.0,*" Reporte técnico del CMU-SEI, 2006.

UNIVERSIDAD AUTÓNOMA METROPOLITANA

Casa abierta al tiempo

Adaptación de una Metodología de Desarrollo Arquitectónico al Contexto de Equipos de Desarrollo Pequeños

Para obtener el grado de
MAESTRO EN CIENCIAS
(Ciencias y Tecnologías de la Información)

P R E S E N T A:
Lic. José Ismael Nuñez Reyna

Asesores

Dr. Humberto Cervantes Maceda

Sinodales

Presidente: Dr. Cuauhtémoc Lemus Olalde
Secretario: M. en C. Eduardo Rodríguez Flores
Vocal: Dr. Humberto Cervantes Maceda

Humberto Cervantes Maceda

16 de octubre de 2009