Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

Itinerario: Ingeniería de Software (código asignatura 31105128)
Título: “ArchE: Arquitecturas y Toma de decisiones”
Estudiante: Iván Builes Victoria
Director: José Félix Estívariz López
Curso académico 2012/13 (defensa convocatoria de septiembre)

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

Itinerario: Ingeniería de Software (código asignatura 31105128)
Título: “ArchE: Arquitecturas y Toma de decisiones”
Trabajo tipo A
Estudiante: Iván Builes Victoria
Director: José Félix Estívariz López

Impreso TFdM05_Autor. Autorización de publicación

y difusión del TFdM para fines académicos

Autorización

Autorizo/amos a la Universidad Nacional de Educación a Distancia a difundir y utilizar, con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del/los ~~Autor/~~es

Juan del Rosal, 16
28040, Madrid

Tel: 91 398 89 10
Fax: 91 398 89 09

www.issi.uned.es

Resumen:

La construcción de software prestando atención sólo a los requisitos funcionales puede acabar en fracaso por no atender a la importancia de los atributos de calidad del software. Estos atributos de calidad determinan en buena medida la arquitectura de un sistema software. Se hace por tanto necesario aplicar métodos que permitan asegurarnos la calidad del software, como Attribute-Driven Design y emplear también herramientas que nos asistan en esta tarea, como Architecture Expert (ArchE). El valor de estas herramientas se revela en ejemplos concretos de sistemas diseñados desde cero y también en sistemas que emplean componentes comerciales off-the-shelf.

Palabras clave:

ArchE, Arquitectura Software, Atributos de Calidad, Attribute Driven Design, Calidad, COTS, Diseño, Patrones Arquitectónicos, Tácticas.

Contenido

Listado de Imágenes

Listado de Tablas

Introducción y estructura del trabajo

El presente documento es el Trabajo fin de Máster denominado "ArchE: Arquitecturas y Toma de Decisiones". Como todo Trabajo fin de Máster, tiene ciertos contenidos mínimos y estructura que vienen determinados por la oferta publicada para dicho trabajo. Moviéndome dentro de dicha estructura y contenidos, el presente documento tiene una **primera parte en la que se realiza una introducción a la Arquitectura de Sistemas Software** a partir de la cual podemos ver la **importancia de los Atributos de Calidad**. Este tema sigue con un repaso sobre los **escenarios generales y concretos para atributos de calidad**.

En la **segunda parte** me centro en la **obtención de la calidad en la construcción de un sistema software**, para lo cual se revisan los conceptos de **tácticas de calidad** y **patrones arquitectónicos**. Se introduce al lector también en **métodos de obtención de arquitecturas que nos aseguren la calidad, como Attribute-Driven Design**.

En la **tercera parte** se introduce la **herramienta de asistencia al diseño arquitectónico ArchE**, revisando su **instalación** y las **ventajas de virtualizarla en lo que he denominado un ArchE appliance**. Basándose en una detallada **descripción de las tácticas de modificabilidad** se estudia un **ejemplo sencillo de la herramienta en un escenario de modificabilidad con el ejemplo de aplicación que se ha venido empleando hasta ahora**.

En una breve **cuarta parte**, se comentan dos **retos detectados para ArchE** que consiste en las situaciones de **aplicación de varias tácticas que terminen en aparentes problemas recursivos** y la **problemática de integración de sistemas COTS en los diseños arquitectónicos**.

En la una **quinta parte** del trabajo, **expondré el análisis completo del sistema cuyas partes se han venido utilizando como ejemplo a lo largo del texto, un Sistema de Gestión de Telefonía IP**. Este análisis **servirá de punto de partida para someterlo a un análisis con ArchE**.

Finalizaré el trabajo con unas **conclusiones** y lo que podrían ser unas **líneas de trabajos futuros**.

Iván Builes Victoria

1. Arquitectura Software

1.1 ¿Qué es la Arquitectura Software? ¿Por qué se considera importante?

En palabras de los autores del texto base empleado para este trabajo [1], podemos definirla de la siguiente manera:

> *La arquitectura software de un programa o sistema de computación es la estructura o estructuras del sistema, que comprenden los elementos software, las propiedades visibles externamente de dichos elementos y las relaciones entre ellos.*

Dentro de la definición se habla de visibilidad externa de propiedades, que se debe entender como las suposiciones o presunciones que otros elementos pueden hacer sobre uno dado, especialmente acerca de los servicios que proporciona o su rendimiento o recursos que emplea, etc… Básicamente, los autores defienden que esta definición tiene cuatro grandes implicaciones:

1. De la definición se deduce que todo sistema computacional con software, tiene una arquitectura software, aunque no tenga ésta una representación gráfica, algo que entra dentro del dominio de la documentación del sistema. La representación gráfica no es la arquitectura.
2. Se entiende por estructura una representación de un conjunto de elementos, tal y como existen en el software o hardware representado, junto con sus relaciones[1]. Hablar de estructura o estructuras no es gratuito, sino pretende dejar claro que una única estructura no puede reclamar ser la "arquitectura", sino que puede ser un conjunto de estructuras las que nos den dicha arquitectura buscada[2]. Hay que tener en cuenta que dentro de las estructuras para describir un sistema algunas puede focalizarse en aspectos estáticos y otras en aspectos de interacción en tiempo de ejecución y es posible necesitar una o varias de estas estructuras.
3. El comportamiento de un elemento es parte de la arquitectura desde el momento que pueda ser observado desde el punto de vista de otro elemento. Lo que permite a los diferentes elementos interaccionar es claramente parte de la arquitectura.
4. La arquitectura define los elementos software, es una abstracción del sistema que omite los detalles que no afectan el cómo se usa un elemento software o cómo se relaciona o interacciona con los otros.

Si ésta es nuestra respuesta a qué es la Arquitectura Software, el por qué se considera importante hemos de encontrarlo en que proporciona un magnífico vehículo de comunicación entre todos los interesados, ya que aporta un nivel de abstracción adecuado y un lenguaje común a todos. Adicionalmente la arquitectura explicita todas las decisiones de diseño que han tomado los interesados y finalmente supone un modelo transferible, se convierte en una propiedad intelectual de valor para la empresa y que puede incluso demostrar valor en el futuro, conforme vaya evolucionando.

[1] Aunque la utilización de "estructura" por parte de los autores de [1] pueda ser algo oscura en la definición, se puede entender mucho mejor si pensamos que dichas estructuras son representables en cualquiera de los diagramas UML estándar. De hecho este es el uso que se le da sin ambigüedad alguna durante el libro. Un ejemplo de estructura estática estaría en un diagrama de módulos o en uno de despliegue, mientras que una dinámica podría ser un diagrama de actividad o uno de secuencia, por ejemplo.

[2] ¿Cuántas veces hemos visto una descripción comercial de un producto software en la que en un único diagrama se nos dice que ahí tenemos la arquitectura? Esta precisión de los autores es sencillamente genial y todo un aviso a navegantes. Ante este tipo de prácticas, el arquitecto deberá prestar más atención si cabe para encontrarla.

A los tres elementos de importancia citados anteriormente, podríamos sumarles además la siguiente lista:

- Una Arquitectura puede terminar dictando toda una estructura organizativa: sólo con pensar en las estructuras que empleemos para representarla, encontraremos que en ellas se pueden distinguir módulos o grupos de elementos en el sistema que pueden servir de base para la división del trabajo.
- Nos permite analizar y estimar mejor los costes asociados y también los tiempos de puesta en marcha de nuevos sistemas. Incluso puede servir de base para la formación de los especialistas.
- Una Arquitectura nos puede ayudar a estudiar la evolución del sistema. Podemos incluso obtener modelos ejecutables que nos ayuden a estudiar con agilidad el sistema, antes de realizar desarrollos completos. Nos permite claramente analizar mejor los cambios y determinar si los cambios son locales, no locales o arquitecturales[3].
- Del análisis de una Arquitectura se puede predecir los atributos de calidad del sistema, es decir, sus características en cuanto a rendimiento, modificabilidad, seguridad, escalabilidad y reusabilidad, o cualquier otra de las características no funcionales. (Hablaremos más delante de estos atributos de calidad).
- La Arquitectura de un sistema por sí sola no puede garantizar una funcionalidad o un atributo de calidad, pero las decisiones de diseño que sí permiten un conjunto de funcionalidades, pueden terminar minando la capacidad de una arquitectura para proporcionar un determinado atributo de calidad.

Aunque pareciera que una definición sobre Arquitectura Software debiera ser el punto de partida para el texto base, sin embargo no lo es y forma parte del contenido del segundo capítulo, ya que los autores prefieren comenzar su disertación con un primer capítulo sobre el ciclo de vida de la arquitectura software desde el punto de vista de sus influencias y de cómo el entorno técnico, de negocio o el entorno social afectan e influencian la arquitectura software. En dicho capítulo inicial se expresa que muchos de los interesados en la construcción de un sistema software van a poder listar requisitos funcionales y objetivos de negocio, pero el entorno empresarial puede establecer restricciones (monetarias, organizativas), el propio entorno tecnológico imponer también las suyas en incluso los arquitectos software proyectar ciertas influencias por su experiencia en proyectos anteriores. Todo esto modela el resultado final de una arquitectura software que no vendrá determinada únicamente por los requisitos funcionales que satisface. En opinión de los autores, el arquitecto software necesitará también evaluar una arquitectura software por las calidades o atributos no funcionales que soporta, algo esencial para asegurar que el sistema que se construye satisface las necesidades de los interesados.

El comenzar el texto base de esta manera, de forma muy intencionada, atrae la atención precisamente hacia el dominio de los atributos de calidad, de manera que centremos nuestra atención en ellos para ver cómo **los atributos de calidad llegan a ser hasta tal punto determinantes, que no prestarles**

[3] En este caso indicar que los cambios locales son los que se realizan en un solo elemento, mientras que los cambios no locales afectarían a varios. Sin embargo, en ambos casos no existen cambios en la arquitectura. Para estos casos, se emplea la denominación de cambio arquitectural. Como es comprensible, este último tipo de cambio es de mayor calado.

atención en el diseño de un sistema software, puede acabar en un sistema que sea tan inservible como uno que no cumpliera los requisitos funcionales[4].

1.2 ¿Qué son los atributos de calidad?

Leyendo el texto base tendremos una idea clara de qué son los atributos de calidad, normalmente por comparación entre requerimientos de calidad y requerimientos funcionales de un sistema software. Si los **requisitos funcionales** nos dicen lo que ha de **hacer** el sistema, los **requerimientos de calidad** nos pueden decir cómo debe **ser** el sistema. De esta manera algunos autores establecen una clasificación de atributos de calidad obtenibles de este tipo de requerimientos, en dos grupos:

- Atributos de calidad durante la ejecución: pueden medir la seguridad, usabilidad y observabilidad (monitorizable).
- Atributos de calidad ligados a la evolución del sistema: como la testabilidad, modificabilidad, extensibilidad y escalabilidad.

Mientras que el texto base se decanta por la siguiente clasificación:

- Calidades del sistema: disponibilidad, modificabilidad, rendimiento, seguridad, testabilidad y usabilidad.
- Calidades de negocio: como time-to-market, coste/beneficio, tiempo de vida del producto, integración con otros sistemas, mercado al que va dirigido el producto, todas ellas se ven afectadas por la Arquitectura y los otros atributos de calidad.
- Calidades sobre la propia Arquitectura: relacionadas con la integridad conceptual y que se ve afectada por la modificabilidad.

Los autores del texto base, aunque todavía cautos y conocedores del debate existente sobre los atributos de calidad y del diferente vocabulario que se genera entre comunidades especializadas en los diferentes atributos, aportaron en la tercera edición de su libro [2] esta interesante definición que no encontraremos en el texto base:

> *Los atributos de calidad son calificadores de los requerimientos funcionales o del producto al completo. Cualificar un requerimiento funcional puede ser un elemento tal que diga cómo de rápido una función debe realizarse o cómo de resistente a inputs erróneos. Cualificar el producto completo puede ser un elemento como el tiempo de despliegue del producto o una limitación de costes operacionales.*

[4] Me reafirmo en esta impresión sobre el porqué no comienza el texto base con una definición sobre Arquitectura Software. El texto base no sólo es un excelente libro sobre Arquitectura Software, además está bien escrito y es ameno. No en balde es un libro que ha influenciado a muchos otros autores. Por claridad expositiva, yo decidí comenzar de inicio definiendo qué es la Arquitectura de Sistemas Software. Sin embargo, he de decir que ya escritas mis primeras páginas, tuve curiosidad de ojear el texto base en su tercera edición de 2012. Para mi sorpresa, el orden expositivo en la tercera edición comienza con la definición de Arquitectura y su importancia. ¿Se retractan los autores casi una década después? Creo que no. Creo que los autores entienden que ha pasado el tiempo suficiente como para que muchos de los conceptos que propugnaron en las dos primeras ediciones están ya suficientemente asentados y no necesitan de un efecto literario para atraer la atención sobre ellos. Adicionalmente, se puede comprobar que en la segunda edición cuesta encontrar una definición clara y explícita de qué es un atributo de calidad. Se define por contraposición a los requisitos funcionales, por la explicación de cada uno de los atributos individualmente, clasificándolos, o por su relación clara con la arquitectura, pero sorprendentemente una definición genérica de todos ellos no existe en el libro. Esto no representa ninguna dificultad, pues cualquiera que lea el texto y lo comprenda, tendrá claro qué es realmente un atributo de calidad. Sin embargo, en la tercera edición sí que encontramos una definición explícita, síntoma creo yo de lo que ya he comentado. En la tercera edición, los autores cosechan el éxito de ver que este concepto está plenamente establecido.

Volviendo al texto base, la Arquitectura es el peldaño en el proceso de creación de software en el que podemos y debemos abordar los atributos de calidad. Atributos funcionales y de calidad son entendidos como ortogonales, en el sentido que la elección de determinados atributos funcionales no determina el nivel que se obtiene de determinados atributos de calidad. Sabemos que es posible implementar un software que de forma monolítica realice una tarea dada, sin importar el uso de una o varias posibles estructuras para implementarlo, con lo que estamos obviando que el simple hecho de descomponer dicho software en módulos lo haría más entendible y propiciaría seguramente una cierta variedad de usos. Esto lleva a afirmar que **la asignación de funcionalidades del sistema a determinadas estructuras software es lo que determina el soporte de una arquitectura para unos determinados atributos de calidad**.

Quizás no existe consenso completo en la comunidad de investigadores sobre los atributos de calidad, a la hora de clasificarlos o cuando entramos en el dominio concreto de uno de ellos. También podemos tener incluso la impresión de que los autores del texto base incluso han movido sus posiciones al definirlos en las diferentes ediciones de su texto. Sin embargo, creo que dichas variaciones no impiden identificar los atributos de calidad y en el caso concreto de los autores, podemos observar una aportación quizás ahora no tan evidente. El simple hecho de que en su nueva definición se distinga entre cualificar una función o al sistema (producto) completo, indica un cuidado especial en distinguir que los atributos de calidad no son exclusivos del sistema software en su conjunto, como quizás se entienda con la clasificación que se ofrece, sino que también es aplicable a cada uno de los componentes de un sistema[5].

De ahora en adelante, al igual que el texto base, me centraré en los atributos de calidad denominados: disponibilidad, modificabilidad, rendimiento, seguridad, testabilidad y usabilidad, teniendo en cuenta que serán conceptos no necesariamente aislados entre sí, es más, es sabido que **los atributos de calidad no se obtienen de forma aislada y coexisten con cierta tensión entre ellos**. Es normal que los intentos de maximizar o lograr un atributo de calidad añada dificultades o facilidades para obtener otro. El ejemplo paradigmático es que la obtención de casi cualquiera de ellos tiene un impacto negativo en el atributo de calidad relacionado con el rendimiento.

1.3 Escenarios para atributos de calidad

Quizás en las anteriores definiciones puede haber quedado algo oscurecida o poco aclarada la distinción entre atributo de calidad y requerimiento de calidad. Los autores del texto base defienden que una de las mejores maneras de evitar la ambigüedad o la colisión de términos que diversos autores emplean en diferentes atributos de calidad es establecer lo que denominan escenario de calidad.

> *Un escenario de atributo de calidad es un requerimiento específico para un atributo de calidad. Todo escenario tiene 6 partes: fuente del estímulo, estímulo, entorno, artefacto, respuesta y medida de la respuesta.*

El escenario nos pone en una situación muy concreta, a modo de requerimiento a obtener, totalmente dirigido a un atributo de calidad y no a un atributo funcional. Tenemos que aclarar entonces qué son cada una de esas 6 partes:

[5] Esto es de especial relevancia cuando hablamos de Software Component Architecture y de Off-The-Shelf components, como se verá más adelante.

- Fuente del estímulo: puede ser un humano u otra fuente automática o sistema software que genera el estímulo.
- Estímulo: es la condición que se debe considerar cuando llega al sistema.
- Entorno: Son las condiciones del estímulo y por tanto muy variadas y muy relacionadas con el tipo de atributo de calidad que consideramos.
- Artefacto: Es el artefacto o pieza del sistema que recibe el estímulo. Pudiera ser todo el sistema.
- Respuesta: Es lo que haremos, la actividad a emprender ante el estímulo.
- Medida de la respuesta: Toda respuesta deberá ser medible de alguna manera, para poder testar el requerimiento.

Hay que distinguir no obstante escenarios de atributos de calidad genéricos (escenarios generales), de aquellos que elaboramos específicamente para un sistema dado (escenarios concretos) que estemos construyendo.

No es mi intención realizar un resumen exhaustivo de todo el libro y materiales que he consultado, sino destacar aquellos elementos que son de uso más directo en este trabajo y dan una visión de la arquitectura software como la entienden los autores. Por tanto, no voy a dar una relación y descripción completa de todos los atributos de calidad, para lo que recomiendo el trabajo de investigación de Jesús Martín Fernández [3] recomendado por el equipo docente y el propio libro de texto para ejemplos de todos los escenarios. Sin embargo, sí quiero ver en detalle alguno de los que más pueden ayudar en la comprensión del presente trabajo. Por eso vamos a ver en detalle un escenario para el atributo de modificabilidad.

Primero tenemos que tener claro qué es la **modificabilidad**. En nuestro caso, es un **atributo asociado al coste del cambio, de manera que logremos controlar el tiempo/coste de implementar, testar y desplegar cambios**. Por tanto nuestro interés estará centrado en qué cambia, quién hace el cambio y cuando se realiza. Todo ello da lugar a nuevas implementaciones, tests que cuestan tiempo y por tanto dinero. Tiempo y dinero pueden ser la base de nuestra medida objetiva.

Lo habitual es contar con un escenario general de modificabilidad, a modo de tabla o plantilla como la siguiente:

Tabla 1 - Escenario general del atributo de calidad: Modificabilidad

Escenario general del atributo de calidad: Modificabilidad

Fuente del estímulo	*Puede ser un usuario final, un desarrollador, un administrador de sistemas…*
Estímulo	*Deseo de añadir, eliminar o modificar una funcionalidad o un atributo de calidad*
Artefacto	*Cualquier artefacto como el interfaz de usuario, plataforma, entorno, sistemas con los que interactúa…*
Entorno	*El de ejecución o producción, en tiempo de compilación, de construcción o de diseño.*
Respuesta	*Consiste en localizar los artefactos a modificar en la arquitectura, hacer las modificaciones sin afectar otras funcionalidades, probar las modificaciones y ponerlas en producción.*

Medida de la respuesta	*Es el coste medido en número de elementos afectados, esfuerzo en tiempo/persona, dinero… sumado en todos los elementos afectados.*

A partir de este escenario general, podemos obtener uno específico para un sistema concreto, sin más que seleccionar los elementos que son útiles para nuestro caso. Veamos el siguiente ejemplo:

Tabla 2 – Descripción de una prueba que da lugar a un escenario concreto para un atributo de calidad

> Imaginemos que tenemos un sistema de Telefonía IP que tiene una consola de gestión web que permite a los administradores dar de alta dispositivos (el aparato telefónico), usuarios y líneas de teléfono[6]. Durante las pruebas de aceptación, el administrador observa que la consola tiene la posibilidad de dar de alta un dispositivo y también darlo de baja. Estos dispositivos son tres diferentes modelos hardware denominados modelo básico, modelo de display extendido y modelo de display a color.
> Sin embargo, una compra de licencias de última hora pone a su disposición dispositivos telefónicos software (softphone) que pueden ser empleados también en el proyecto. El administrador desea poder dar de alta dichos teléfonos, pero no puede ya que determinados parámetros de configuración de los nuevos dispositivos no están presentes en la pantalla de alta. El administrador quisiera tener esta funcionalidad incluida en la próxima iteración del proceso de desarrollo, prevista para dentro de 2 semanas.

Con esta descripción está claro que podríamos elaborar el siguiente escenario concreto de modificabilidad:

Tabla 3 - Escenario concreto del atributo de calidad: Modificabilidad

Escenario concreto del atributo de calidad: Modificabilidad

Fuente del estímulo	*Administrador del sistema de Telefonía IP*
Estímulo	*Deseo de añadir la funcionalidad de dar de alta/baja dispositivos softphone para la próxima iteración de desarrollo.*
Artefacto	*Pantalla de alta de dispositivos*
Entorno	*Entorno de diseño del sistema, entorno de pre-producción.*
Respuesta	*Modificación de la pantalla de alta de dispositivos y del módulo de comunicaciones con el Call Manager (centralita de Telefonía IP en la que se da de alta el dispositivo), sin afectar al resto de módulos.*
Medida de la respuesta	*Se realiza el cambio en 10 días laborables (2 semanas).*

A través de este ejemplo, se puede ver porqué los autores afirman que

> *Los escenarios concretos de atributos de calidad juegan el mismo papel en la especificación de atributos de calidad, que los casos de uso juegan en la especificación de requerimientos funcionales.*

[6] De hecho estoy dando un adelanto del sistema concreto que consideraremos más adelante en el texto.

No es que los escenarios generales y concretos de atributos de calidad nos den un método de generación de escenarios, pero sí que nos proporcionan una anología muy poderosa y útil por su comparación con los casos de uso. Por supuesto, tampoco hemos de dejar de apreciar que suponen una poderosa herramienta de comunicación con todos los interesados en la construcción del sistema.

2. Obtención de la Calidad en la construcción de un sistema software

2.1 Tácticas

El diseño de un sistema es un conjunto de decisiones. Algunas de ellas estarán encaminadas a obtener la funcionalidad del sistema y otras a controlar la respuesta de los atributos de calidad. Los autores del texto base llaman tácticas precisamente a un conjunto de decisiones dirigidas a los atributos de calidad.

> *Una táctica es una decisión de diseño que influencia la consecución de una respuesta para un atributo de calidad.*

Por tanto, una táctica puede formar parte de las técnicas que un arquitecto software puede emplear para obtener atributos de calidad. En este contexto, a estas técnicas las denominamos **Tácticas Arquitectónicas**[7]. Nos ayudará a aclarar el concepto, las siguientes características que cumple una táctica:

- Una táctica está focalizada en la respuesta de un único atributo de calidad.
- Una táctica puede refinar a otra. Son refinamientos que un arquitecto puede emplear para hacer de una redundancia algo más concreto, por lo que puede haber una **jerarquía de tácticas**.
- Dentro de una táctica no hay consideraciones sobre compromisos o equilibrios para aplicarla[8]. Estos compromisos deben ser controlados y considerados por el diseñador. Es un aspecto en el que difiere de los patrones arquitectónicos, que ya tienen dichos equilibrios incluidos en el patrón.
- La gran diferencia entre patrón y táctica es que **los patrones empaquetan tácticas. Las tácticas son los bloques constructivos con los que creamos patrones arquitectónicos**.

La mejor manera de entender las tácticas es aplicarlas en su contexto. Por tanto vamos a hacer un repaso a continuación de una táctica para un atributo concreto, que ayudará mucho a ilustrar todos los conceptos, pero antes, revisaremos qué es un patrón arquitectónico, un concepto que ya ha aparecido y que necesitaremos también.

---

[7] Nótese que estoy realmente empleando la definición de táctica de [2] y no de [1]. En la segunda edición del texto base esto se denomina Estrategia Arquitectónica, pero me ha parecido más apropiada y focalizada la nueva definición.

[8] Debo destacar que aporto una particular traducción de la palabra "tradeoff" (también trade-off). En [1] se emplea sobre todo en el contexto social de relación entre los interesados (stakeholders), sobre todo cuando el diseñador debe llegar a acuerdos para el diseño con todos ellos. En este contexto es buen sinónimo de "exchange" (compensación) o "compromise" (término medio o solución intermedia). Ambas son las traducciones de wordreference (English-Spanish) y los sinónimos del diccionario inglés Collins. Sin embargo, en [2] encontramos tradeoff en el contexto específico del diseño, en una tarea específica del diseñador que implementa una táctica. Sabemos que los atributos de calidad coexisten con tensiones entre ellos y la táctica no está pensada para solucionar todas esas tensiones porque está dirigida a un único a tributo y una única respuesta. Es magnífica en ese contexto la palabra "equilibrio" propuesta por el diccionario de traducción English-Spanish de Oxford. Los patrones arquitectónicos tienen resueltos muchos equilibrios entre atributos de calidad, porque obviamente muchos de ellos implementan varias tácticas que pueden ir dirigidas a uno o varios atributos de calidad. Claramente esto es algo que no existe en una táctica.

2.2 Patrones Arquitectónicos

Los autores del texto base no aportan una gran cantidad de información sobre los patrones arquitectónicos[9], de manera que el concepto queda más claro para aquellos ya introducidos en el mundo de los patrones de diseño en programación orientada a objetos debido a alguna analogía con ellos, o para aquellos que directamente conozcan el trabajo de Mary Shaw y David Garlan [4] citado en el texto y que ha sido objeto de estudio en la Asignatura Arquitectura de Sistemas Software del Máster.

De hecho la visión ofrecida sobre los Patrones Arquitectónicos es exactamente la de Shaw & Garlan, pero reconociendo que en el momento de la escritura del libro, emplear Estilos Arquitectónicos como Shaw & Garlan ya había perdido la batalla contra la palabra Patrones y era más ampliamente utilizada por la comunidad de investigadores en Ingeniería del Software. Por eso me centraré más en una visión como la ofrecida por la tercera edición del texto. En esta edición podemos comprobar que la visión de los autores ha aumentado ofreciendo un catálogo más amplio de lo que se considera hoy en día un Patrón Arquitectónico y además ofrece una definición que resulta de interés:

> *Un Patrón Arquitectónico es un conjunto empaquetado de decisiones de diseño que se puede encontrar repetidamente en la práctica, que tiene propiedades que le permiten ser reutilizado y que describe una clase de arquitecturas.*

Es por esto que ya había avanzado que las tácticas son bloques de construcción para los Patrones Arquitectónicos. La mayoría de Patrones Arquitectónicos emplean varias tácticas y por eso se dice que un patrón empaqueta tácticas.

El hecho de decir que un Patrón Arquitectónico se "encuentra repetidamente", no indica más que la historia que han seguido para ser descubiertos, que ha sido encontrar sus características en común aplicadas en problemas reales. Por eso se dice que no se inventan, se encuentran. Si realmente hay un Patrón en la resolución de un tipo de problema, emergerá esa "clase" arquitectónica debido a la reutilización que se puede hacer de ella.

No nos extrañará entonces con esta definición que en [2] se diga que **un Patrón Arquitectónico establece una relación entre un contexto, un problema y una solución.** Dentro de la solución tendremos un conjunto de elementos, un conjunto de mecanismos de interacción, una disposición topológica de los elementos y un conjunto de restricciones semánticas.

Esta forma de tipificar el Patrón Arquitectónico mediante el triplete (contexto, problema, solución) recuerda mucho a la de los Patrones de Diseño y sirve de plantilla para dejarnos claro qué atributos de calidad proporciona el patrón mediante la configuración estática y dinámica de los elementos. Con la siguiente tabla vemos una explicación de todos los ingredientes del Patrón Arquitectónico:

Tabla 4 - Patrón Arquitectónico = (contexto, problema, solución)

Patrón Arquitectónico	
Contexto	*Situación del mundo real y recurrente que da lugar al problema*
Problema	*Problema apropiadamente generalizado que surge de un contexto. Se describen las variantes del mismo y los atributos de calidad que deben cumplir*

[9] Sorprendentemente, antes de entrar realmente en los temas de diseño de la arquitectura, sólo en los breves apartados 2.3, 5.8 y 5.9.

Solución		*Una solución arquitectónica y abstracta que describa las estructuras arquitectónicas para resolver el problema y el balance de fuerzas entre los diferentes elementos. La solución describe responsabilidades, relación estática y/o dinámica entre los elementos*
	Conjunto de elementos	*Repositorios de datos, procesos, objetos….*
	Conjunto de mecanismos de interacción	*Llamadas a procedimientos, eventos, bus de mensajes….*
	Conjunto de Restricciones Semánticas	*Que incluyen la topología, comportamiento de los elementos y mecanismos de interacción*

El catálogo de Patrones Arquitectónicos sigue creciendo y es mucho más amplio que el detallado por Shaw & Garlan en [4] y también mayor que el del texto base en [1], admitiendo los autores que por su naturaleza, todo catálogo de patrones es incompleto.

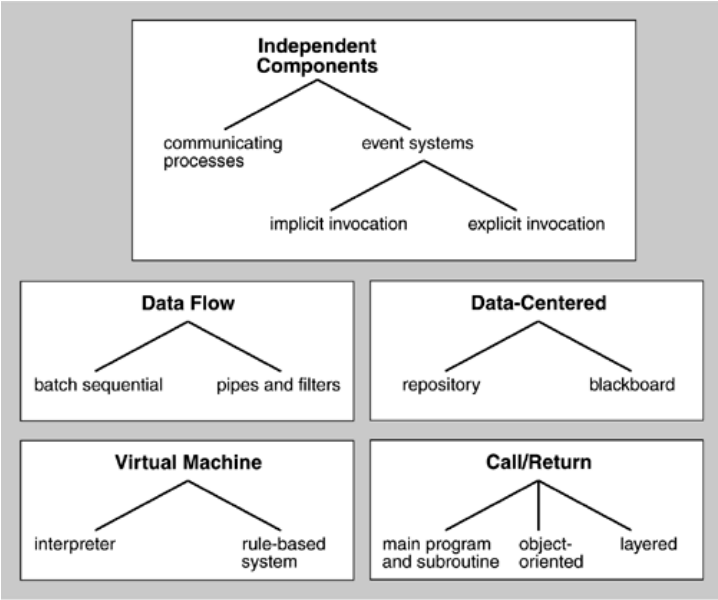


Imagen 1 - Patrones Arquitectónicos en [1] siguiendo la línea marcada por Shaw & Garlan en [4]

Respecto a la clasificación de los Patrones Arquitectónicos, en [1] se prefirió la más tradicional que se muestra en la imagen 1, pero en [2] encontramos una clasificación mucho más audaz. Consiste en **clasificarlos según el tipo de elementos predominantes que nos muestran**. Por eso eligieron clasificarlos en **Patrones de Módulo, de Componentes y Conectores y de Localización**. Estos últimos muestran una combinación de elementos software que caen dentro de las dos anteriores.

En la siguiente tabla se pueden ver todos los que se presentan en [2] en su nombre original en inglés para mejor compararlos con la clasificación de [1].

Tabla 5 - Listado de Patrones Arquitectónicos según [2]

Listado (incompleto por naturaleza) de Patrones Arquitectónicos según [2]

Module Patterns	*Layered*
Component-and-Connector Patterns	*Broker, MVC, Pipe-and-Filter, Client-Server, Peer-to-Peer, SOA, Publish-Subscribe, Shared-Data,*
Allocation Patters	*Map-Reduce, Multi-tier*

Lo que más llama la atención de la clasificación que admiten incompleta, es la inclusión de Patrones Arquitectónicos como P2P o SOA, por ejemplo, toda una llamada de atención sobre el hecho de que

como Arquitectos Software tenemos que estar atentos a la emergencia de una solución en forma de patrón[10].

2.3 Cómo emplear las tácticas en el marco de un escenario concreto: Ejemplo con el atributo de usabilidad

Anteriormente ya comenté que rechazaba dar detalle de todos los posibles escenarios genéricos para todos los atributos de calidad, con ejemplos específicos para cada uno de ellos, remitiéndome a otros autores para su consulta. Sin embargo, sí quisiera ilustrar toda la información completa sobre cómo emplear una táctica, con la referencia a su escenario. Por no volver al atributo de modificabilidad que empleé antes, vamos a ver otro, el de usabilidad, que los autores consideran íntimamente relacionado con la modificabilidad.

Primero, hemos de partir del concepto de usabilidad**. La usabilidad se puede medir en cómo de fácil sea para un usuario realizar una tarea dada y en el soporte que el sistema proporciona al usuario para ello**. Por tanto, es fácilmente entendible que tenga las siguientes áreas:

- Capacidad del sistema para el aprendizaje: ¿Qué puede hacer el sistema para facilitar su aprendizaje?
- Uso eficiente del sistema: ¿Qué puede hacer el sistema para que su uso sea eficiente por parte del usuario? (Como realizar varias actividades simultáneas, ser capaz de agregar datos…).
- Minimizar el impacto de errores: ¿Qué puede hacer el sistema para que un error humano tenga impacto mínimo? (como proporcionar la opción de cancelar, deshacer).
- Adaptación a las necesidades del usuario: ¿Cómo podemos hacer la tarea del usuario más fácil?
- Incremento de la satisfacción y confianza: ¿Cómo damos confianza al usuario en que la acción correcta se está llevando a cabo?

Son todas áreas que aparecen perfectamente en un escenario general sobre usabilidad como el siguiente:

Tabla 6 - Escenario general del atributo de calidad: Usabilidad

Escenario general del atributo de calidad: Usabilidad

Fuente del estímulo	*El usuario final, aunque posiblemente en un rol especializado*
Estímulo	*El usuario final intenta emplear el sistema eficientemente, quizás aprender a usarlo, a minimizar el impacto de errores, adaptar el sistema o configurar el sistema*
Artefacto	*Sistema o una parte de él con la que el usuario final interactúa*
Entorno	*El de ejecución o de configuración*
Respuesta	*El sistema debe o bien proporcionar al usuario las características que necesita o*

[10] En estos detalles es donde el texto en cualquiera de sus ediciones demuestra su claridad de exposición y sobre todo su profundidad conceptual. Parece que nos invita claramente a reflexionar sobre el siguiente hecho: ¿Porqué le llaman tecnología o arquitectura cuando quieren decir Patrón?. Es muy común encontrar en las descripciones comerciales de un producto "Tecnología SOA", "Arquitectura P2P", etc… hablando poco o nada de los atributos de calidad del producto. Afortunadamente el arquitecto software puede ir a su catálogo y revisar cuales son las fortalezas y debilidades de dicho Patrón Arquitectónico en cuanto a atributos de calidad. Pero no olvidemos que el Patrón no impone reglas inviolables a la hora de implementarlo, ya que se pueden buscar compromisos en busca de favorecer un determinado aspecto de calidad a la hora de implementar el patrón, aun en detrimento de otros.

	anticiparse a las necesidades del usuario
Medida de la respuesta	*Una o varias de las siguientes: tiempo de una tarea, número de errores, número de tareas terminadas, satisfacción del usuario, ganancia de conocimiento del usuario, ratio de operaciones éxito/fracaso o tiempo/datos perdidos cuando sucede un error*

Cuando nos enfrentamos a un escenario concreto, las tácticas de usabilidad que se nos proponen estarían clasificadas en dos grandes grupos. Debido a que los investigadores en Interacción Hombre-Máquina han acuñado los términos de iniciativa del usuario, iniciativa del sistema e iniciativa mixta para discernir quién toma la iniciativa en realizar una acción, **los escenarios de usabilidad combinan iniciativas desde ambas perspectivas: iniciativa del usuario e iniciativa del sistema**[11]. En la siguiente imagen podemos ver un gráfico de la jerarquía de tácticas de usabilidad

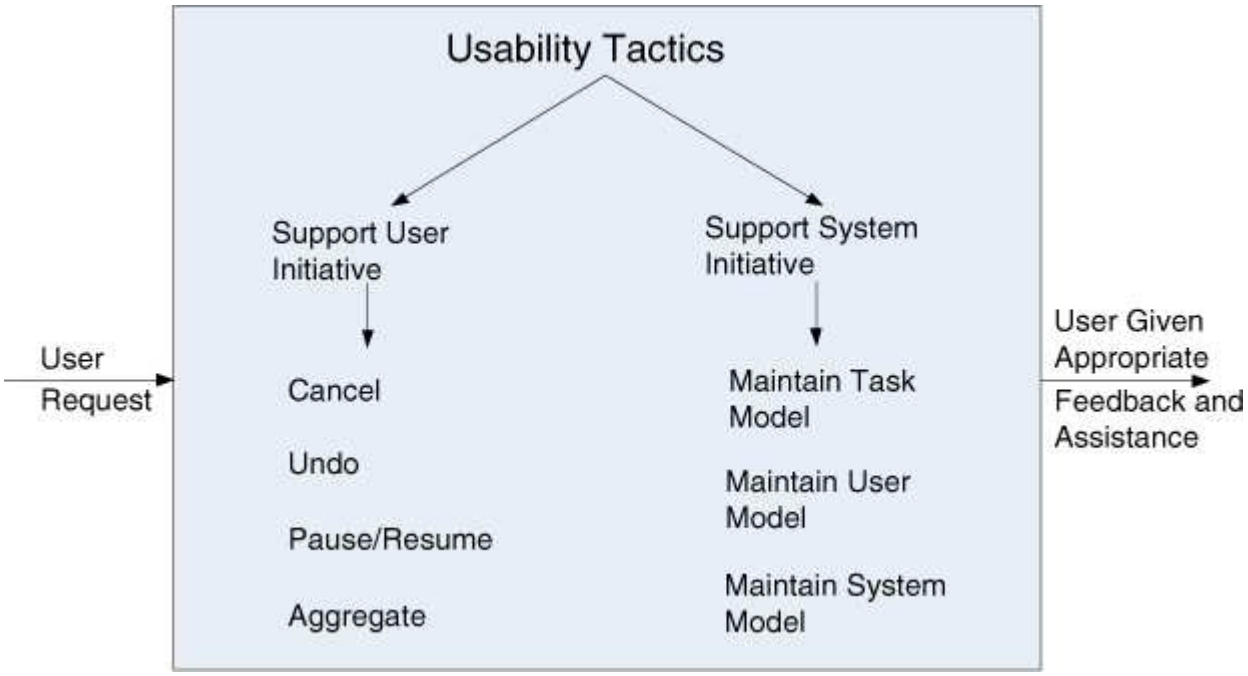


Imagen 2- Jerarquía de las tácticas de usabilidad según [2]

Lo primero que llama la atención en la imagen 1 es el empleo de la palabra "support". Se nos está indicando que **el arquitecto ha de diseñar el sistema para la iniciativa del usuario, colocando y asignando responsabilidades en el sistema para responder a los comandos del usuario**[12].

[11] Si bien no existe a penas diferencia entre [1] y [2] sobre los escenarios de usabilidad, sí que existe mucha diferencia en cuanto a las tácticas de usabilidad. En mi opinión el texto base realiza una mezcla de conceptos poco acertada. Por un lado clasifica las tácticas de usabilidad en dos: "runtime tactics" y "design-time" tactics. Todas las incluidas como tácticas en tiempo de ejecución son clasificables según la iniciativa del usuario o del sistema como se propone en la tercera edición, y finalmente esta clasificación forma parte de la jerarquía. Las denominadas tácticas en tiempo de diseño no es sino la tomar como táctica la aplicación de un patrón de diseño que independice el interfaz del usuario del resto de la aplicación (como MVC, por ejemplo). En el texto base estas tácticas en tiempo de diseño, forman parte de la jerarquía de tácticas de usabilidad junto con las de iniciativa del sistema o del usuario. Sin embargo en ambas ediciones, más claramente en la tercera, se nos insiste en la distinción entre patrón y táctica. Por este motivo creo yo que en la tercera edición se ha eliminado de la jerarquía de tácticas de usabilidad el empleo de patrones que independicen el interfaz del usuario.

[12] Volveremos más adelante sobre el concepto de "responsabilidad", que es fundamental en el presente trabajo. De momento, ya que se emplea "colocar o asignar una responsabilidad", debemos entender que se trata de una asignación de una funcionalidad a uno de los elementos presentes en una de las diferentes estructuras o representaciones del sistema.

Por tanto debe existir para cada táctica (que implica un comando o iniciativa del usuario), un soporte adecuado:

- Cancel: cuando el usuario cancela, el sistema debe de estar escuchando para responder y dar una respuesta adecuada como terminar la tarea y liberar recursos.
- Undo: para proporcionar una acción de deshacer, el sistema debe ser capaz de guardar información del estado previo.
- Pause/Resume: Se puede imaginar que para volver a continuar una tarea pausada se necesita información de estado, además de la capacidad de liberar recursos durante la pausa.
- Aggregate: Se trata de liberar al usuario de operaciones repetitivas mediante la aplicación de la misma operación a un grupo de objetos.

Lo mismo sucede del lado de la iniciativa del sistema. **Cuando el sistema toma la iniciativa depende de un modelo del usuario, de la tarea que éste realiza o del estado del sistema. Las tácticas de soporte a la iniciativa del sistema son aquellas que identifican el modelo que el sistema usa para predecir su propio comportamiento o la intención del usuario**.

- Maintain Task Model: el modelo de la tarea se puede emplear para ayudar o asistir al usuario.
- Maintain user model: representa el conocimiento que tiene el usuario del sistema e incluye su comportamiento en forma de respuesta esperada o la personalización del interfaz para él.
- Maintain system model: Es la situación en la que el sistema mantiene un modelo de sí mismo. Es el típico caso de una barra de progreso que indica el tiempo necesario para completar una tarea en curso.

Ahora que conocemos lo que es la usabilidad, su escenario general y las tácticas aplicables, podemos ver qué se podría hacer ante la siguiente situación que proporcionan los requerimientos para un escenario de usabilidad concreto basado en el ejemplo del sistema de Telefonía IP ya mencionado anteriormente.

Tabla 7 – Descripción de una prueba que sirve para generar un escenario concreto de usabilidad

El administrador del sistema de telefonía IP observa que tiene en su consola de gestión una pantalla para dar de alta dispositivos, otra para dar de alta líneas[13] y una tercera para asociar dispositivos a líneas. Todas ellas son operaciones separadas e independientes. La pantalla de alta de dispositivos solicita la dirección MAC[14] del dispositivo, al igual que lo hace la pantalla de asociación de dispositivo a línea. Teniendo que crear la línea antes de asociar el dispositivo, más de una vez se ha equivocado y ha colocado la dirección MAC incorrecta en el segundo formulario. El administrador opina que el interfaz gráfico debería de hacer esto más eficiente y menos proclive a errores[15].

[13] La línea sigue el concepto tradicional que tenemos de una línea, es decir, es el objeto que tiene asociado el número de teléfono.

[14] MAC = Media Access Control o identificador único asignado a un interfaz de red en tecnologías como Ethernet. Típicamente son 6 grupos de 2 dígitos hexadecimales, como 00:05:9A:3C:7A:00. Cada grupeto de 2 dígitos hexadecimales representa un byte, de manera que son direcciones de 48bits. Un organismo internacional asigna las direcciones MAC a los fabricantes de hardware, de manera que es posible identificar al fabricante por los tres primeros bytes. Los tres siguientes bytes personalizan el interfaz de red específico que se ha fabricado. En el ejemplo, 00:05:9A corresponde al fabricante Cisco.

[15] El ejemplo es reflejo de una situación real en mi entorno laboral. He realizado una simplificación en aras de la claridad de exposición. La situación real de la aplicación, de la que se dará detalle más adelante, incluía también el alta de usuarios, que han de asociarse igualmente a la línea. En la asociación de usuario a línea era necesario repetir el dato del número de teléfono y el identificador de usuario, propiciando un error

Nuestro escenario concreto sería el siguiente:

Tabla 8 - Escenario concreto del atributo de calidad: Usabilidad

Escenario concreto del atributo de calidad: Usabilidad

Fuente del estímulo	*Administrador del sistema de Telefonía IP*
Estímulo	*Deseo de mejorar la productividad a la hora de añadir dispositivos al sistema que se van a asociar inmediatamente a una línea nueva*
Artefacto	*Pantalla de alta de dispositivos*
Entorno	*Entorno de producción.*
Respuesta	*Modificación de la pantalla de alta de dispositivos para que al finalizar el alta, presente la opción de crear una línea vinculada sin necesidad de repetir la MAC del dispositivo*
Medida de la respuesta	*Se disminuyen drásticamente los errores de vinculación de dispositivo a línea, satisfacción del administrador y disminución del tiempo de la tarea de crear líneas y dispositivos vinculados*

A la vista de este escenario concreto, ahora podemos decir que sería apropiado emplear tácticas de usabilidad con soporte a la iniciativa del sistema. Obviamente el sistema de Telefonía IP del ejemplo cumple los requisitos funcionales de crear dispositivos, líneas y asociar ambos. Sin embargo no puede realizar una operación de crear dispositivo y a continuación crear una línea que tenga asociada el dispositivo creado anteriormente. Con una táctica de mantenimiento del modelo de tarea (Task Model) se podría tener un modelo de tarea conjunta de alta de dispositivo con alta posterior de línea asociada. Dicho modelo de tarea mantiene datos de un paso a otro, para evitar la engorrosa situación de tener que proporcionar la MAC una segunda vez.

Es más, conocemos del primer escenario de modificabilidad con el que iniciamos el contacto con el ejemplo del sistema de Telefonía IP, que contamos con 3 diferentes modelos hardware y un modelo software adicional. Por tanto, un buen modelo de tarea, permitiría que al seleccionar en el primer formulario el nombre del modelo de dispositivo, el sistema nos rellenaría los tres primeros grupetos hexadecimales de la MAC que corresponden al fabricante de dicho modelo (adivinamos las intenciones del usuario, una táctica de iniciativa del sistema).

Sólo la táctica Task Model por sí sola, ya logra disminuir los errores de asignación línea-dispositivo y aumenta la productividad de la tarea. Es por tanto perfectamente computable el beneficio, tal y como buscamos en la implementación de los atributos de calidad.

Si a esta táctica le añadimos una de deshacer, es decir, con soporte a la iniciativa del usuario, tendríamos un sistema muy completo. Hay que tener en cuenta que no es lo mismo deshacer un error empleando las operaciones de baja de dispositivo/línea/asociación, con las que podríamos cometer el error de aplicar la operación a un elemento no erróneo y sumar así un segundo error, que proporcionar una operación de deshacer que elimina un error recién cometido.

similar al de repetir la MAC en las otras pantallas. Sorprendentemente los diseñadores implementaron todas las operaciones de forma aislada, creando un sistema que cumplía todos los requisitos funcionales, pero cuya usabilidad estaba muy por debajo de estándares mínimos.

Finalmente, no quiero dejar pasar la oportunidad de volver a comentar sobre la conexión entre el atributo de usabilidad y el de modificabilidad. Cuanto menos modificable sea el sistema, más tensión existirá con la usabilidad, que será más difícil de obtener en escenarios como el planteado debido a que básicamente las tácticas de usabilidad consisten en asignar responsabilidades en elementos de las estructuras del diseño[16].

2.4 Relación entre Tácticas y Patrones Arquitectónicos

Teniendo en este punto mucho más claros los conceptos de Táctica y Patrón Arquitectónico, es momento de insistir todavía más en su relación. Ya sabemos que la mayoría de Patrones se construyen a partir de un grupo de tácticas. Sin embargo, cabe decir que algunas esas tácticas se podrían haber elegido para promover un único atributo de calidad, o lo que resulta más común, que un grupo de ellas sirvan para un atributo de calidad, mientras que otro grupo trabaje en la promoción de un atributo de calidad diferente.

Para que un diseñador logre que un Patrón Arquitectónico funcione en un contexto determinado, tiene que examinar cuidadosamente por un lado los compromisos y equilibrios que realiza el Patrón en su diseño para promocionar unos atributos de calidad y quizás disminuir la fortaleza de otros, mientras que por otro lado debe analizar aquellos atributos de calidad que no intervienen en el diseño del Patrón pero que son importantes en la aplicación y que podrían verse afectados por la aplicación del Patrón.

Si revisamos uno de los Patrones Arquitectónicos de nuestra lista, uno de los más conocidos y empleados, el Patron de Capas, podemos ilustrar adecuadamente estos conceptos:

Tabla 9 – Descripción (contexto, problema, solución) del Patrón Arquitectónico de Capas

Patrón Arquitectónico de Capas (Layered Pattern)

Contexto	*Sistemas complejos necesitan desarrollar, evolucionar y mantener diferentes partes del sistema de forma independiente mediante una correcta separación de intereses*
Problema	*El software necesita segmentarse de tal manera que se permita un desarrollo, evolución y mantenimiento de las partes con escasa interacción entre ellas para facilitar portabilidad, modificabilidad y reutilización*
Solución	*Dividir el software en unidades denominadas capas. Cada una de ellas tiene un grupo de servicios cohesionado que se ofrecen únicamente a la capa*

[16] Ya he resaltado en varias ocasiones ciertas virtudes de la nueva edición del texto base, pero en esta ocasión quiero resaltar una del propio texto base. Existe en la segunda edición un magnífico texto en el tema de los escenarios de usabilidad generales denominado “Usability Mea Culpa or That’s not architectural”. Se trata de un texto resaltado en un cuadro en la pag 92, con explicaciones de apoyo al tema general. Su autor, R. Katzman, nos vienen a decir que existía la tendencia de considerar, por parte de algunos investigadores del mundo de la Ingeniería del Software, que el interfaz de usuario tiene que ver sólo con obtener de forma correcta los detalles de interacción con el usuario. Sin embargo, eso no lo consideraban en modo alguno detalles de tipo arquitectónico.
Según Katzman, las cualidades de usabilidad más difíciles de obtener, particularmente las más difíciles de añadir una vez el sistema ya está terminado, resultan ser precisamente aquellas que son arquitectónicas. A la luz de las tácticas expuestas y de mi experiencia personal, Katzman vuelve a dar en la clave y si se quiere responder a un requerimiento de usabilidad, hay que planificar en la arquitectura cómo obtenerlo. En mi experiencia personal con la aplicación de gestión de Telefonía IP, resultó evidente que los diseñadores no habían planificado para obtener atributos de modificabilidad ni usabilidad. Al manifestarse las deficiencias de usabilidad en el interfaz gráfico, las deficiencias en modificabilidad generaron gran retraso en la entrega del software porque el proveedor se encontró inmerso en realizar no sólo cambios locales o globales en los módulos, sino también importantes cambios arquitectónicos para soportar usabilidad con iniciativa del sistema mediante un modelo de tareas.

		inmediatamente superior a ella. *Puede haber variantes en que una capa superior acceda a servicios de otra inferior y no adyacente (bridging). Puede que en este tipo de variante se vea comprometida la portabilidad y modificabilidad.*
	Conjunto de elementos	*Los elementos son las capas, que son un tipo de módulo. Se representan como cajas apiladas en los diagramas. Cada capa debe indicar el conjunto de servicios que proporciona.*
	Conjunto de mecanismos de interacción	*La relación es de "uso de" otra capa. Una especialización de la más genérica de dependencia. Se debe indicar si sólo se permite el empleo de la inmediatamente inferior, o si existe algún tipo de bridging.*
	Conjunto de Restricciones Semánticas	*Cada pieza de software estará sólo en una capa. Existirán al menos dos capas. Una capa no puede usar la inmediatamente superior ni ninguna otra por encima de esta.*
Debilidades		• *Las capas contribuyen a añadir complejidad y coste al sistema* • *Las capas afectan negativamente al rendimiento*

Se puede observar que añadimos un apartado específico de debilidades de nuestro Patrón Arquitectónico. Un patrón de capas está especialmente pensado en un contexto concreto en el que necesita incrementar atributos de calidad como portabilidad, modificabilidad y reutilización. Sin embargo, esto afecta negativamente a otro atributo de calidad como es el rendimiento. Siempre que hemos hablado de equilibrios o compromisos en el diseño, nos estamos refiriendo al conjunto de opciones en las que se podría mover el Arquitecto Software a la hora de implementar el patrón.

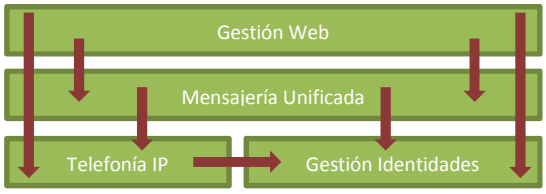


Imagen 4 - Diagrama mostrando las capas del sistema de telefonía IP

En la imagen 4 tenemos un diagrama de la estructura de capas de un sistema de Telefonía IP similar al que hemos empleado en ejemplos anteriores. Se trata de un ejemplo completamente real que nos ilustra que desafortunadamente, muchos de los diagramas que se nos muestran en la documentación de software propietario y especialmente en los panfletos comerciales, no ponen el cuidado en advertir claramente de posibles compromisos de diseño. En este ejemplo se muestra como realmente existe un bridging de la capa superior a cualquiera de las dos inferiores y una de las capas inferiores está accediendo a su adyacente. Si lo comparamos con el Patrón Arquitectónico original en su forma pura, sólo la capa intermedia se comporta de forma canónica. ¿Por qué han tomado los diseñadores esta opción? ¿Quizás por problemas de rendimiento? En este caso se debe a restricciones impuestas por el uso de

	Modifiability										
	Increase Cohesion		Reduce Coupling					Defer Binding Time			
Pattern	Increase Semantic Coherence	Abstract Common Services	Encapsulate	Use a Wrapper	Restrict Comm. Paths	Use an Intermediary	Raise the Abstraction Level	Use Runtime Registration	Use Startup-Time Binding	Use Runtime Binding
Layered	X	X	X		X	X	X			
Pipes and Filters	X		X		X	X			X	
Blackboard	X	X			X	X	X	X		X
Broker	X	X	X		X	X	X	X		
Model View Controller	X		X			X				X
Presentation Abstraction Control	X		X			X	X			
Microkernel	X	X	X		X	X				
Reflection	X		X							

Imagen 3 - Modificabilidad versus Patrones Arquitectónicos

componentes Off-The-Shelf. Obviamente este diseño ha arruinado completamente los atributos de modificabilidad originales del Patrón de Capas, aunque el fabricante puede seguir reclamando la portabilidades de su construcción debido a las tecnologías que emplea, como Java.

Por todo lo dicho anteriormente, no nos ha de extrañar que muchos autores que han escrito sobre Atributos de Calidad y Patrones Arquitectónicos, ofrezcan incluso tablas como la de la imagen 3. En esta tabla se presentan un conjunto de patrones en la columna de la izquierda. En la primera fila de la tabla tenemos de forma jerarquizada las tácticas de modificabilidad que consideró el autor. Las cruces obviamente nos indican las tácticas de modificabilidad concretas que está empleando un Patrón. Vemos patrones que usan un par y otros hasta siete diferentes, todo ello sin considerar qué otros atributos de calidad puede estar potenciando el patrón en cuestión.

2.5 Attribute-Driven Design

Ya hemos visto alguna de las herramientas que nos ha de servir a la hora de analizar y obtener atributos de calidad. Es normal que los investigadores se preguntaran por un **método que les permitiera diseñar una arquitectura para satisfacer requerimientos funcionales y de calidad al mismo tiempo**. Attribute-Driven Design (ADD) está concebido como una extensión de otros métodos como RUP, encajándose dentro de las actividades de diseño a más alto nivel.

> *Attribute-Driven Design es un proceso de descomposición recursiva en la que por cada iteración, empleamos Tácticas y Patrones Arquitectónicos para satisfacer Escenarios de Calidad, hecho lo cual, asignamos funcionalidad a los módulos proporcionados por el Patrón.*

ADD se encuentra en el ciclo de vida después de obtener el análisis de requerimientos, aunque no necesariamente ya totalmente completo. Como **la arquitectura va a ser moldeada por un conjunto de requerimientos funcionales, de calidad y de negocio, a estos elementos los llamamos Impulsos Arquitecturales**[17].

Hemos de tener claro que **el resultado de ADD son los primeros niveles de una vista de descomposición en módulos de una arquitectura (y otras vistas cuando sea apropiado). No todos los detalles de dichas vistas aparecerán de la aplicación de ADD, sino simplemente una descripción del sistema como contenedores de funcionalidades y la interacción entre ellos**. Como ADD es el primer paso en la articulación de una arquitectura, necesariamente hay resultados de grano gordo, pero son resultados necesarios para obtener las calidades buscadas. La diferencia entre una arquitectura como resultado de ADD y una arquitectura lista para implementar, está en las decisiones que resta tomar, como emplear un determinado Patrón de Diseño orientado a objetos o una pieza de middleware específica que traiga con ella ciertas restricciones. Por eso es además interesante obtener los resultados con ADD, porque podemos intencionadamente postponer estas decisiones con el objeto de ganar flexibilidad.

[17] La mejor traducción que se me ha ocurrido para “architectural driver”. También ímpetu, sinónimo de impulso. Obviamente impulso no en su acepción que incluye la irreflexividad, sino en aquella que instiga y que pone en movimiento y en crecimiento algo. Un amigo, también profesional de la informática me sugirió “promotor”. Pero en castellano el promotor no sólo instiga para que algo ocurra, sino que pone los medios para ello. En este caso se quiere dejar claro que somos nosotros quienes hemos de poner los medios a lo que sugieren los Impulsos Arquitecturales con ADD. También decir que en [2] se cambia este término por “Architectural Significant Requirements” (ASR), igualmente muy claro y que emplea para la suma de objetivos de negocio y atributos de calidad.

ADD se compone de 5 pasos[18] [2][3][5]:

1. Elegir un elemento del sistema a diseñar
2. Identificar escenarios de calidad que tiene que satisfacer y funcionalidad a desempeñar
3. Generar una solución mediante Patrones Arquitectónicos y tácticas que satisfaga los escenarios y asignar la funcionalidad a los módulos resultantes. Decidir sobre los interfaces entre módulos.
4. Decidir si se satisfacen correctamente los escenarios, si se han delegado responsabilidades a módulos hijos o si no se satisfacen en absoluto.
5. Volver a realizar 1 – 4 hasta terminar con todos los módulos.

En el primer paso, cuando estamos diseñando un sistema completamente nuevo y desde cero, lo normal es que el módulo a diseñar que tomamos en inicio es el propio sistema, pero como muy bien se nos advierte en [2], pudiera ser que dentro de las restricciones a nuestro problema nos veamos obligados a utilizar una pieza de software que ya existe en nuestra organización, o simplemente el problema que tenemos entre manos es la evolución de un sistema ya existente. Este es el motivo por el cual en 1 no se indica expresamente tomar todo el sistema, sino una parte de él, porque quizás esta sea una de las situaciones más comunes que nos vamos a encontrar.

Obviamente en el paso 4 nos estamos haciendo las preguntas necesarias para decidir si continuamos o no, para saber si hemos llegado a un punto muerto que nos haga replantear nuestro diseño. En ocasiones se dará la situación porque estamos aplicando las tácticas y patrones inadecuados, pero en otras ocasiones podemos llegar a la conclusión de que necesitamos volver a hablar con los interesados para llegar a acuerdos diferentes sobre el sistema.

Veamos un ejemplo práctico basado en el sistema de Telefonía IP que introdujimos cuando ilustramos los escenarios de usabilidad. Realmente es la situación previa que dio lugar al escenarios que describimos anteriormente, su precuela:

Tabla 10 – Precuela de la situación descrita en la tabla 7, para emplear ADD

La empresa ganadora de un concurso público para la implantación de un sistema de Telefonía IP comienza la fase de análisis con su cliente para la implantación del mismo. El sistema a implantar ganador del concurso dispone de varias cajas negras o appliances[19] que aportan la mayor parte de la funcionalidad del sistema. Una de ellas realiza la gestión de líneas, dispositivos, usuarios y la relación entre ellos. Otro de los appliances proporciona la funcionalidad de mensajería unificada. Como realizar las operaciones para dar de alta en los appliances todo lo necesario para que una persona pueda utilizar un teléfono es algo complejo, se ofertó en el concurso una consola de gestión que realiza las operaciones desde un único punto y de forma unificada. Esto es posible porque los appliances habitualmente disponen de APIs públicas para hacerlo. Una vez que tiene los requisitos funcionales para el alta de dispositivos, líneas y la

[18] Por claridad expositiva, no es puramente ni los pasos expuestos en [1] que son de explicación muy extensa y poco clara, ni exactamente los de [2] que son demasiado poco explícitos. También reconozco cierta influencia de un trabajo previo de uno de los autores, L. Bass, junto con dos asiduos colaborades (Bachmann y Klein), que cito como [5]. Como idea general, me centro en los conceptos de la tercera edición, añadiendo elementos de la segunda edición del libro para añadir esa claridad.

[19] Una solución comercial muy de moda, en la que en un único dispositivo hardware el fabricante ha empaquetado todo el software necesario para obtener las funcionalidades ofertadas. El appliance no necesita instalación de software, sólo configuración y las mismas necesidades que cualquier otro servidor de un CPD. En Telefonía IP ese appliance para las líneas y dispositivos que es el que gestiona las llamadas es el equivalente a una centralita (Call Manager).

asociación entre ellos, la empresa comienza el desarrollo de lo que denomina una aplicación con arquitectura web

Este caso real, continúa con la empresa adjudicataria envuelta en las dificultades de la crisis y obligada a realizar un ERE en el que acaba despidiendo a todo su equipo de desarrollo para centrarse sólo en los sistemas de telefonía. El desarrollo se subcontrata a otra empresa sin experiencia en sistemas de telefonía, a la que se le marcan las siguientes restricciones:

Tabla 11 – Restricciones al desarrollo para el subcontratista

- Debe diseñar una aplicación web Java (JEE), ya que es el medio tecnológico del que dispone el cliente
- Debe reutilizar los desarrollos ya efectuados por el equipo previo, que había comenzado por realizar una abstracción de servicios de la API del appliance para simplificar el problema

Con estas restricciones y la presión para responder a un proyecto novedoso deciden modularizar el sistema como indica la imagen 5. Realmente como primer intento, cumple perfectamente con el escenario que se les ha impuesto. Reutilizan en ToIP_API el trabajo previo que depende directamente de la API del appliance. Por otro lado tienen un módulo web en donde se usará directamente esa API desde tres formularios diferentes, cumpliendo todos los requisitos funcionales. El resto de la historia ya la conocemos, porque la implementación final tenía graves problemas de usabilidad.

Lo que más nos interesa ahora es qué sucede si seguimos este problema con ADD e intentamos solucionar el problema descrito en el escenario específico de usabilidad.

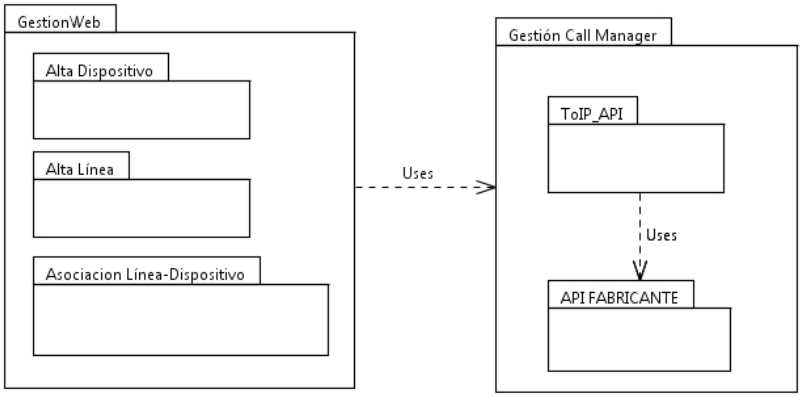


Imagen 5 - Diagrama de paquetes UML de la primera versión del sistema

Nuestro módulo a refinar será el Módulo Web. Nuestro requerimiento de calidad será el que ya describimos.

Sabemos que la aplicación va a crecer mucho más y para ganar usabilidad necesitamos obtener ese soporte con iniciativa del sistema a un Modelo de Tareas. El problema con formularios separados es que tal soporte no existe. Cuando damos de alta un dispositivo nada nos redirige a dar de alta una línea asociada[20]. Empleando un Patrón Arquitectónico MVC puedo obtener un elemento, el controlador, que realice esta tarea y permite llevar datos de vista a modelo y viceversa, que es lo que necesitamos para mejorar la usabilidad. El Modelo se encargará de realizar las operaciones y la vista gestionará los formularios. Así tenemos repartidas las responsabilidades a cada módulo y obtendríamos el siguiente diagrama de la imagen 6).

[20] Para los expertos en programación web que acaban de torcer el gesto, claro que tiene una solución muy chapucera, poner un enlace. Pero eso no es una solución con iniciativa del sistema, depende del usuario emplearlo o no y estos expertos ya conocen los problemas de mantenibilidad que conlleva este tipo de soluciones.

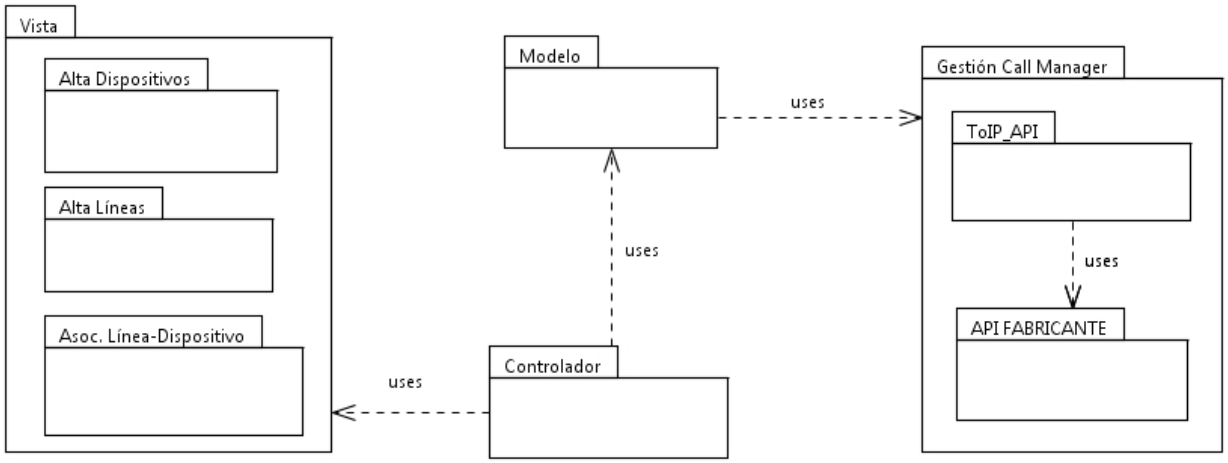


Imagen 6 - Primera iteración ADD. Resultado del paso de proposición de una solución basada en Patrones Arquitectónicos y/o Tácticas

En este paso también tenemos que revisar cómo interactúan los diferentes elementos. No necesitamos tampoco definir los interfaces pero sí tener una cierta idea de qué datos necesitan circular por el sistema. La API del Fabricante es un conjunto de servicios web, de manera que ya sabemos en qué modo interactúa ToIP_API con ella. El Modelo interactúa con ToIP_API instanciando sus objetos mediante patrones de diseño orientados a objeto de tipo creacional. Cuanto más claro tengamos estos métodos de interacción, quizás lleguemos a la conclusión de que la forma en la que el modelo va a llamar al API personalizada (ToIP_API) no va a ser exactamente como antes de plantearnos un escenario de usabilidad. Suponemos que la adición de más elementos y formularios al sistema puede generar problemas de modificabilidad. En lugar de resolver en esta iteración dicho escenario, decidimos delegar ese problema en un determinado módulo, ToIP_API.

De esta manera podríamos pasar a la siguiente iteración, tomando como input el módulo ToIP_API y un escenario de modificabilidad relacionado con los formularios. Como un formulario es de dispositivos, otro de líneas y otro de asociación entre ellos, parece lógico aplicar una Táctica de Modificabilidad como una que permita localizar los cambios y aporte coherencia semántica. Podría ser separar el Módulo ToIP_API en dos, uno que se ocupe de las operaciones sobre dispositivos y otro dedicado a las operaciones sobre líneas.

Recordemos que una arquitectura no viene definida generalmente con una única vista, la de descomposición de módulos en este caso. Nuestra arquitectura se debe completar con una vista adicional, que en este caso puede ser la de despliegue, donde se indica qué responsabilidades asignamos al hardware o componentes software middleware que estamos empleando, como en la imagen 7.

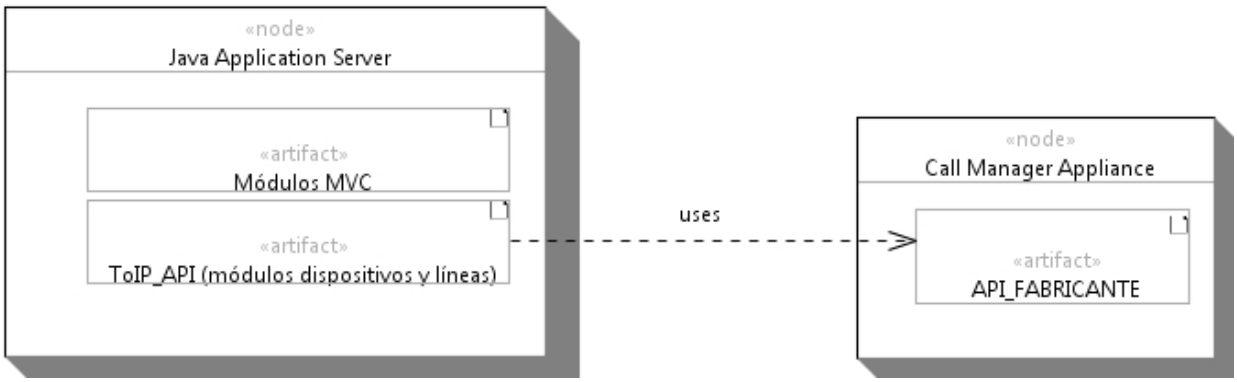


Imagen 7 - Diagrama de despliegue UML que sirve como vista que completa la arquitectura

La asignación de responsabilidades en este caso pasa por tener claro que finalmente empleamos un servidor de aplicaciones Java/JEE y por tanto los módulos MVC terminarán desplegados allí. También los dos módulos que contiene ToIP_API, que realizará a su vez llamadas a los servicios web que proporciona el fabricante del appliance.

2.5.1 Attribute-Driven Design 2.0: Como no puede ser de otra manera, el método evoluciona. De hecho en 2006 dos de los autores Bass y Clements firman un Technical Report del SEI conocido como ADD 2.0 [6]. Basado en este trabajo, otro de 2007 [7] ilustra todos los conceptos con un ejemplo práctico.

Ya adelanto que el núcleo central detrás de ADD sigue siendo el mismo tras una década, pero en ADD 2.0, se explica el proceso en 8 pasos añadiendo dos elementos diferenciadores:

- En la selección de los Impulsos Arquitecturales (Architectural Drivers), se propone clasificarlos por tres grupos de prioridades: High, Medium, Low, según elección de los interesados en el sistema. Además se propone la misma clasificación para indicar el impacto que tendrían en la arquitectura. De los dobletes (Prioridad, Impacto) se propone seleccionar sólo 6 de ellos con la mayor prioridad.
- Se hace mención explícita a los sistemas Legacy y a los sistemas COTS (Commercial Off-The-Shelf), para explicitar cómo se integran en las elecciones de una Táctica o Patrón Aquitectónico y para recordarnos que tendrán asignadas responsabilidades y mecanismos de comunicación.

La lista completa de pasos se puede ver en la imagen, tomada de [7].

Aunque la selección por prioridades de los Impulsos Arquitecturales no aparece en [2], podemos ver en esta edición del texto que existe una preocupación creciente por abordar los problemas derivados del empleo de sistemas COTS. En [1] tenemos un capítulo específico sobre el particular que no aparece en [2], pero por el contrario los sistemas COTS están mucho más presentes en los ejemplos.

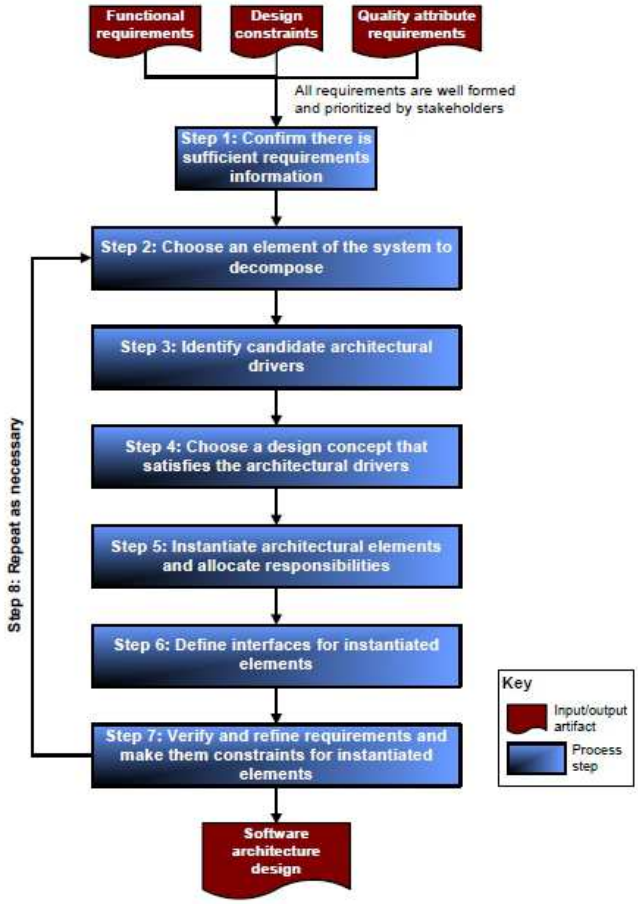


Imagen 8 - Pasos para aplicar el método en ADD 2.0

Yo comparto plenamente esta preocupación, ya que mucho del trabajo profesional que se lleva a cabo hoy en día impone muchas restricciones por su uso y es precisamente por ello que empleo el caso de Telefonía IP, un buen ejemplo de ello. Independientemente del número de pasos o cómo les llamemos el método es especialmente claro en su núcleo central y añade mucho valor cuando podemos hacer consideraciones arquitectónicas sobre los sistemas COTS.

3. ArchE

3.1 Architecture Expert (ArchE): Conceptos básicos sobre la herramienta y su arquitectura

Tal y como indica su manual [8], ArchE es una herramienta de ayuda al arquitecto software en la creación de arquitecturas más efectivas. Basándose en los conceptos que ya hemos visto en las anteriores páginas, ArchE parte de la premisa de que los Atributos de Calidad tienen una influencia dominante en la arquitectura final. Los promotores de ArchE creen también que la experiencia en el diseño arquitectónico aplicada a la creación de soluciones, puede ser capturada para atributos de calidad específicos y codificada como un conjunto de reglas. Si la herramienta permite además expresar los atributos de calidad por medio de escenarios de calidad concretos, se tienen unos buenos mimbres con los que diseñar una herramienta.

Como nos indican sus autores en [9], no pretende ser una completa herramienta de diseño, sino un asistente para el diseño. **ArchE tiene conocimientos sobre los atributos de calidad, pero ningún conocimiento sobre un dominio concreto, es decir, ninguno sobre la semántica del sistema bajo diseño**. Por tanto, nos puede dar consejos sobre cómo satisfacer requerimientos de calidad, pero no sabe qué significa este consejo para el arquitecto, respecto del dominio concreto del sistema a construir.

Después de lo dicho, no nos extrañará que ArchE esté construido en torno al famoso IDE Eclipse, tomando la forma de un plugin. Eclipse es hoy en día uno de los entornos de desarrollo que más facilidades proporciona para la construcción de entornos de desarrollo. ArchE emplea también el motor de reglas Jess que le permite obtener el elemento necesario para la creación de las características de un sistema experto.

Es precisamente como un sistema experto que lo describe uno de los documentos más tempranos que hablan sobre ArchE [10], un documento dedicado al diseño preliminar de la herramienta. En [10] se lo describe como un sistema experto con una arquitectura que sigue el Patrón BlackBoard.

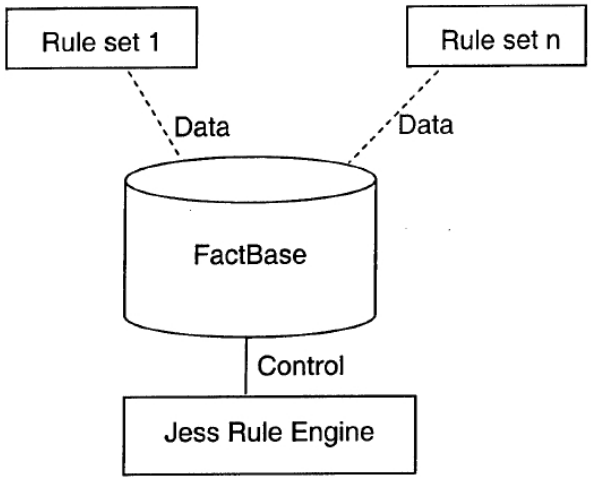


Imagen 9 - Relación entre ArchE y el Patrón BlackBoard según se describe en [10]

BlackBoard es un patrón arquitectónico muy empleado en el contexto de la inteligencia artificial. Parte de un conocimiento común que denominamos BlackBoard (pizarra), que es iterativamente actualizado por fuentes de expertos en la materia. Se comienza especificando un problema y se intentará llegar a una solución. Con cada una de las actualizaciones de soluciones parciales del problema que los expertos envían a la pizarra, se va generando la solución. Obviamente los expertos no van actualizando esas soluciones parciales si el estado de la pizarra no cumple las condiciones en las que ellos saben que pueden

aplicar ese conocimiento.

Esto último es lo que explica el papel del motor de reglas Jess. Cada regla se puede disparar por los datos del repositorio, pudiendo colocar a su vez nuevos datos en el mismo. En este esquema, los atributos de calidad forman parte del cuerpo de conocimiento y al repositorio de datos se le denomina FactBase. El papel de repositorio de datos lo realiza la base de datos relacional MySQL (ver imagen 9). **Todo el conocimiento que ArchE tiene sobre un atributo de calidad está empaquetado en lo que denomina un Reasoning Framework**. Desafortunadamente, **ArchE sólo dispone en su distribución oficial de dos Reasoning Frameworks, uno para Modificabilidad (Modifiability) y otro para Rendimiento (Performance)**.

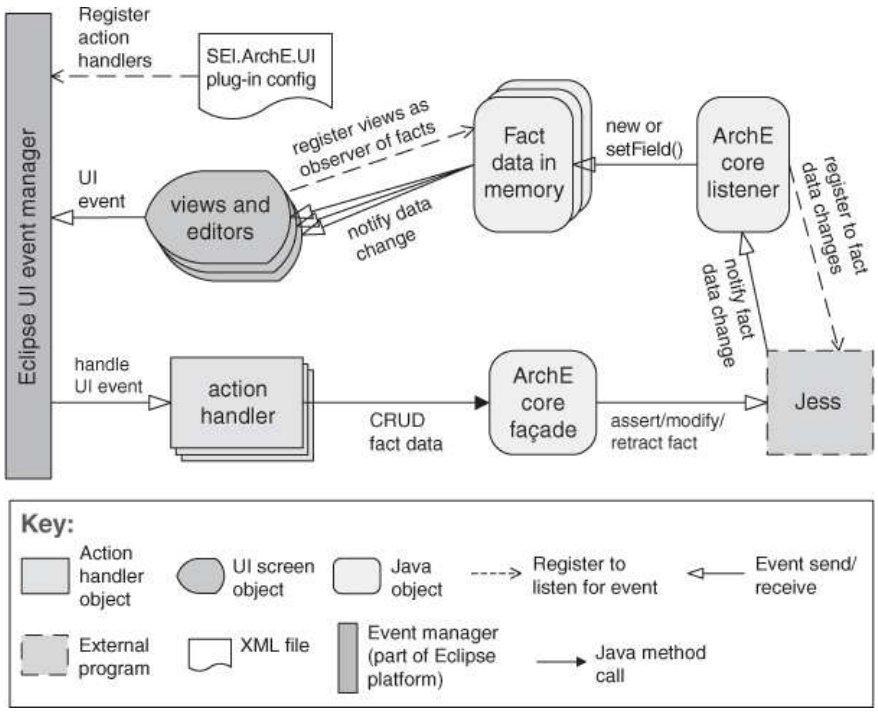


Imagen 10 - ArchE como ejemplo de aplicación de un Patrón Arquitectónico Publish-Subscribe

Quizás pueda parecer de poco interés para el presente trabajo hablar de la arquitectura de la herramienta, o sólo un mero ejercicio relacionado con la temática del trabajo, pero como se puede comprobar durante la instalación y también durante el manejo de la herramienta, resulta de vital importancia para entender qué estamos haciendo.

Curiosamente en la tercera edición del texto base los autores ponen como ejemplo de un Patrón Publish-Subscribe, precisamente a ArchE. Una solución de este tipo es la que permite que ArchE encaje en una herramienta como Eclipse. Se podrá comprobar ejecutando la herramienta, que no es necesario ejecutar comandos para que la herramienta realice determinadas acciones como recalcular las recomendaciones para resolver el problema o arquitectura bajo estudio. Conforme vamos añadiendo más elementos a nuestro problema, la herramienta va reaccionando y recalculando sus recomendaciones para resolverlo porque arquitecturalmente está pensado para hacerlo.

Habrá que tener cuidado con este comportamiento si se pretende estudiar el efecto de varios cambios consecutivos en el problema, ya que con cada cambio se recalcula todo. Si no recordamos el estado del problema y realizamos tres cambios consecutivos, la valoración que realiza ArchE sobre si el problema ha mejorado o ha empeorado se realiza siempre en base al último cambio, no respecto al estado del problema antes de la serie de tres cambios.

3.2 Instalación de la herramienta

Ha resultado de mucha ayuda tanto el documento guía de oferta del Trabajo de Fin de Máster proporcionado por el equipo docente, como los detalles de instalación que proporciona J. Martín en [3]. Sin embargo, aun así no pude hacerlo funcionar a la primera y entiendo que puede ser una instalación compleja, especialmente si no se tiene experiencia con tecnología Java y no se conoce bien Eclipse[21].

Voy a proporcionar los pasos que me han permitido reproducir la instalación en varias ocasiones. He repetido la instalación varias veces para poder anotar correctamente los pasos y estar seguro de la influencia de cada uno de ellos en el resultado final. El sistema operativo en el que instalo es Windows XP y empleo el software proporcionado por el equipo docente (las versiones de Eclipse, GEF, Jess, XmlBlaster, MySQL y ArchE proporcionadas en la página web del Trabajo del Máster).

1. Se instala la última versión disponible de Java 1.5 32bits, fijando la variable de sistema JAVA_HOME. Incluir el directorio bin de JAVA_HOME en el PATH.
2. La instalación de MySQL es de tipo "completa", con configuración estándar e instalando el servidor como un servicio (inicio automático con el sistema). Marcamos la opción de incluir los directorios binarios en el path. El password de root que le damos a la instalación de la BD es **ingesoft** y permitiremos login remoto. (Anotar el password será una necesidad si se quiere revisar qué hace la instalación de ArchE en la base de datos).
3. Descomprimir Eclipse en c:\eclipse
4. Descomprimir GEF en una carpeta temporal. Observaremos una carpeta denominada eclipse y en su interior contiene plugins de eclipse. Si realizamos un drag&drop de esta última carpeta sobre la carpeta de eclipse y solicitamos que se funda el contenido de ambas carpetas, el plugin estará instalado. Este es un procedimiento habitual con Eclipse, ya que los plugins están en la carpeta con ese nombre y basta con copiarlos a esa carpeta para su instalación.
5. Descomprimir JESS en una carpeta al mismo nivel que eclipse (c:\Jess71p1) y fijar la variable JESS_HOME. En el directorio de la instalación hay un directorio eclipse en cuyo interior tenemos gov.sandia.jess_7.1.0.zip. Este fichero, al descomprimirlo tiene un plugin de eclipse en una carpeta plugin. Si realizamos un drag&drop de esta carpeta, sobre la carpeta de eclipse y solicitamos que se funda el contenido de ambas carpetas, el plugin se instalará.
6. Descomprimir XMLBLASTER y fijar la variable XMLBLASTER_HOME, que permitirá usar el fichero bat proporcionado como ayuda (arranca_xmlblaster.bat).
7. El paso final es ejecutar el instalador de ArchE, que tiene capacidad para detectar que todo el software necesario está presente en el sistema. La lista de software se ha de satisfacer para que la instalación concluya correctamente.

3.2.1 Inicio de la herramienta: ArchE necesita que la base de datos MySQL esté arrancada y también XmlBlaster, ya que este último es un middleware orientado a eventos que proporciona las características arquitectónicas que hemos descrito antes. La base de datos se debe arrancar primero o directamente podemos elegir durante la instalación (o posteriormente) que arranque como un servicio del sistema. XmlBlaster se puede también instalar como servicio o arrancarlo con el fichero bat proporcionado.

[21] Se ha intentado infructuosamente la instalación con versiones de Java de 64 bits en Windows 7 x64.

Arrancar XmlBlaster con el fichero bat permite ver la consola y el disparo de eventos. Si tenemos algún problema que pensamos puede estar relacionado con la comunicación entre los diferentes componentes, algo que puede suceder si tenemos un equipo con varias IP, se recomienda mirar la dirección IP y puerto TCP en la que está escuchando XmlBlaster (se ve en los mensajes de la consola) y compararlo con lo que nos indican los comandos del sistema operativo, por ejemplo con netstat:

```
netstat –a –n
```

Con el comando podemos ver si efectivamente esa dirección IP está escuchando en el puerto indicado por la consola de XmlBlaster.

3.3 ArchE appliance

Entiendo que la instalación de ArchE no es sencilla para los no expertos en tecnología Java. Esta complejidad no ayuda en absoluto a la fácil transferencia de conocimiento sobre la herramienta. A esta situación, hemos de sumarle que la herramienta ya ha dejado de ser soportada por sus promotores en el SEI y también los avances propios en la tecnología Java en la que se basan todos los elementos software empleados para la instalación (salvo MySQL). Yo particularmente no he conseguido que funcione con Java 64bits y las versiones de Java con la que funciona todo el ecosistema de ArchE en el momento que se concibió la herramienta era Java 1.5[22].

Para facilitar esa transmisión de conocimiento, facilitar a otros el trabajo con ArchE y además evitar la dificultad de que a medio plazo alguno de los elementos software no se pueda encontrar, he empaquetado toda la instalación mediante un sistema de virtualización de libre acceso.

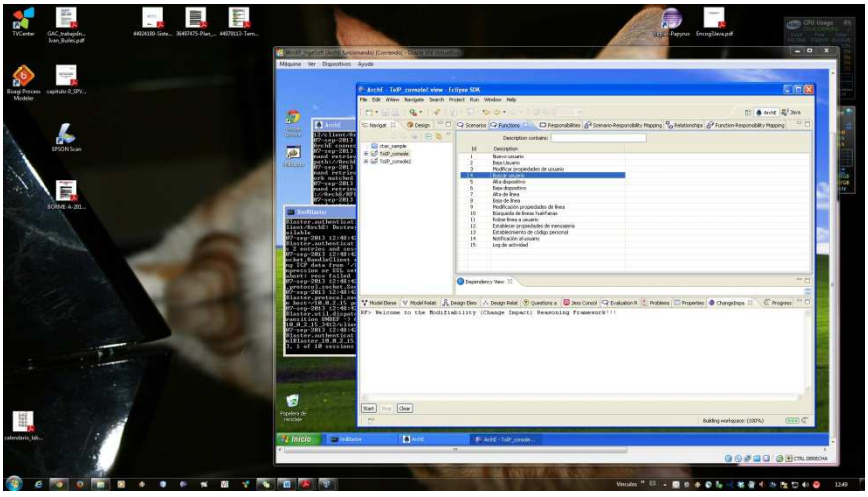


Imagen 11 - Imagen del ArchE appliance ejecutándose en una máquina con Windows 7 x64 Professional Edition. Es sólo un de los sistemas operativos sobre los que puede funcionar

La instalación de ArchE descrita anteriormente se ha realizado sobre una máquina virtual Windows XP, virtualizada mediante VirtualBox[23]. Aunque este software de virutalización era propiedad de SUN Microsystems y pertenece, al igual que Java, a Oracle, su naturaleza open source lo mantiene muy activo y de libre distribución. Forma parte del software que podemos encontrar en muchas distribuciones Linux, como Ubuntu.

[22] Java 5 nació en septiembre de 2004 y fue marcada End-Of-Life en octubre de 2009. La actual dueña de la tecnología Java, Oracle, permite su descarga para desarrolladores con el propósito de que puedan realizar tareas de debug y compatibilidad sobre sistemas antiguos. Soporte adicional no existe para esta versión de Java, si no es a través de un contrato específico de mantenimiento con Oracle. La siguiente versión Java 6 también llegó al fin de su soporte en febrero de 2013, siendo Java 7 la versión actualmente soportada.

[23] Virtualbox: Software de virtualización Open Source sobre plataformas Intel. https://www.virtualbox.org/

De esta manera, mediante la exportación del servicio virtualizado, la herramienta ArchE puede estar disponible para arquitectos que trabajen con sistemas operativos diferentes basados en plataforma Intel, como Windows 7/8 x86 o x64, Linux x86/x64 o los equipos de Apple con Mac OS X e incluso Solaris, ya que VirtualBox es instalable en todos ellos. Adicionalmente, el formato de exportación de la máquina virtual, también un formato abierto, permite que la máquina virtual sea importada no sólo por VirtualBox sino por otras herramientas del mercado como VMWare.

3.4 ArchE: conceptos fundamentales sobre su funcionamiento

ArchE se basa en tres conceptos fundamentales, dos de los cuales ya hemos ido revisando a lo largo del presente trabajo:

- Escenarios para atributos de calidad: ArchE proporciona formularios para que el diseñador introduzca las seis partes de un escenario. Forma parte de los requerimientos con los que razonará la herramienta y por tanto tendremos que subministrárselos antes de comenzar con el diseño.
- Tácticas arquitectónicas: Son las decisiones que podemos tomar que afectarán a un atributo de calidad en nuestra aquitectura. Durante el trabajo con la herramienta, esta nos podrá sugerir que utilicemos alguna táctica o patrón.
- Responsabilidades: Son actividades que realiza el software bajo diseño. ArchE emplea las responsabilidades como medio para expresar requerimientos funcionales. Forman parte de los escenarios y es el medio para que la herramienta pueda relacionar varios escenarios entre si.

En mi opinión, son las responsabilidades el concepto más difícil de comprender dentro de la herramienta. Resultó reconfortante ver que en [9] también resultó asi para los estudiantes. El concepto de responsabilidad se puede leer ya en el documento de diseño preliminar de ArchE [10] que está directamente extraído del trabajo de Wirfs-Brock y McKean [12]. En [9], los instructores solucionaron esta dificultad aludiendo al concepto de mapear una vista de requerimientos del sistema a una vista con propiedades funcionales. De hecho en ADD ya hemos hablamos de asignar responsabilidades a módulos, sin reparar mucho en lo que conceptualmente quería decir y nuestro testo base [1] era también el texto base de los estudiantes en [9].

3.4.1 El concepto de responsabilidad

proviene del trabajo de Wirfs-Brock y McKean [12], como nos dicen los autores de ArchE. Este trabajo está dedicado al diseño en programación orientada a objetos y resulta reveladora la definición que nos aporta:

> *Una responsabilidad se puede expresar como una frase de contenido general sobre un objeto software. Debe incluir tres elementos principales:*
>
> - *Las acciones que el objeto realiza*
> - *El conocimiento/datos que el objeto mantiene*[24]
> - *Las decisiones importantes que un objeto realiza y que afectan a otros*

Wirfs-Brock nos pone el ejemplo de una Kettle[25], de la que nos proporciona esas frases de contenido general que ayudan a entender qué hace el objeto:

[24] En la doble acepción que la palabra tiene en inglés y castellano: datos que **guarda** y también de los que realiza un **mantenimiento**.

- Verter su contenido sin salpicar ni derramar
- Contener agua que puede ser calentada hasta hervir
- Avisar cuando hierve el agua
- Proporcionar medios para moverla de forma segura

La idea de Wirfs-Brock es llegar a estas frases a través de diferentes fuentes, siendo los casos de uso una de las fuentes principales, pero también son otras fuentes otras características adicionales del software en construcción[26]. Nos dice que, **una vez identificadas las responsabilidades de nuestro sistema, deberemos asignarlas a los objetos**. Incluso defiende esta asignación como un proceso de refinamiento iterativo, porque cuanto más específica es una responsabilidad, más sencilla es de asignar, pero una descripción más tosca y general puede necesitar descomponerse en actividades más pequeñas. Luego la idea tiene un sentido coherente con lo que hemos venido haciendo hasta ahora en ADD, incluso en su metodología iterativa.

Wirfs-Brock nos dice que **existe una brecha entre las descripciones de los casos de uso y las responsabilidades de un objeto**. **Los casos de uso describen el comportamiento del sistema desde el punto de vista de la interacción con los actores. Es una perspectiva de un observador externo. No dicen cómo han de realizarse las acciones**, asi que es una idea más tosca de cómo el sistema funcionará y las tareas implicadas. **Es tarea del diseñador crear un puente en esa brecha, transformando las descripciones en los casos de uso en frases concretas sobre responsabilidades de acciones, información o toma de decisiones por parte de los objetos**. Esto es una situación perfectamente normal, porque los casos de uso son descriptivos, no prescriptivos. Nos cuentan una historia que sirve de guía general al diseño, pero intencionadamente deben dejar a un lado los detalles.

Esto explica por qué **en ArchE tenemos el concepto de Funciones y de Responsabilidades**. **Las funciones están más cerca de las descripciones en los casos de uso**, mientras que las responsabilidades están más cerca de la implementación. Donde Wirfs-Brock asignaba responsabilidades a objetos, nosotros **asignaremos las responsabilidades a módulos**.

3.5 Flujo de actividades para el diseño con ArchE

Cuando diseñemos un sistema, deberemos seguir los siguientes pasos con la herramienta.

1. Deberemos obtener los requerimientos funcionales y de calidad del sistema. Trasladaremos los funcionales a la herramienta por medio de agregar Funciones. Las funciones no tienen más atributos que su descripción y un identificador numérico. Se observará que la herramienta crea una responsabilidad del mismo nombre para cada función. Las responsabilidades sí tienen atributos adicionales.
2. Se debe indicar si existe alguna relación entre las responsabilidades (dependencia, inclusión…)
3. A continuación construimos los escenarios concretos de calidad.
4. Indicamos qué responsabilidades están relacionadas con un escenario dado.

[25] Kettle o Teakettle es el típico hervidor de agua en el Reino Unido, indispensable en una cultura del Te.

[26] Podemos estar haciendo un diseño desde cero de un nuevo sistema, pero tener ya la restricción de tener que emplear software COTS, por ejemplo. Esto no se refleja en un caso de uso, pero luego en una descomposición modular tendrá su impacto.

Imagen 12 - Diferentes pestañas en ArchE para ir rellenando la información

Tal y como podemos ver en la imagen 12, las funciones se irán introduciendo como una lista en "Functions", que a su vez irá creando responsabilidades en "Responsabilities". En "Function-Responsability Mapping" tenemos la relación uno a uno recién creada. En "Relationships" se deben introducir las relaciones entre responsabilidades. En "Scenario" debemos introducir los escenarios y en "Scenario-Responsability Mapping" debemos introducir la relación que tienen con las responsabilidades.

Muy importante: Sorprendentemente, **en el manual no se advierte de la imposibilidad de crear escenarios o relaciones entre responsabilidades cuando los marcos de razonamiento no están cargados**. Como podemos observar en la imagen 13.

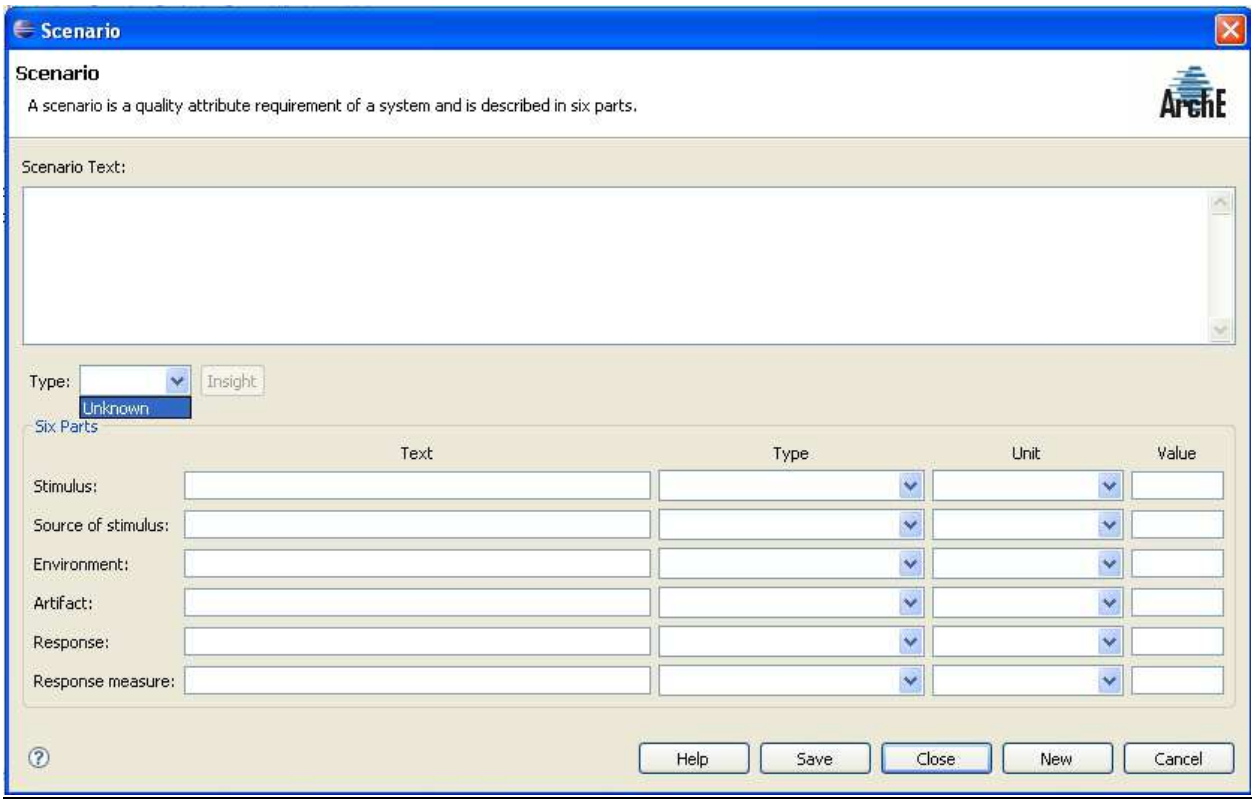


Imagen 13 - El escenario a crear no permite elegir el atributo de calidad cuando el marco de razonamiento correspondiente no está cargado

En todo escenario se debe indicar el atributo de calidad al que nos referimos, y para que dicha lista muestre atributos candidatos, es necesario que el marco de razonamiento correspondiente esté cargado. Esto se consigue seleccionando la secuencia de menús Window – Show View – Other, que nos mostrará una ventana de selección de vistas[27]. Entre ellas podemos seleccionar en el apartado ArchE External RF Samples una de las siguientes:

[27] Incluso para mí fue chocante esta situación, aun teniendo experiencia de desarrollo con Eclipse. No sólo en el manual [8] no se advierte de cómo arrancar un Reasoning Framework, sino que en [9] no se nos advierte de esta situación tampoco. En las clases descritas en [9] la práctica totalidad de los estudiantes tenían experiencia con Eclipse y contaban con un experto en la herramienta para guiarles. Es un aspecto del uso de la herramienta que de desconocerse te puede hacer pensar que está mal instalada y por ello no funciona correctamente.

- ChangeImpact Modifiability RF View (Reasoning Framework de Modificabilidad)
- ICM Performance RF View (Reasoning Framework de Rendimiento)

La selección de uno de los marcos de razonamiento hará que aparezca (imagen 14), normalmente en la zona de ventanas inferior, la ventana correspondiente a esta vista. La presencia del marco de razonamiento por sí sólo no hará que funcione lo que necesitamos, primero hay que pulsar el botón "Start" para iniciar el marco de razonamiento y que esté disponible para nuestras actividades de diseño.

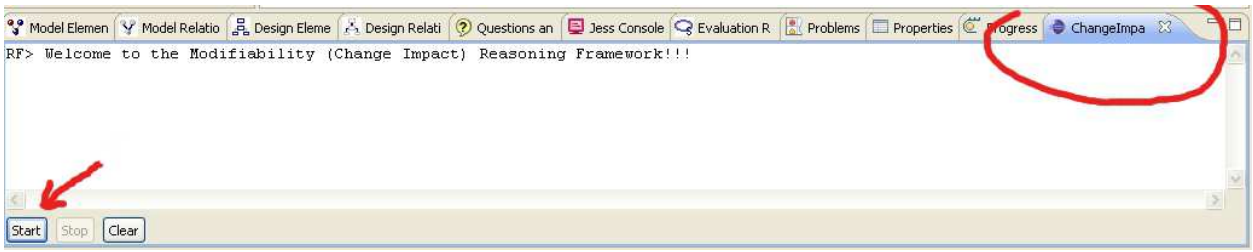


Imagen 14 - Vista de marco de razonamiento en ArchE

Terminada esta acción, será posible crear escenarios en los que utilicemos el atributo de calidad al que se refiere el marco de razonamiento y también crear relaciones entre las responsabilidades[28].

Finalmente ArchE nos mostrará si se satisface o no un determinado escenario mediante una bola de color gris (no se ha evaluado el escenario), verde (el escenario se ha evaluado y se satisface) o roja (el escenario se ha evaluado y no se satisface), según se muestra en la imagen 15.

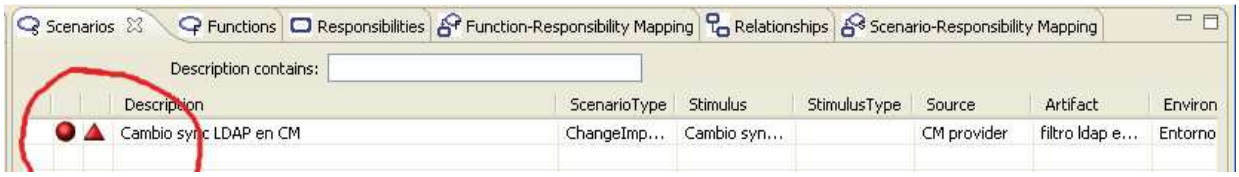


Imagen 15 - Indicaciones de ArchE sobre la evaluación del escenario

En la imagen 15 se observa también un triángulo. Cuando es de color verde indica que el diseño actual ha mejorado respecto del anterior para este escenario. Cuando es rojo indica un empeoramiento. Acercándonos al icono con el ratón se nos dice los porcentajes de mejoría/empeoramiento. ArchE está continuamente evaluando cualquier cambio que realicemos.

Finalmente, nuestra última actividad con ArchE y decididamente la más larga es

5. Aplicar tácticas sugeridas por ArchE o modificaciones propias con la finalidad de mejorar la arquitectura. Esto puede redundar en la aparición de nuevas responsabilidades.

ArchE nos informará de todas las sugerencias (ver imagen 16) mediante una bombilla en la vista "Question and Alerts". Una doble pulsación del ratón permitirá aplicar la táctica sugerida.

Eclipse funciona con perspectivas (formas de mostrarnos un proyecto que está adaptada al tipo de tecnología y lenguaje empleado) y vistas (diferentes aspectos de un mismo proyecto). El plugin de ArchE proporciona una perspectiva ArchE y varias vistas, entre ellas la de los reasoning frameworks. Obviamente usuarios experimentados de Eclipse llegarán a utilizar más rápidamente ArchE.

[28] Debo advertir igualmente de otro efecto: Cuando abrimos un proyecto con el que ya hemos trabajado antes, las relaciones entre responsabilidades han desaparecido. Sólo aparecerán cuando carguemos la vista correspondiente al marco o marcos de razonamiento empleados en los escenarios y arranquemos el marco de razonamiento tal y como he descrito en los pasos anteriores. Igual sucede con los escenarios, que sí son visibles, pero no editables.

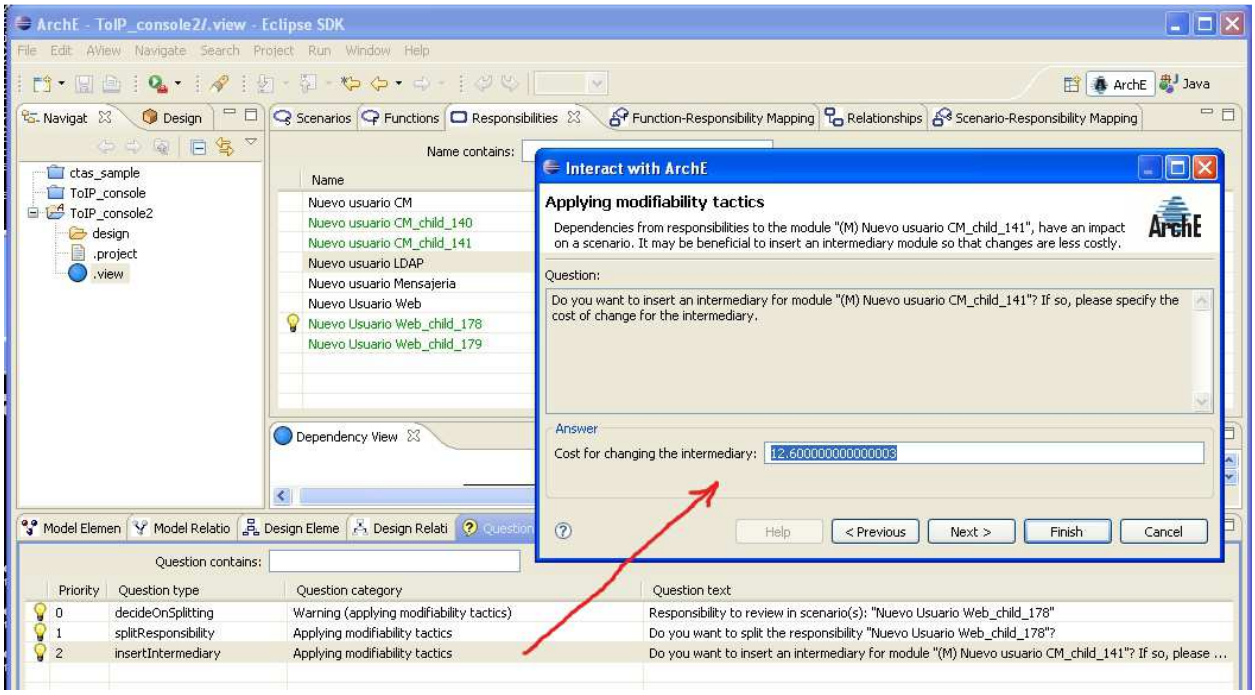


Imagen 16 - Bombilla indicando las sugerencias que genera la herramienta

En el caso de la imagen 16, al tratarse de una táctica de modificabilidad, se requere el dato del coste de aplicarla.

ArchE tiene también una vista de Evaluación de Resultados (imagen 17). En ella presenta con unas bolas de colores asociadas a las tácticas, el resultado predecido sobre la satisfacción o no de un escenario dado, en caso de que decidamos aplicar dicha táctica (en rojo no satisfacerá el escenario y en verde sí).

Imagen 17 - Vista de Evaluación de Resultados

Existe también un botón de evaluación de resultados que cambia los iconos por unos triángulos de colores que predicen el impacto de la táctica sobre el escenario (rojo = empeora, verde = mejora, ambar = no cambia).

Podemos realizar entonces un ejemplo realizando estas actividades, pero dado que ArchE tiene soporte sólo para dos Atributos de Calidad, a saber, Rendimiento y Modificabilidad y sólo hemos estudiado en detalle el atributo de Usabilidad hasta ahora. Será bueno que primero tengamos una perspectiva completa de alguno de ellos. A mi particularmente me interesa la Modificabilidad, que ya introdujimos anteriormente, pero de la cual no conocemos sus tácticas.

3.6 Atributos de Calidad con marcos de razonamiento existentes en ArchE: Tácticas de Modificabilidad

Ya hemos visto anteriormente[29] lo que es la modificabilidad y sabemos que se ocupa del coste del cambio. No hemos visto sin embargo sus tácticas y conocerlas es esencial para trabajar con ArchE. El principal objetivo de las tácticas de modificabilidad es controlar la complejidad de hacer cambios, así como el coste asociado.

En [2] los autores modifican ligeramente el marco en el que establecen sus tácticas de modificabilidad. Se considera por un lado que el concepto de **acoplamiento**, que es un enemigo de la modificabilidad. Cuando tenemos dos módulos que se solapan de alguna manera, un pequeño cambio puede afectar a ambos. **Se puede establecer una medida del acoplamiento como la probabilidad de que una modificación se propague entre módulos**. Un acoplamiento alto será negativo para la modificabilidad.

Otro de los conceptos que se emplean es la **cohesión**, que mide cómo están relacionadas las responsabilidades de un módulo. **La cohesión de un módulo es la probabilidad de que un escenario de cambio que afecta a una responsabilidad, termine afectando también a otras. Cuanto más alta la cohesión, menor probabilidad de que ese cambio afecte a múltiples responsabilidades**. Una cohesión alta será buena para la modificabilidad.

Finalmente, se interesan por cuándo ocurre el cambio en el ciclo de vida del desarrollo de software. **Si tenemos que acomodar un cambio, mejor que sea lo más tarde posible en el ciclo de vida del software**. Esto es una recomendación que realmente no tiene en cuenta el coste de preparar la arquitectura para ello y se denomina **defer binding**.

En la imagen 18 muestro la jerarquía resultante.

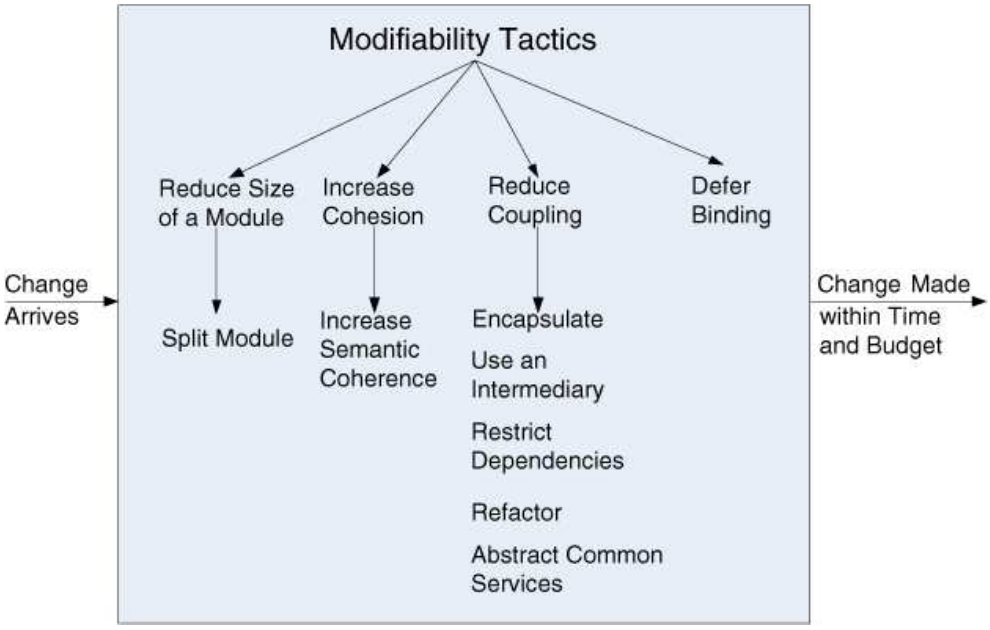


Imagen 18 - Jerarquía de Tácticas de Modificabilidad según [2]

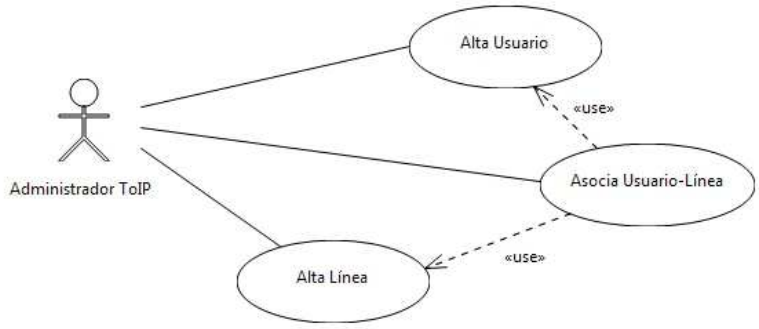
[29] Lo vimos en el apartado "Escenarios para Atributos de Calidad". Revisamos en esa sección el escenario general y también vimos un ejemplo de un escenario concreto.

Dentro de nuestra lista de tácticas, figuran entonces las siguientes:

- Reducción del tamaño de un módulo: Una táctica que separe (Split) un módulo en varios puede reducir el coste de hacer modificaciones si la separación en módulos es fiel reflejo del tipo de cambios que podría sufrir el módulo.
- Acoplamiento: Para reducir el acoplamiento entre módulos, tácticas que colocan intermediarios de varios tipos entre los módulos. El acoplamiento puede producirse por el tipo de datos (que el que uno produce sea consumido por el otro, por ejemplo), por la signatura del servicio que permite la relación entre ellos, por la semántica (de los datos o del servicio). El intermediario puede tomar la forma de diversos patrones arquitectónicos (facade, bridge, mediator, stategy…)
- Cohesión: Un módulo con baja cohesión puede ver aumentada su cohesión eliminando responsabilidades del módulo que no se ven afectadas por cambios que anticipemos.
- Defer Binding: Se pueden emplear tácticas en tiempo de compilación (reemplazo de componentes, parametrización de la compilación), en tiempo de despliegue (parámetros de configuración del despliegue), en tiempo de inicialización (ficheros de recursos) o en tiempo de ejecución (interpretación de parámetros, plug-ins, repositorios compartidos, polimorfismo, búsqueda dinámica de servicios, publish-subscribe….)

En nuestro trabajo con ArchE podemos ver cómo se nos ofrecen varias de estas posibilidades, que por supuesto pueden estar en la cabeza de un arquitecto experto. La novedad que proporciona ArchE es la capacidad de calcular cómo afectan los cambios a distintos escenarios cuando aplicamos una de las tácticas propuestas por ArchE, o a iniciativa del arquitecto.

3.7 Ejemplo de uso de ArchE: Asociación de usuarios a líneas en el sistema de Telefonía IP



Continuamos con el ejemplo que ya hemos ido viendo. En esta ocasión nos vamos a centrar en el alta de usuarios, líneas y su asociación. Los casos de uso son los presentados en la imagen 19 y que se describen en la tabla inferior.

Imagen 19 - Casos de uso para altas de usuarios y líneas en el sistema de Telefonía IP

Tabla 12 – Descripción de los casos de uso de la imagen 19

	Podemos describir una restricción del sistema de telefonía: El Sistema de Telefonía IP permite dar de alta usuarios, líneas y la asociación entre ambos. Un usuario que tiene una línea asociada, es el propietario de ese número de teléfono y el sistema puede informar de la identificación de llamada. La identificación de llamada incluye en este caso además del número llamante, el nombre y apellidos

del propietario de la línea[30].
En el diseño del sistema se está empleando como Call Manager un appliance que permite crear la línea a través de un API de forma inmediata, pero que sólo puede asociar a la línea usuarios presentes en un repositorio local del appliance. Existe un proceso automático en el appliance que permite poblar el repositorio local de usuarios mediante una sincronización selectiva y periódica de los usuarios de un directorio corporativo compatible LDAP.

Por tanto, visitando cada uno de los casos, tenemos lo siguiente:
Alta de línea: Es una operación en el appliance mediante su API que se produce de forma inmediata.
Alta de usuario: El administrador no da realmente de alta usuarios, sino que a un usuario del directorio corporativo LDAP[31] le añade ciertos atributos para que el sistema de Telefonía IP entienda que es un usuario del sistema. Es una operación inmediata.
Sin embargo, para el Call Manager es una operación asíncrona. Lleva los usuarios LDAP a su repositorio interno de forma periódica.
Asociación Usuario-Línea: El alta de línea se debe hacer previamente y la del usuario también. Se depende de la replicación interna LDAP para que estando ya presente el usuario en el repositorio interno del Call Manager podamos asignarlo a la línea de forma inmediata con la API del Call Manager.

A la hora de introducir estos datos en ArchE, tenemos la siguiente lista de funciones: Alta Usuario, Alta Línea y Asociación Línea-Usuario, que coinciden con la funcionalidad del sistema. A la hora de asignar responsabilidades, ArchE nos crea responsabilidades con el mismo nombre, pero nosotros deberemos crear dos más, conocedores de que el alta de usuarios tiene una parte LDAP y otra de sincronización en el Call Manager. También debemos asignarles valores a los atributos de modificabilidad, que supone establecer el tiempo en días que permite realizar modificaciones en dicha responsabilidad.

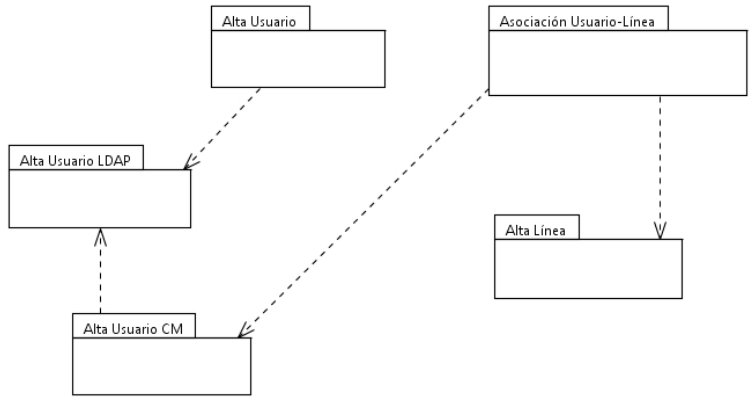


Imagen 20 - Módulos del sistema antes de aplicar las tácticas

En la imagen 20 tenemos el conjunto de módulos del sistema. En la Imagen 21 vemos cómo hemos creado las relaciones ente las responsabilidades. Se ha creado un escenario de modificabilidad que implica la posibilidad de cambiar la replicación LDAP que realiza el Call Manager (Alta Usuario CM). En el escenario están implicadas todas las responsabilidades salvo Alta Línea.

[30] Es una característica de la Telefonía IP. No necesitamos grabar en nuestro dispositivo una asociación Persona – Número, porque nos la proporciona el sistema si el administrador la ha realizado previamente.
[31] Ver en acrónimos

Se observa que ArchE nos plantea hasta 3 posibilidades diferentes.

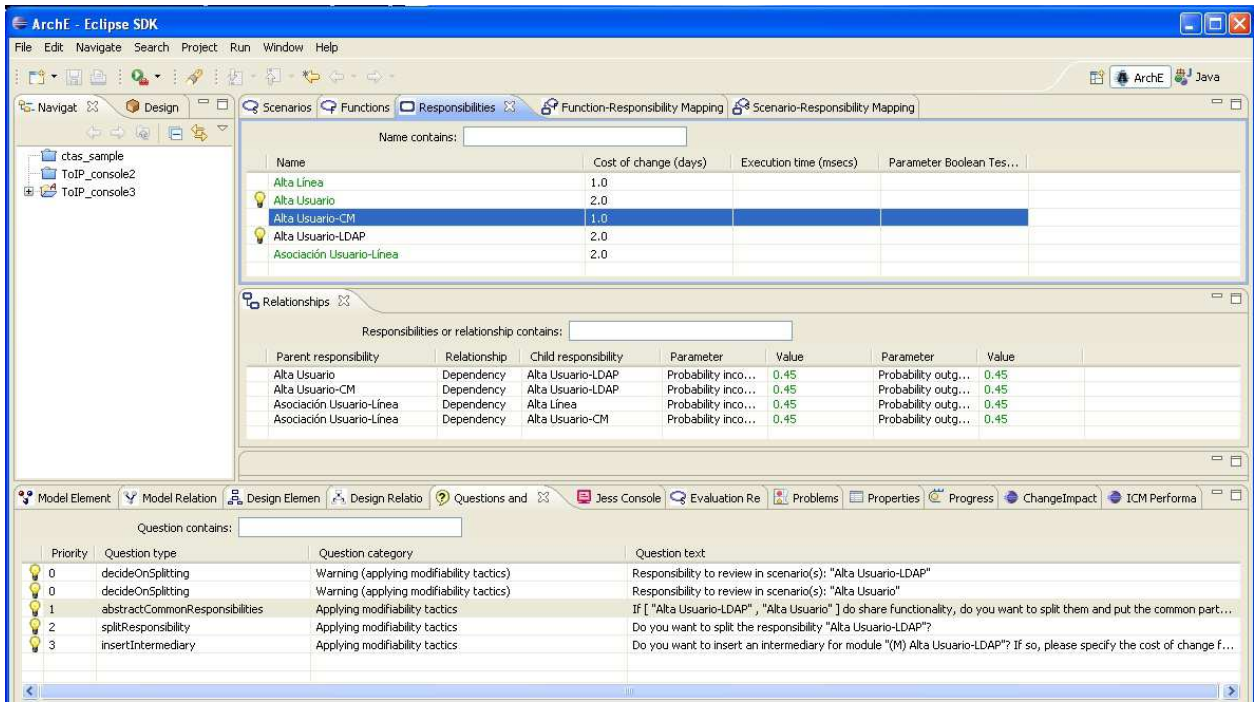
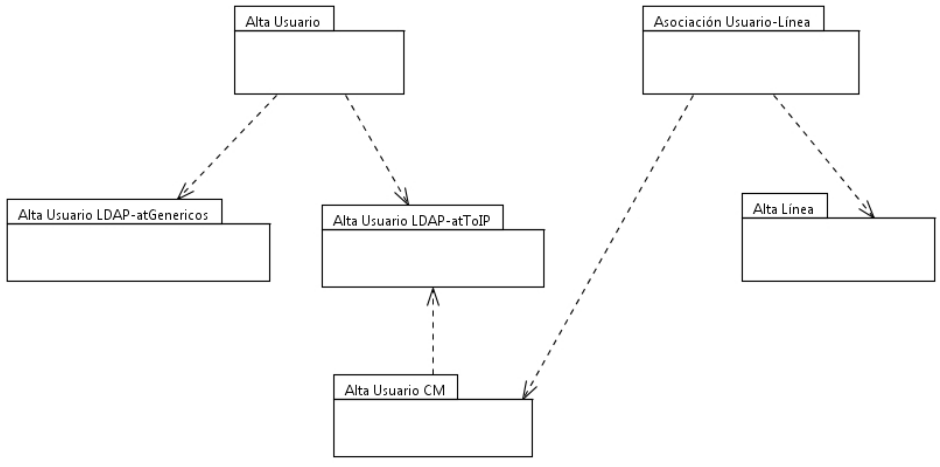


Imagen 21 - Sugerencias para aplicar tácticas ante el nuevo escenario de modificabilidad

Por un lado se ofrece en (1) agrupar funcionalidad de Alta Usuario-LDAP y Alta Usuario, separando y uniendo la funcionalidad común. No es aplicable porque en realidad una es un formulario web y la otra una API LDAP. En (3) se nos ofrece colocar un intermediario para Alta Usuario-LDAP, táctica que no tuvo mucho éxito. La táctica (2) de separar la responsabilidad de Alta Usuario-LDAP se le puede dar sentido si sirve para crear una responsabilidad de modificación de atributos genéricos LDAP por una lado y de atributos LDAP específicos para Telefonía IP. Debemos de ser nosotros quienes pongamos nombres con sentido, porque ArchE simplemente añadirá sufijos del tipo _child_XX con XX un número.

Sin producir una mejoría espectacular, la táctica (2) es una táctica que mejora nuestro escenario. Hay que decir que es muy complejo llegar a satisfacer los escenarios, sobre todo si la utilización de software COTS te impide utilizar alguna de las tácticas propuestas por ArchE, pero sí que son muy beneficiosas las reflexiónes que se realizan con la herramienta de cara a la mejora del sistema.



El reparto de responsabilidades en módulos, quedaría entonces de la siguiente manera como se muestra en la imagen 22.

Imagen 22- Módulos resultantes de aplicar la táctica

4. Retos en el uso de un asistente al diseño como ArchE

4.1 Aplicar varias tácticas juntas: problemas recursivos

En ocasiones el arquitecto software que esté empleando la herramienta ArchE para que le asista en un diseño, se puede sentir desesperado porque la aplicación de una Táctica o Patrón Arquitectónico mejora un escenario y tiene el efecto colateral de empeorar otro. Las sucesivas correcciones aplicadas mediante diferentes tácticas pueden ir mejorando paulatinamente la situación hasta que lleguemos a un punto en el que nos parezca que estamos en el punto de partida, no sólo porque la valoración del escenario que ArchE nos indique sea igual o peor que al principio, sino porque además tengamos para resolver una situación parecida a la de inicio. En el camino habremos introducido un buen número de nuevas responsabilidades y asociaciones entre ellas. Algunas de ellas entrarán a formar parte de los escenarios y otras no, pero tras todo el trabajo y reflexión realizados, parece que estamos mucho peor que al principio.

Esta situación está magníficamente descrita en [2] sin hacer referencia a ArchE, pero es sin duda la situación que se va a encontrar el diseñador tanto si trabaja sobre el papel, como si trabaja con ArchE. El ejemplo expuesto en el texto, imagina un sistema en el que el diseñador necesita cumplir un escenario de disponibilidad mediante una estrategia de detección de errores como ping/echo. Lo normal al aplicar esta estrategia es preguntarse cómo va a encajar la modificación (un escenario de modificabilidad), si esto puede suponer problemas de seguridad (¿resiste o previene el sistema un ataque ping flood?) y finalmente si la adición del ping/echo generará un problema de rendimiento.

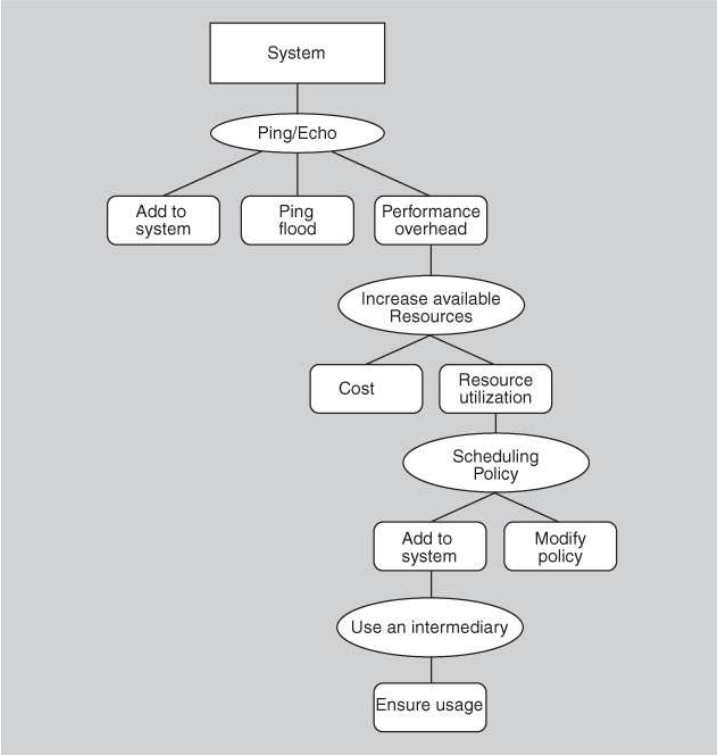


Imagen 23 - Ejemplo de uso de tácticas sucesivas que aparentemente acaban en un problema recursivo según [2]

En el supuesto de que abordar el problema de rendimiento sea lo más importante, se podría abordar mediante una táctica de rendimiento de incrementar los recursos disponibles (que a su vez genera escenarios de coste y reutilización de los recursos). Esta reutilización de recursos se puede abordar con una táctica de rendimiento denominada Scheduling Policy (una política de utilización de los recursos), que a su vez genera dos escenarios de modificabilidad (uno sobre cómo añadirla al sistema y otro sobre como modificar la propia política).

Se puede ver cómo acaba la historia en la imagen 23. Abordar el escenario de modificabilidad para añadir la Scheduling Policy mediante un intermediario que asegure que toda la comunicación pase por él y ser él quien realice el reparto de la carga, genera a su vez un escenario de disponibilidad: ¿Cómo

aseguramos el no fallo del intermediario? Cualquier arquitecto estaría temeroso de que alguien de su equipo le dijera ¿y si revisamos su disponibilidad mediante una táctica ping/echo?.....

Esta sería una de las pesadillas de un arquitecto en la fase de diseño. De ahí uno de los valores de emplear alguna herramienta que nos asista. En el caso de ArchE, aunque no lográramos resolver completamente el escenario, podríamos tener una valoración numérica de las diferentes posibilidades y volver a sentarnos con los interesados resolviendo los escenarios dentro del encaje de intereses de cada uno.

4.2 Sistemas COTS

En varias ocasiones los autores nos indican su preocupación por los sistemas y componentes Commercial Of-The-Shelf, como dedicando un capítulo específico en [1] o usando activamente en sus ejemplos en [2] este tipo de sistemas. Otros autores vinculados al SEI como Albert & Brownsword [13] también los han tratado en detalle. La utilización de componentes cambia el proceso de diseño e introduce restricciones en la arquitectura. Aunque se eligen componentes para obtener ciertas funcionalidades, estos componentes también plasman ciertas elecciones arquitectónicas y por tanto también de los atributos de calidad. Por tanto, el diseñador deberá prestar una especial atención a lo que suceda con los atributos de calidad en el diseño de su sistema.

Cuando decimos que los sistemas y componentes COTS cambian el proceso de diseño, nos referimos a que en [1] la integración de componentes COTS se concibe como un proceso de búsqueda, en la que se tratará de verificar cómo se comportan diferentes componentes candidatos del mercado frente a los atributos de calidad del sistema bajo diseño. La forma de discernir esto es creando modelos sobre los que trabajar y ArchE puede ser también una ayuda para esto.

En [13], conscientes de la dificultad de integración de sistemas COTS, se centran en el conjunto de prácticas de ingeniería que potencien y aseguren el uso de COTS en una organización. Denominaron su marco de trabajo EPIC (Evolutionary Process for Integrating COTS). Nos proporcionan una definición que creo muy ajustada a la realidad actual de los que es un sistema COTS

> *El término componentes, tal y como es usado en EPIC, incluye componentes hardware y software en el mercado comercial, sistemas legacy (una pieza a ser reemplazada), librerías, y cualquier otra fuente pensada para la reutilización (como freeware y shareware).*

Entre las características de **COTS** se incluyen ciertas **restricciones** impuestas por el mercado:

- Son componentes que se venden, alquilan o licencian al público/clientes en general, con un vendedor buscando un beneficio.
- El soporte lo proporciona el vendedor, que retiene la propiedad intelectual del producto.
- Se usa sin modificación del código del producto[32].

[32] Para aquellos que andan pensando que en la definición falta el software open source o de código abierto, esta es la característica clave que los excluye. En el software de código abierto podemos realizar las modificaciones necesarias e incluso evaluar el diseño desde el punto de vista que hemos explicado durante todo el texto. Esto nos puede animar a usarlo tal cual o tomar la libertad de realizar las modificaciones que aseguren atributos de calidad en la integración con nuestro sistema.

Por lo anteriormente dicho, en el SEI identifican a los componentes COTS por las siguientes características inherentes a esas restricciones de mercado:

- Es el mercado y no las necesidades de nuestro sistema, las que gobiernan el desarrollo y la evolución de un componente. Será normal que el mercado, y el componente con él, sufran cambios continuos, pero en todo caso la frecuencia y contexto de cada versión del componente COTS queda a discreción del vendedor.
- Existe en el mejor de los casos, una visibilidad limitada de los interiores del componente y su comportamiento. Lo que asume un componente COTS de los procesos puede no coincidir con las asunciones de una organización específica.
- Los componentes COTS tienen a menudo dependencias inesperadas con otros componentes COTS[33].

EPIC nos dice que el control de componentes críticos ha pasado de las manos del proyecto, a las manos del subministrador del componente. Cada vez más soluciones se diseñan empleando componentes y cada vez más componentes están a su vez hechos de componentes. Se requieren buenas prácticas de desarrollo en espiral o iterativas para descubrir, con una exploración evolutiva, los elementos de riesgo. Se necesita además un equilibrio entre administrar la obsolescencia de un componente y darle estabilidad al diseño, lo que obligará a los arquitectos a **monitorizar constantemente el mercado, durante el ciclo de vida del proyecto, para anticipar los cambios**.

Una arquitectura que retiene su estructura y cohesión, pero permite responder fácilmente a los cambios, es decir, que sea evolucionable, se convierte en un activo importante para una organización. Para ello hay que tener claro que diseñar con componentes no es lo mismo que diseñar para la reutilización. EPIC nos recomienda mantener actividades que monitoricen el cambio de los procesos internos en los que interviene el componente COTS y también documentar adecuadamente cómo el componente apoya la solución final. Estos procesos serán una mejor garantía para afrontar los cambios que realice el vendedor.

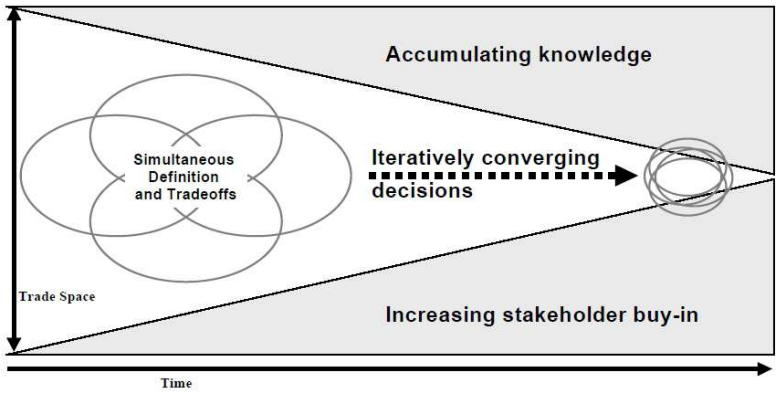


Imagen 24 - Convergiendo a una solución con EPIC

En el proceso de diseño de un nuevo sistema software en el que intervienen componentes COTS, EPIC propone un método iterativo que intenta basar la elección de un componente en el equilibrio entre cuatro esferas de influencia: los interesados en la construcción del sistema y los usuarios de los procesos, la propia arquitectura del

[33] Sin duda esto es una de las situaciones a la que más trabajo dedican hoy en día los especialistas en administración de sistemas operativos o aquellos dedicados a la gestión del software middleware. Afrontar la situación de un proyecto que tiene ciertos requerimientos funcionales que no se pueden obtener con la utilización de un componente COTS en la versión instalada, pero que sí se pueden obtener con una versión posterior, termina por lo general en una reunión de coordinación en la que todos los especialistas implicados han estudiado la "matriz de certificación de su producto". Esta matriz de certificación de un fabricante nos dice por lo general el contexto de instalación de una versión del producto junto con la lista de versiones de productos con las que es interoperable. Los cambios aceptables en un sistema con varios componentes COTS son aquellos para los que las diversas matrices pueden llegar a una solución. Las organizaciones que realicen implementaciones fuera de dicha matriz, aun cuando funcionen para el uso que le estén dando, pierden automáticamente el soporte contratado con el vendedor.

nuevo sistema, las ofertas de componentes COTS del mercado y finalmente los análisis de riesgos para el proyecto. Conforme se va construyendo iterativamente el necesario conocimiento, esas cuatro esferas van coincidiendo y se incrementa la capacidad de compra, como se visualiza en la imagen 24.

Estos procesos iterativos de construcción del conocimiento son requerimiento en todas las fases del ciclo de vida del proyecto hasta que este sea retirado. En EPIC van más allá que en [1] y no sólo defienden la necesidad de un modelo para esa construcción del conocimiento que he comentado, sino que además defienden que debe ser un **modelo ejecutable**. Realmente EPIC es una expansión o refinamiento del método RUP en la que en un principio se evalúan muchas soluciones para los componentes COTS candidatos, pero que en todo caso necesita como elemento esencial la posibilidad de experimentar y evaluar los cambios en los componentes con un modelo.

En cualquier caso, con modelo ejecutable o no, mi propia conclusión es que **una herramienta como ArchE puede ser de extraordinario interés para evaluar escenarios de atributos de calidad en los que estén implicados modelos de componentes COTS**. No en balde el propio Len Bass explica en sus seminarios para educadores [14] que **el marco de razonamiento de ArchE es apropiado para predecir el comportamiento antes de que un sistema se construya, para tomar decisiones durante la construcción, pero también para entender el comportamiento de un sistema una vez que ya se ha construido**. Precisamente esto último es lo que necesitamos entender con un componente COTS.

5. Aplicación de ejemplo: Sistema de Gestión de Telefonía IP (SGTIP)

5.1 Contexto social de una aplicación y su influencia sobre la arquitectura final del sistema.

Presento en este capítulo todos los aspectos relacionados con el análisis de una solución integral de Telefonía sobre IP. El contexto real de la aplicación es el de una institución universitaria que acomete los planes de modernizar su infraestructura de telefonía, basada en una centralita de tecnología convencional sobre par de cobre, por una basada en comunicaciones sobre protocolo de comunicaciones IP, la conocida Telephony over IP (ToIP).

Lo normal en esta situación es que la institución elabore un pliego de condiciones para un concurso público en el que se buscará la ayuda del sector privado para la implementación de una solución. Previamente a la elaboración del concurso público, se debe realizar un conjunto de actividades propias del ciclo de vida del desarrollo de un sistema software, pero que en este caso no están dirigidas a la implementación directa por parte de la institución o por parte de un colaborador/empresa concreto, sino que suponen una guía general de requerimientos funcionales, requerimientos de calidad y requerimientos de negocio a seguir por el ganador del concurso.

Los concursos públicos están regulados por Real Decreto Legislativo 3/2011, de 14 de noviembre, por el que se aprueba el texto refundido de la Ley de Contratos del Sector Público[34], que establece la imposibilidad de la adjudicación directa cuando los concursos superan una determinada cuantía. Por tanto no existe la posibilidad de contratar exactamente lo que los arquitectos software y técnicos implicados en la elaboración de los pliegos del concurso han identificado como más adecuado para satisfacer los objetivos de la institución. Obliga además ésta legislación a establecer un criterio de evaluación de ofertas que valore tanto la solución técnica como la propuesta económica. Dicha comisión de evaluación debe ser independiente del conjunto de profesionales que hayan redactado los pliegos.

La legislación también pone trabas del lado de las empresas que se presentan a estos concursos. El R.D. 3/2011 es garantista respecto de los intereses de la administración y exige a las empresas avales y la obligatoriedad de estar registradas como contratistas para el sector público. Esto hace que en la práctica muchas empresas de tamaño medio o pequeño no puedan acudir solas a los concursos, sino que lo hacen de la mano de multinacionales que sí han cumplido las exigencias de los registros y que suelen tener provisiones presupuestarias para los avales. Por tanto la subcontratación en este tipo de proyectos no viene sólo de la normal dinámica empresarial, sino que se ha visto incentivada por determinados aspectos de la legislación.

Lo descrito es un claro ejemplo real de contexto social que va a influenciar la arquitectura del sistema software que se busca implantar, tal y como se describe en [1]. Las actividades del ciclo de vida del software desarrolladas para poder redactar un pliego de concurso han dejado muy claro a los arquitectos de la Institución Universitaria que todas las soluciones del mercado que cumplen sus requisitos están basadas en software comercial, y que cualquier solución aceptable pasa por la

[34] Real Decreto Legislativo 3/2011, de 14 de noviembre, por el que se aprueba el texto refundido de la Ley de Contratos del Sector Público. Disponible online: http://www.boe.es/boe/dias/2011/11/16/pdfs/BOE-A-2011-17887.pdf

integración de 2 ó 3 componentes COTS. Por tanto, es muy importante que estos pliegos no se centren exclusivamente en la descripción de requerimientos funcionales, sino que detallen en todo lo posible los atributos de calidad buscados en el nuevo sistema, de manera que éstos formen parte de los criterios técnicos objetivos que contribuyan a la decisión por una oferta.

5.2 Objetivos del proyecto y restricciones impuestas por componentes COTS

Una solución de Telefonía IP puede consistir únicamente en un dispositivo hardware/software que hace el papel de una centralita, pero eso aporta poco de lo que esperamos sea una solución de telefonía con mejores prestaciones. **Los objetivos del proyecto consistían en obtener una solución de telefonía con mayores prestaciones y sin coste alguno para las llamadas internas, gracias a la utilización de la red de datos de la institución**. Por tanto en el proyecto se centró en la obtención de los siguientes elementos:

1. Una solución de gestión de líneas y dispositivos para Telefonía IP, lo que se denomina una centralita o Call Manager. Este módulo puede proporcionar características avanzadas como identificación de número y persona llamante, movilidad de extensión (puedo usar mi número de teléfono desde un dispositivo que no es el mío), o la utilización de dispositivos software, etc...
2. Una solución de Mensajería Unificada, que aporta las soluciones tradicionales de buzón de voz, pero que añade enrutamiento de mensajes, como recibir un fax a tu número de teléfono que es reenviado como email a tu dirección de correo institucional, enviar un SMS desde tu email, etc…
3. Un directorio telefónico accesible desde los propios dispositivos.
4. La integración de todo lo anterior con la solución de Gestión de Identidades que ya posee la institución. Esta Gestión de Identidades aprovisiona un Directorio Corporativo LDAP
5. Una solución de gestión unificada de la mayoría de actividades de operativa diaria con el sistema, que llamaré en adelante SGTIP.
6. Una solución de tarificación, que permita comprobar el uso del sistema, obtener datos de explotación y corroborar la facturación que realice la empresa adjudicataria del servicio de llamadas externas.

Los elementos (1) Call Manager y (2) Mensajería Unificada, son habitualmente sistemas COTS a elegir entre las soluciones del mercado. (3) Es un requisito que se puede obtener a partir de la correcta integración del Call Manager con el Directorio Corporativo LDAP. (4) Es un sistema COTS preexistente que da servicios de autenticación y autorización al Portal Institucional, correo electrónico y red WiFi de la Universidad, así como a una variedad de otros sistemas que requieren autenticación. (5) Será una solución a medida para la gestión que tendrá que desarrollar el ganador del concurso. Finalmente (6) es una solución independiente de tarificación que no es objeto de estudio en el presente trabajo.

En buena medida, el éxito del proyecto está en la correcta implementación de un sistema software (5) completamente nuevo, que tiene que satisfacer las necesidades de gestión diarias que implican líneas, dispositivos, mensajería, usuarios y la relación entre ellos, pero que tiene restricciones importantes debido a la utilización de varios sistemas COTS. Dentro del ciclo de vida del sistema, las tareas a realizar en los sistemas COTS se centran en actividades de configuración, mientras que las tareas a realizar para la solución de gestión SGTIP deberán comenzar por el análisis y toma de requisitos. Es la solución SGTIP la que va a centrar nuestra discusión posterior y análisis de su arquitectura.

5.3 Requerimientos funcionales y de calidad del sistema SGTIP

A continuación damos un detalle de todos los requerimientos funcionales y de calidad de SGTIP[35]:

Tabla 13 – Requisitos de SGTIP para el pliego de un hipotético concurso público

SGTIP debe ser capaz de dar de **alta en el sistema dispositivos, líneas y usuarios**. Como quiera que los dispositivos siempre van asociados a una línea y que los usuarios también, deberá ser capaz de realizar dichas asociaciones. Toda esta funcionalidad la debe proporcionar tanto de forma separada, como de forma coordinada atendiendo a **requisitos de usabilidad**. Estos requisitos de usabilidad vendrán determinados por el flujo de trabajo normal de un operario de telefonía, para el que lo habitual es dar de alta un dispositivo y a continuación tener la oportunidad de dar de alta una línea asociada, finalizando con dar de alta a un usuario asociado a dicha línea. El cumplimiento de los requisitos de usabilidad evitará errores y facilitará la comprensión de la operativa del sistema.

El sistema SGTIP deberá ser capaz de implementar las operaciones de **baja de dispositivos, líneas y usuarios**, teniendo en cuenta que alguna de estas operativas de baja es susceptible de romper asociaciones previamente establecidas. Dichas asociaciones deberán por tanto ser actualizadas o dadas de baja.

Los **usuarios y líneas podrán modificarse después de haber sido dados de alta**, de manera que los permisos de llamadas, las características de la mensajería asignada y demás atributos de la línea o el usuario puedan ser modificados a posteriori. No existe una operación de modificación de los dispositivos, ya que éstos se modifican globalmente con plantillas de características por modelo que proporciona el Call Manager.

Cuando un usuario pierde su relación con la institución, su teléfono permanece, por lo tanto se considera interesante, **como requisito de usabilidad, añadir una operación que permita robar la línea de un usuario para asignársela a otro**. Operación que podrá ser empleada en estas situaciones o cualquier otra que considere el administrador.

Para mejorar la usabilidad y evitar errores, se requiere facilitar una opción de búsqueda del usuario en el Directorio Corporativo LDAP, para iniciar con el resultado de la búsqueda cualquiera de las operaciones de alta, baja o modificación del usuario. Esto evitará errores en la introducción de datos personales que ya existen en el sistema LDAP y que son proporcionados y mantenidos por el Sistema de Gestión de Identidades.
La necesidad de **integración con el Directorio Corporativo LDAP para la obtención de un directorio telefónico**, que se ha detallado en parte en el punto anterior, requiere que los sistemas COTS que lo utilicen para su operativa (Call Manager y Sistema de Mensajería Unificada) proporcionen algún método para no tener que crear dichos usuarios en sus repositorios internos de forma manual, sino que esto se realice mediante procedimientos automáticos a partir de los datos del

[35] El proyecto real en el que está basado SGTIP en realidad añade la complejidad de tener además de usuarios, lo que se denominó una sala, es decir, la posibilidad de teléfonos sin usuario asignado, pero asignados a un uso compartido o de carácter funcional. Los típicos ejemplos son los teléfonos de salas de reuniones, laboratorios, etc… Debido a que aporta mayor complejidad y no trae beneficios a nivel conceptual para la temática del trabajo, he decidido no incluirlos.

Directorio Corporativo LDAP.

Debido a la dependencia con el sistema LDAP, se deben detallar propuestas específicas para asegurar la disponibilidad del sistema SGTIP en horario laboral.

Podemos decir que el relato que he dado de los requerimientos, serían los requisitos a publicar en un concurso. Una vez resuelto el concurso, nos podemos encontrar con una **lista de restricciones que imponen los sistemas COTS** que la empresa ganadora del concurso ha propuesto para la solución. Podría ser una lista como la siguiente:

- El subsistema de Mensajería Unificada debe emplear un repositorio interno de usuarios de mensajería. La única forma de integrarlo con el Directorio LDAP es que sincronice periódicamente con él. Se realizará una importación de usuarios LDAP a su repositorio local. Los usuarios a importar son sólo aquellos marcados en el LDAP con ciertos atributos específicos del sistema de mensajería (no son reutilizables otros atributos presentes en el Directorio LDAP).
- El subsistema Call Manager también necesita un repositorio interno de usuarios, que son los únicos asociables con las líneas. Aunque el Call Manager proporciona una amplia API en forma de servicios web para realizar las operativas con dispositivos, líneas y usuarios, la única forma de integrarse con el Directorio LDAP es cargando ficheros Excel con los usuarios o configurando una sincronización periódica contra el Directorio LDAP. En este último caso serán cargados en el repositorio interno del Call Manager sólo aquellos usuarios marcados con atributos específicos para el Call Manager (no son reutilizables atributos ya presentes en el Directorio LDAP. Los atributos además no coinciden con los que necesita la Mensajería Unificada).

No se listan restricciones impuestas por el Subsistema Directorio Corporativo LDAP, ya que son previamente conocidas.

5.4 Casos de uso

Los casos de uso, que obtenemos en SGTIP son los que se pueden ver en la imagen 25. Se puede observar un único actor, el administrador del sistema de Telefonía IP que interactúa con todos los casos. También se observan las relaciones entre diversos casos, fruto en su mayoría de los requerimientos de un atributo de calidad como es la usabilidad, ya que varios casos de uso se podrán emplear en solitario o podrán ser el primer paso de un procedimiento más complejo. Ante esta situación habrá que implementar las tácticas adecuadas para satisfacer los requerimientos.

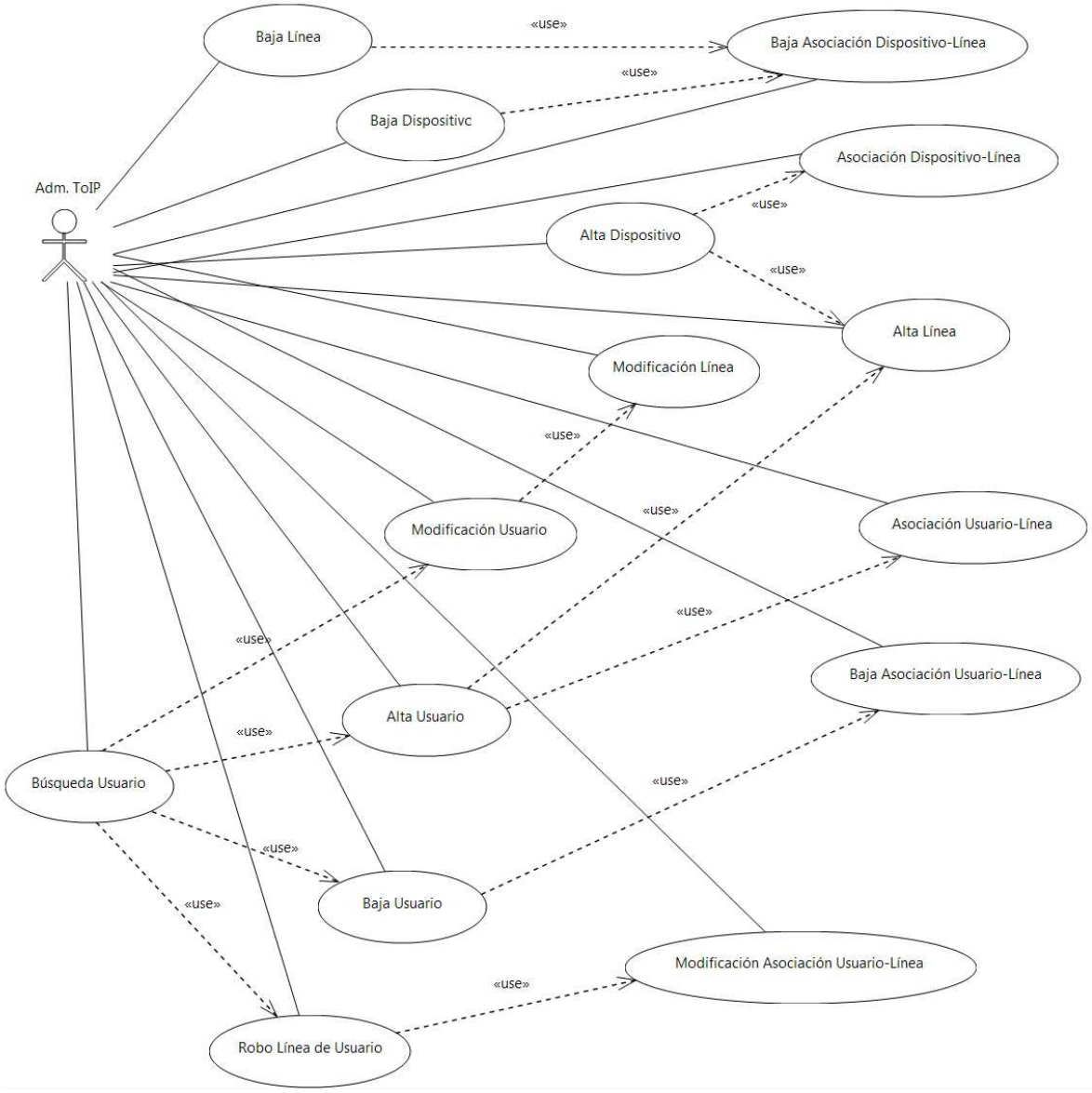


Imagen 25 - Casos de uso en SGTIP

5.5 Vistas más importantes de la arquitectura propuesta para SGTIP

Se consideran como más importantes las vistas de despliegue (imagen 27) y de módulos (imagen 26), junto con un diagrama de actividad (imagen 28) que resalta el carácter asíncrono de las operaciones de alta de usuarios. El diagrama de actividad de la imagen 28 es el único diagrama que presento tomado de la documentación de implantación del caso real en que está basado este ejemplo de SGTIP. Por eso se observará referencias a salas y pre-condiciones y post-condiciones correspondientes a la prueba. El equivalente del Call Manager en dicho diagrama es CUCM y del Sistema de Mensajería es MRS, y representan los productos reales o appliances en los que se basó la implementación.

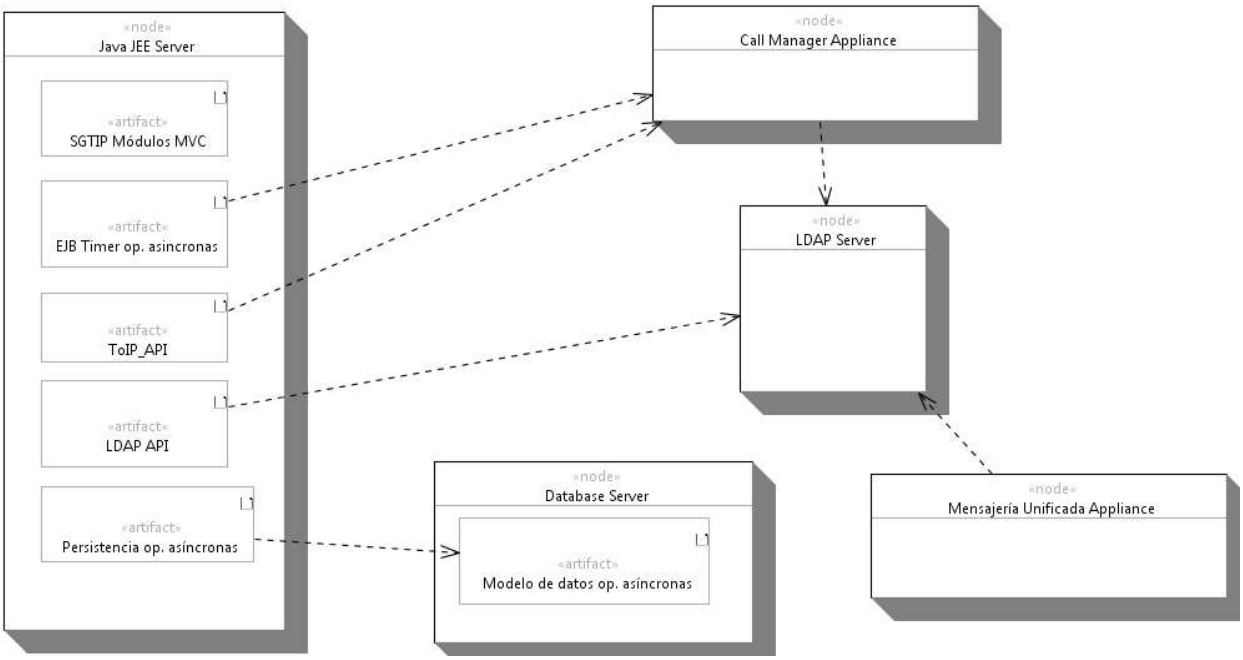


Imagen 27 - Diagrama de despliegue de SGTIP

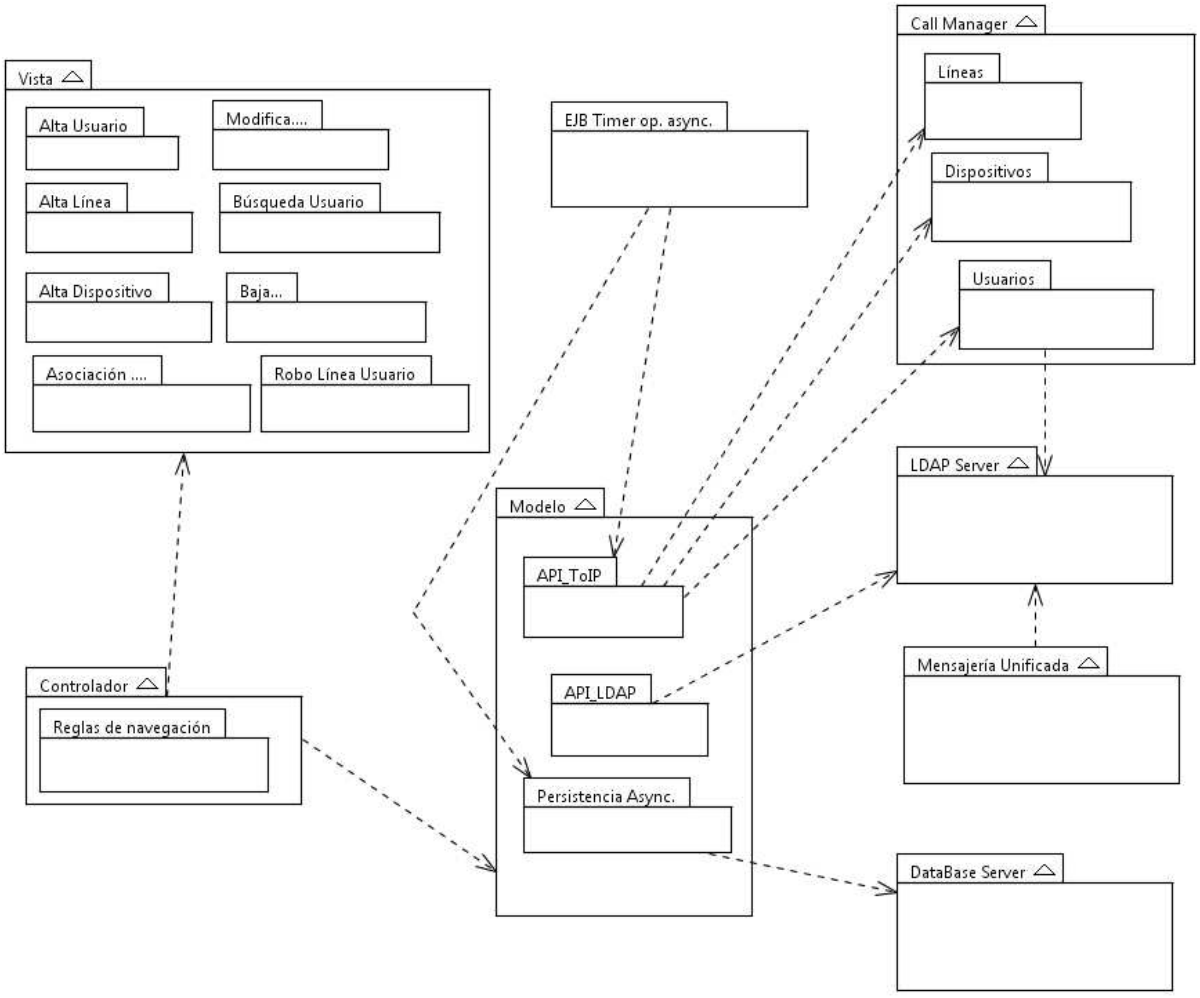


Imagen 26 - Diagrama de Módulos de SGTIP

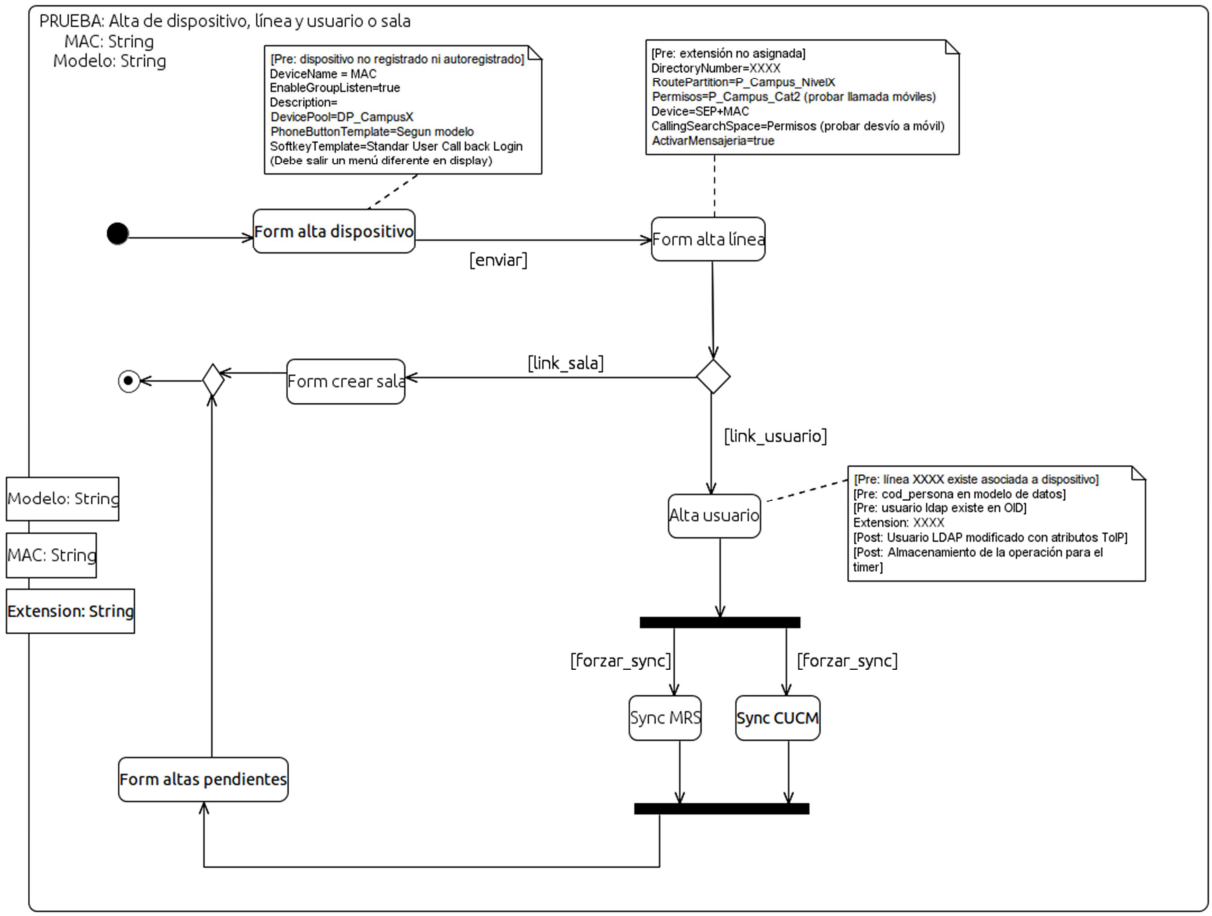


Imagen 28 - Diagrama de secuencia para un alta consecutiva de un dispositivo, línea y usuario asociados. Corresponde a la documentación de casos de prueba original de SGTIP

En el diagrama de módulos se han resumido varios paquetes con una temática común en uno único para ganar claridad en la exposición del diagrama.

Se puede observar que el subsistema de Mensajería Unificada no supone gran problema dentro de la arquitectura propuesta. El caso de uso de Alta de Usuario (modificación/baja), en el que tendríamos que incluir los servicios de mensajería del usuario como una propiedad LDAP del mismo, consiste únicamente en rellenar esos atributos por parte del módulo API_LDAP en el Servidor LDAP. El sistema de mensajería actuará de forma asíncrona y periódicamente creará/actualizará usuarios con las propiedades de mensajería deseadas.

Para comprender mejor el funcionamiento del Call Manager **se emplea un modelo conceptual de lo que tiene en su interior este subsistema COTS**. Realmente podemos considerar que está subdividido en tres módulos, de los cuales dos de ellos, Líneas y Dispositivos tienen un comportamiento muy diferente al otro, Usuarios. Líneas y Dispositivos son servicios web completos con capacidad de altas/bajas y modificaciones, mientras que Usuarios es accesible como servicio web sólo para la obtención de un registro de usuario. El módulo tiene un comportamiento parecido al del appliance de mensajería, es decir, importa los usuarios LDAP según una configuración dada. Esto quiere decir que las operaciónes de líneas y dispositivos son operaciones síncronas en las que se emplea el módulo API_ToIP atacando los servicios web del Call Manager, pero un alta de usuario es asíncrona y periódica.

Lo peor del modelo del Call Manager es que al ser el alta de usuarios asíncrona, esto convierte la asociación de Usuario-Línea también en asíncrona. Este es el motivo de introducir el módulo "EJB Timer op. async.". Una operación de asociación de Usuario-Línea se realizará si está ya disponible el usuario, pero de no estarlo, se almacenarán los datos de la operación en una base de datos y el módulo EJB Timer (ver sección de acrónimos), preparado para despertar periódicamente, recuperará las operaciones pendientes de asociación Usuario-Línea para ejecutarlas.

5.6 Estudio de escenarios con ArchE: Sustitución del servicio LDAP en SGTIP

El estudio en detalle de la arquitectura del sistema demuestra que existen muchas dependencias del Servidor LDAP, que para más riesgo, es uno de los sistemas COTS implicados en la arquitectura final. Este escenario podemos considerarlo relacionado con uno de los requisitos del sistema, mejorar la disponibilidad, altamente dependiente del Servidor LDAP. Con este tipo de servicios de red es posible aplicar técnicas de disponibilidad como incrementar los recursos disponibles (configurar el servicio en cluster, de manera que tengamos dos servidores), táctica a su vez que necesitará otra que asegure el reparto de la carga (mediante un round-robin DNS, o mediante un appliance especializado en el balanceo de servicios de red).

El resultado del empleo de las tácticas anteriores puede abrir un escenario de modificabilidad. Quizás el servicio LDAP ya no pueda funcionar en el mismo puerto, quizás necesitemos cambiar el nombre de la máquina por el del balanceador o quizás simplemente el modelo de Servidor LDAP no admite una configuración en cluster y tengamos que sustituirlo por otro.

Vamos a ponernos en el peor caso, una sustitución completa del Servidor LDAP en un escenario en el que se ven afectados los drivers que emplea el módulo API_LDAP y la configuración de los appliances COTS que empleamos, para ver si podemos resolverlo en un coste de 8 días.

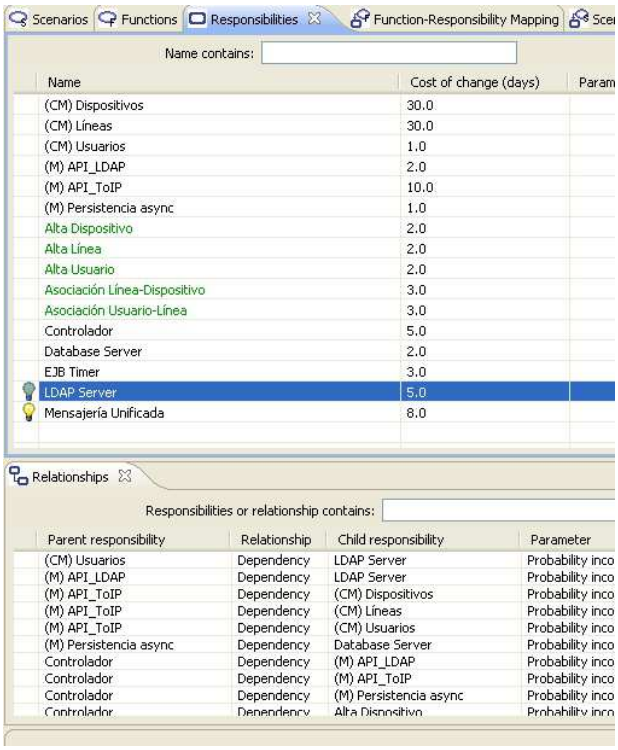


Imagen 29- Responsabilidades y sus relaciones en el escenario de sustitución del Servidor LDAP

Cuando establecemos el coste del cambio para cada responsabilidad, se ha hecho en base a la experiencia, procurando valores más bajos para elementos de la vista que suponen una operación atómica, un poco más alto para otros elementos de la vista con varias operaciones coordinadas. Igualmente otros elementos reciben un coste más bajo cuando se supone que en lo que se verán afectados es en una configuración (como el COTS que corresponde a la base de datos, por ejemplo, ver imagen 29).

Sin embargo, se ha pretendido emplear un coste muy elevado para aquellos elementos que sabemos que no podemos cambiar, como la API del Call Manager de Líneas y Dispositivos, pero la realidad es que **ArchE no permite introducir valores**

superiores a 30 días ni inferiores a 1, de manera que este es el valor máximo con el que se ha trabajado.

En la primera iteración, la herramienta propuso las siguientes opciones:

1. Separar funcionalidad común en Servidor LDAP y Mensajería Unificada para juntarlas en un mismo módulo: Se descarta al no ser aplicable a los dos sistemas COTS implicados.
2. Separar responsabilidades en Servidor LDAP: No es aplicable al sistema COTS.
3. Introducir un intermediario para el Servidor LDAP: Sería aplicable si pensamos que introducimos un balanceador.

Una de las conclusiones que sacamos trabajando con ArchE es que el modelo de coste no es muy claro. Si persistimos en pensar en situaciones reales, en días, es fácil ver empeorar nuestros escenarios de modificabilidad. Si pensamos el coste en términos comparativos de las diferentes tareas en base 30 (siendo esta la más compleja), las recomendaciones de ArchE empiezan a cobrar mucho más sentido (no olvidando hacerlo también para la definición del escenario). Empleando esta técnica, se ha llegado a alguna recomendación aplicable en las siguientes iteraciones, como la de separar el módulo API_LDAP, que puede tener sentido ya que tenemos operaciones LDAP dedicadas a atributos de mensajería y otras a atributos de telefonía (para el Call Manager). En otra iteración posterior, se aplicó un intermediario a API_ToIP. Con el nuevo esquema de costes y dos iteraciones más aplicadas tenemos un escenario que no se cumple, pero está muy cerca de hacerlo y con una mejora de los atributos de calidad en cuanto a modificabilidad y disponibilidad en el escenario planteado. En la imagen 30 se ve el diagrama de módulos obtenido.

5.7 Evaluación con ArchE de arquitecturas alternativas para SGTIP con diferentes sistemas COTS

En esta otra situación, nos imaginamos que evaluamos una arquitectura de partida diferente para SGTIP. En este caso, los appliances de Call Manager y Mensajería Unificada incluyen en su sistema de sincronización LDAP para poblar su repositorio interno de usuarios, de una API capaz de realizar la sincronización/alta de un usuario dado[36] bajo petición. Esto elimina la necesidad de operaciones asíncronas y simplifica la arquitectura. Ahora tendremos en el modelo de nuestro MVC una API_MensajeriaUnif que realizará esa petición al sistema de mensajería, mientras que el módulo que ya teníamos en la anterior arquitectura, API_ToIP también tendrá esta responsabilidad de solicitar sincronizaciones. Los diagramas de despliegue y diagrama de módulos se pueden ver en las imágenes 31 y 32 respectivamente.

Lo interesante de esta opción arquitectónica es que su comportamiento respecto al escenario de modificabilidad estudiado anteriormente con ArchE, es prácticamente idéntico en la primera iteración. Lógicamente un arquitecto optaría por esta opción, de ahí el valor del estudio de una opción arquitectónica mediante un asistente con ArchE, cuando le presentamos a la herramienta alguno de los elementos que modelan el sistema COTS, al menos desde el punto de vista del atributo de modificabilidad.

[36] Habría sido magnífico, sí. Sorprendentemente los appliances en la solución real no tenían tal cosa.

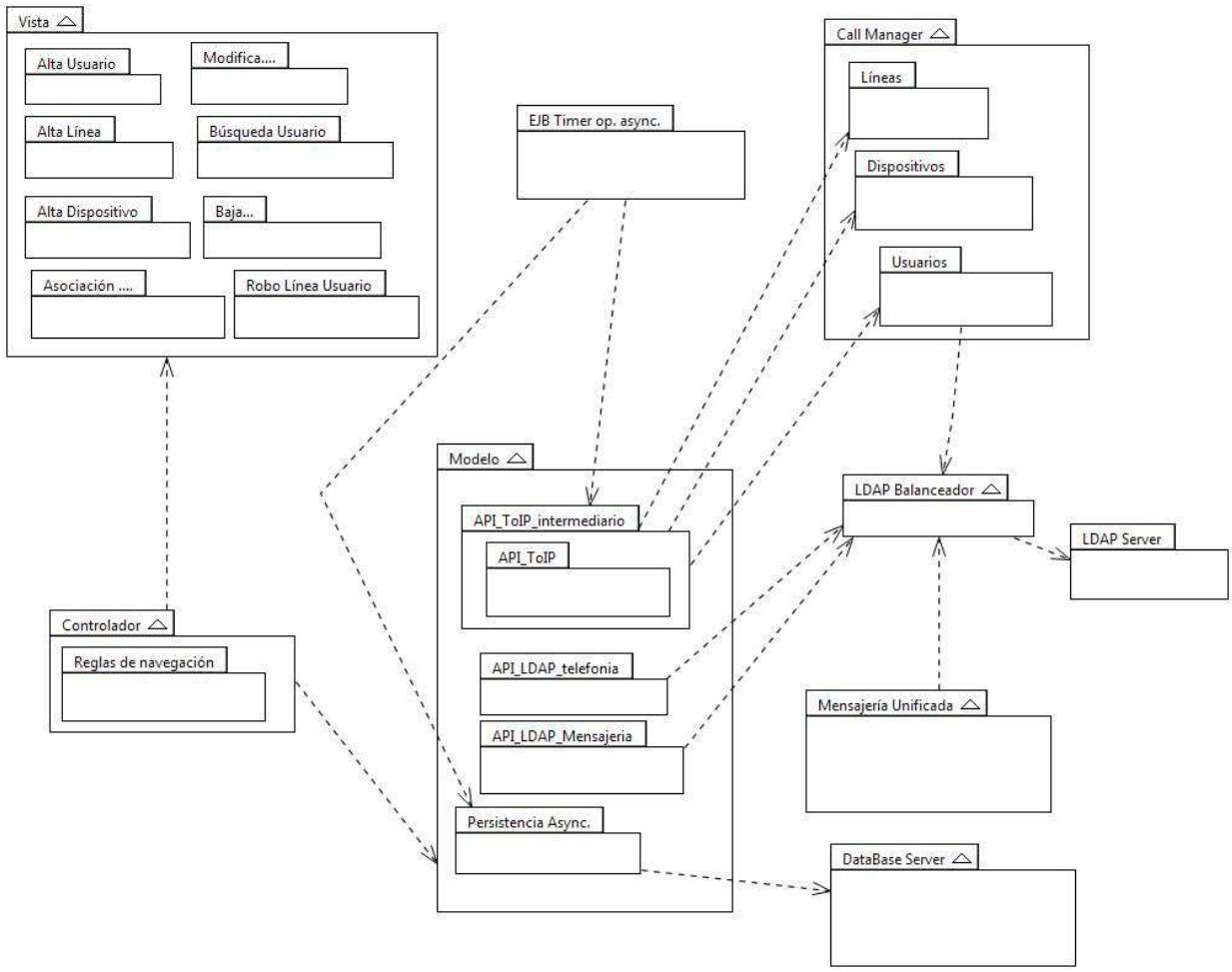


Imagen 30 - Diagrama de módulos resultante tras el análisis con ArchE

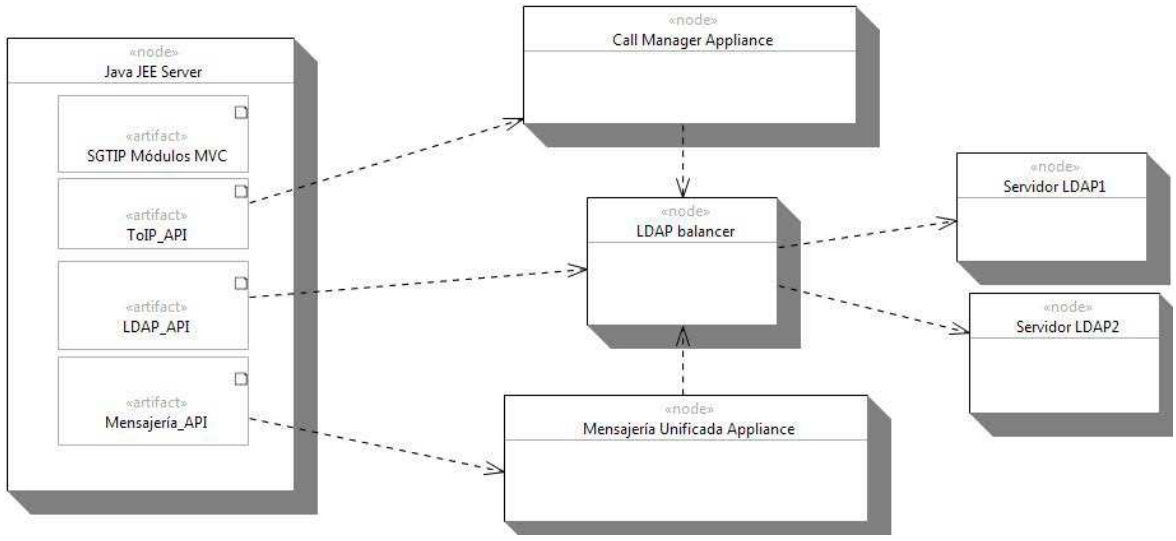


Imagen 31 - Arquitectura alternativa de SGTIP con diferentes appliances (Diagrama de despliegue)

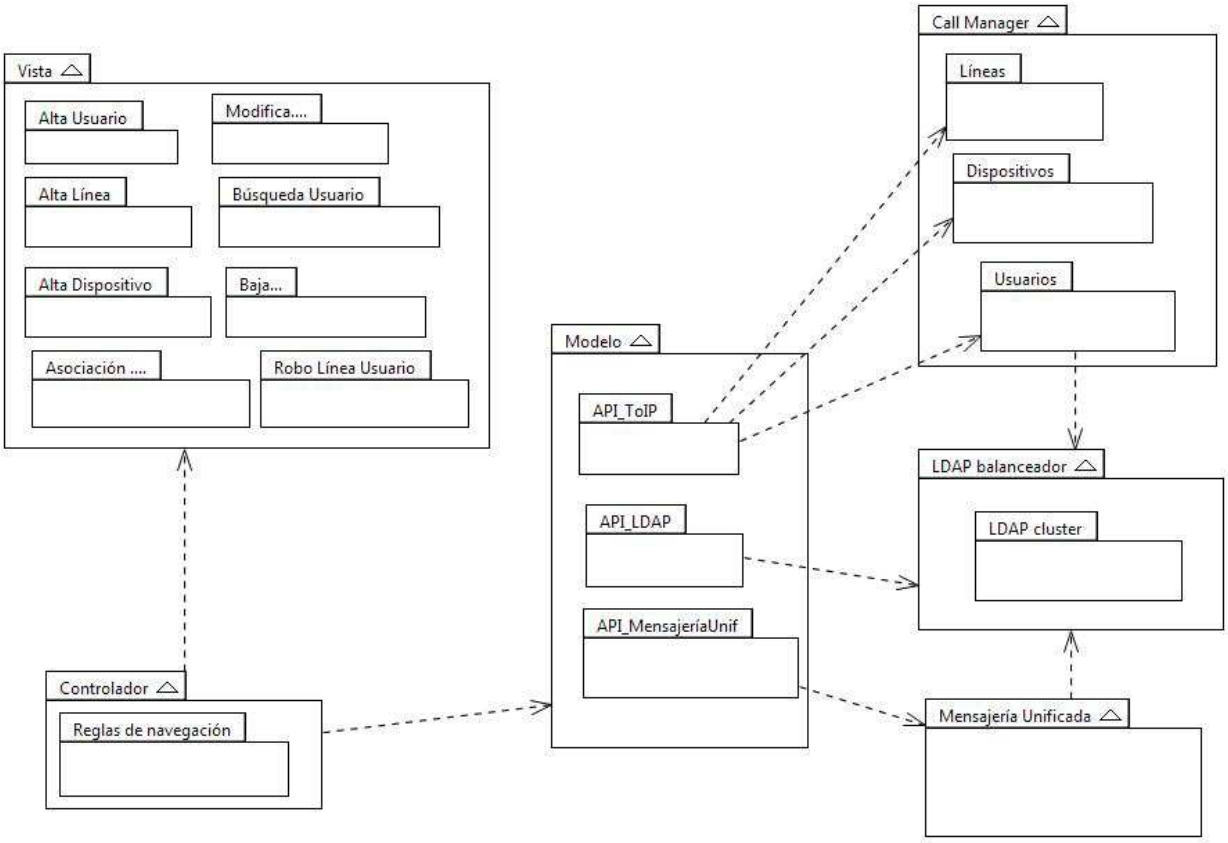


Imagen 32 - Arquitectura alternativa de SGTIP con diferentes appliances (Diagrama de módulos)

6 Conclusiones y líneas de trabajos futuros

6.1 Conclusiones sobre el marco teórico del presente trabajo

La importancia de los atributos de calidad en la construcción de un sistema software es algo que ya pocos autores ponen en duda. Si bien los autores de [1] han ido modulando alguna de sus ideas originales en la última década, se ha ido estableciendo en la comunidad de investigadores en ingeniería del software que el hecho de construir un sistema software sin prestar atención a los atributos de calidad puede terminar en un fracaso tan evidente como construirlo sin prestar la adecuada atención a los atributos funcionales. Los atributos de calidad determinan en buena medida la arquitectura resultante.

Attribute-Driven Design (ADD) se propone como el método iterativo que va a permitir emplear tácticas arquitectónicas a la resolución de escenarios concretos de calidad, de manera que el arquitecto pueda obtener una correcta asignación de responsabilidades a las estructuras resultantes. ADD es una aplicación del principio de que la asignación de funcionalidades del sistema a determinadas estructuras software, es lo que determina el soporte de una arquitectura para unos determinados atributos de calidad.

Hoy en día es muy común la utilización de sistemas COTS para aportar determinadas funcionalidades. No es raro el proyecto en el que pueden intervenir varios de ellos como en el ejemplo que hemos ido viendo a lo largo del presente texto. Los sistemas COTS añaden una preocupación al arquitecto software, que tendrá que realizar actividades específicas para el aseguramiento de la calidad, que pueden pasar por modelar estos sistemas y estudiarlos en el contexto de escenarios de calidad concretos, dentro del sistema en el que se van a integrar.

6.2 Conclusiones generales sobre la herramienta ArchE

El conocimiento de tácticas y patrones arquitectónicos para todos los atributos de calidad posibles, más los necesarios conocimientos y experiencia en la semántica de problemas concretos, se antoja un gran esfuerzo. Por eso los promotores de ArchE diseñaron la herramienta como un asistente al diseño, con vocación de sistema experto capaz de capturar esa experiencia. Lamentablemente la herramienta dispone únicamente de dos marcos de razonamiento para Rendimiento y Modificabilidad. También se echa de menos un modelo de coste para el marco de razonamiento de Modificabilidad que permita aproximaciones más precisas. Por desgracia, la herramienta ya no tiene actualizaciones y esto la puede poner en peligro de desaparición. Por ello me ha parecido interesante diseñar su instalación para que sea virtualizable, poniendo a disposición de los interesados en ella un appliance. Todo mi estudio de la herramienta se ha realizado con éxito mediante este sistema.

ArchE está muy bien alineado con los postulados de ADD y es buena ayuda para razonar sobre las estructuras resultantes de la aplicación de ADD. Aunque la herramienta no conoce la semántica del problema, está basada en responsabilidades, relaciones entre ellas y la relación de las responsabilidades con escenarios de calidad concretos. Todo esto se emplea para medir numéricamente cómo va evolucionando la estructura de módulos del sistema conforme aplicamos tácticas y reasignamos responsabilidades.

6.3 Conclusiones sobre la aplicación de ArchE al estudio de la arquitectura de ejemplo SGTIP

En el dominio de la Telefonía IP es conocida la importancia de integrar soluciones COTS para obtener conjuntos de funcionalidades complejas como las aportadas por un Call Manager, de manera que es prácticamente imposible encontrar implementaciones que no cuenten con este tipo de soluciones. El sistema de ejemplo propuesto, SGTIP, es un buen ejemplo de ello. Se han empleado subsistemas de SGTIP tanto para ilustrar los conceptos teóricos de Escenarios de Atributos de Calidad (2.3) o del proceso ADD (2.5), como para ilustrar el funcionamiento de la propia herramienta ArchE con un ejemplo concreto (3.7).

Sin embargo, dado que SGTIP está basado en un sistema real, ha resultado de interés dedicar el capítulo 5 del presente trabajo a poner a prueba la herramienta ArchE con dos problemas diferentes que se consideran de especial valor en este dominio.

El primero de los problemas propuestos en 5.6 trata de la sustitución de uno de los sistemas COTS. El Escenario de Modificabilidad generado es el que puede causar más impacto en el sistema, ya que se trata de la sustitución del servicio LDAP, que es el sistema que genera más dependencias en el diseño. La conclusión que sacamos del uso de la herramienta en este caso es que es muy difícil llegar a cumplir el Escenario de Calidad Concreto que le proponemos a la herramienta. Uno de los motivos por los que resulta difícil es porque constantemente nos vemos forzados a interpretar la coherencia de las propuestas de ArchE cuando estamos empleando un sistema COTS. Muchas de las propuestas dejan de tener sentido con estos sistemas. No obstante, esto no impide obtener mejorías en nuestra arquitectura en términos de Modificabilidad, tal y como numéricamente nos informa la herramienta, sino que además, atendiendo a otros criterios de calidad, somos capaces de interpretar que obtenemos grandes mejorías en Disponibilidad y Rendimiento.

Efectivamente, el resultado final del estudio de 5.6 no es un escenario de calidad cumplido, sino sólo uno mejorado. No obstante, la interpretación de la aplicación de determinadas tácticas, como la introducción de intermediarios, se ve claramente que aportan valor no sólo desde el punto de vista de mejora de la Modificabilidad, sino también desde la Disponibilidad y Rendimiento. En este caso interpretamos este intermediario como un sistema de balanceo hardware para el servicio LDAP[37], con la consiguiente mejoría de la arquitectura propuesta.

En cuanto al segundo de los problemas propuestos, en 5.7 nos planteamos una situación más propia de lo que en [1] se define como proceso de búsqueda asociado a la integración de sistemas COTS, o de lo que en EPIC [13] forma parte del proceso de construcción de conocimiento para la elección de un sistema COTS. Estamos proponiendo en el ejercicio planteado un modelo del sistema COTS (Call Manager) que permite eliminar las operaciones asíncronas. Esto da como resultado una arquitectura inicial más sencilla que la arquitectura de partida en 5.6 y que seguidamente sometemos al mismo Escenario de Modificabilidad con la sustitución del servicio LDAP.

[37] Este tipo de balanceadores hardware sirven no sólo para servicios LDAP sino para muchos otros, entre ellos HTTP. Entre las características que disparan el rendimiento cuando los usamos, está la posibilidad de realizar encriptación SSL por hardware de los servicios balanceados. Varios fabricantes como Alteon o F5 disponen de estas características. Por supuesto, estamos hablando de incluir otra solución COTS dentro del ejercicio planteado.

Las conclusiones que obtenemos son muy interesantes. ArchE nos permite evaluar los resultados desde el punto de vista de la Modificabilidad de la arquitectura y concluimos que en este caso tenemos una arquitectura muy superior a la propuesta en 5.6, porque somos capaces de obtener resultados numéricamente equivalentes a los de 5.6 en tan sólo una iteración con la herramienta. Hablamos en este caso de una arquitectura de mejor calidad que la obtenida en 5.6 no sólo por la Modificabilidad, sino porque interpretados los resultados bajo el prisma de otros atributos de calidad en los que vemos mejorías. Como bien se nos dice en [2], algunas de las tácticas que favorecen la Modificabilidad, también favorecen la Testabilidad. Emplear tácticas que favorecen la cohesión y separación de intereses o reducen el acoplamiento, típicas Tácticas de Modificabilidad, favorecen también las tácticas que intentan limitar la complejidad para favorecer la Testabilidad. La arquitectura resultante de 5.7 no es sólo mejor en términos de Modificabilidad, sino que siendo menos compleja, hemos favorecido además la Testabilidad.

Ya he dicho en las conclusiones generales que ArchE sólo tiene dos marcos de razonamiento, de manera que puede parecer un desperdicio no haber intentado añadir un Escenario de Rendimiento dentro del presente trabajo. El motivo de no haberlo hecho es por un lado que en el sistema real que ha inspirado SGTIP no se han detectado necesidades en esta área. Si bien el rendimiento es fundamental en un sistema de Telefonía IP, especialmente en el área estrictamente de transporte de los paquetes de voz, no lo es en absoluto cuando hablamos del área de gestión de dispositivos, líneas y usuarios, que es el cometido funcional de SGTIP.

Una justificación adicional para no haber incluido Escenarios de Rendimiento para SGTIP lo podemos encontrar también en un cierto cambio de actitud de los autores de [1] y [2] respecto del Rendimiento. En la segunda edición del texto base, este Atributo de Calidad aparece en pie de igualdad con el resto de Atributos de Calidad tratados en el texto, mientras que en la tercera edición se abre la discusión sobre el mismo con una mirada diferente. Esta nueva consideración del Rendimiento en [2] comienza por advertir de que ha sido el Rendimiento uno de los motores que en el pasado ha guiado la consecución de muchas arquitecturas, comprometiendo en ocasiones otros Atributos de Calidad. Sin embargo, la relación precio/rendimiento del hardware continua cayendo, mientras que la del desarrollo software sigue creciendo. Esto hace que su importancia relativa haya disminuido y que el empleo de Tácticas de Rendimiento, como el aumento de recursos, sean cada vez más frecuentes. Como bien dicen los autores en [2], actualmente el Rendimiento está muy unido a la Escalabilidad[38] y ésta, a su vez, es una forma de modificabilidad. Los diseñadores de sistemas COTS están poniendo especial cuidado en proporcionar Escalabilidad en sus sistemas. En mi opinión esta nueva mirada sobre el Rendimiento no hace sino aportar todavía más valor a los análisis que sobre la Modificabilidad podamos realizar con ArchE.

Finalmente, quiero comentar que mi interés por investigar Escenarios de Modificabilidad con ArchE se basa en la experiencia profesional, viendo la importancia que tiene en proyectos concretos. He podido ver corroborada esta impresión con el artículo de L. Bass y otros [15] en el que se detalla la recogida de más de 1000 Escenarios de Atributos de Calidad a lo largo de 8 años, pertenecientes a proyectos de diversos dominios. En los datos recogidos, la Modificabilidad encabeza las prioridades de los interesados en el proyecto, seguida de cerca por Rendimiento y Usabilidad y más de lejos por la

[38] Entendiendo la Escalabilidad como la capacidad del sistema de cambiar de una forma particular, aumentando recursos, permitiendo un funcionamiento normal del sistema y ganando en rendimiento.

Mantenibilidad (muy relacionada con la Modificabilidad igualmente). Si bien ArchE no posee numerosos marcos de razonamiento para todos los Atributos de Calidad que interesen en un proyecto, sí que tiene uno del que seguramente una gran mayoría pueden obtener beneficio, que es la Modificabilidad.

Del trabajo con la herramienta ArchE en el ejemplo SGTIP, se concluye que cuando estamos resolviendo un Escenario Concreto de Modificabilidad, muchas de las propuestas que nos realiza la herramienta podremos también interpretarlas en función de la tensión o favorecimiento que dichas propuestas generan en la Testabilidad, Disponibilidad, Interoperabilidad[39] y Usabilidad o incluso en el Rendimiento. Estos tres primeros Atributos de Calidad citados, han sido los más directamente relacionados con una implementación real de SGTIP y podremos guiar nuestras elecciones de propuestas de una forma realista. El objetivo será quizás no obtener un cumplimiento al 100% del escenario planteado, pero sí obtener una mejoría notable del mismo con una ganancia en otros Atributos de Calidad que sean de interés para el sistema en estudio.

Concluimos adicionalmente, que el empleo de modelos de sistemas COTS con la herramienta ArchE nos permite evaluar distintas opciones COTS que pudieran formar parte de una solución para la arquitectura de SGTIP. ArchE nos permite desarrollar actividades de construcción del conocimiento y de búsqueda de soluciones COTS que van a ser necesarias en SGTIP durante el ciclo de vida de dicha arquitectura. Se podrán afrontar por tanto estudios de impacto de las variaciones que van a sufrir los sistemas COTS en SGTIP (LDAP, Call Manager y Mensajería Unificada) para determinar la evolución futura del sistema. El ejercicio planteado en 5.7 da buena medida de los beneficios que podemos obtener en nuestra arquitectura al realizar este tipo de evaluación con ArchE.

6.4 Trabajos futuros

Los trabajos futuros que se plantean a la luz de todo lo estudiado los enfocaría en tres grandes líneas:

- Metodologías Agile
- Model Driven Architecture (MDA) y asistentes al diseño arquitectónico
- ADD 2.0 en asistentes al diseño arquitectónico

Leyendo el texto base que inspira los contenidos teóricos del presente trabajo, es inevitable encontrar algunas analogías entre las metodologías de tipo Agile y ADD. De entrada en ADD no tenemos un ciclo de vida en cascada clásico, sino uno iterativo, que puede incluir volver a renegociar requisitos funcionales y de calidad con los interesados, ya que el método reconoce que podemos empezar los pasos ADD antes de tener absolutamente todos los requisitos. También la voluntad de postponer los detalles de implementación lo más posible y dedicarnos a la arquitectura es un rasgo Agile. Son elementos centrales en Agile los descritos y casi parece que ADD se adhiere al principio Agile “changes are welcome”. En la tercera edición de su libro [2], los autores propugnan la compatibilidad ADD vs. Agile en un breve capítulo, pero ciertamente esto requiere de mayor desarrollo y también de la integración de una herramienta como ArchE en este esquema.

[39] Este atributo de calidad no es objeto de estudio en [1], pero sí en [2]. En la tercera edición del texto base se define como la habilidad de dos o más sistemas para intercambiar información vía interfaces en un contexto particular. Es una doble habilidad sintáctica y semántica. Obviamente es de mucho interés cuando hablamos de incorporar sistemas COTS a un diseño.

Otro de los aspectos interesantes de ampliación en un trabajo futuro estaría en la viabilidad de una herramienta de asistencia al diseño como ArchE, pero con un enfoque MDA. Está claro que con ArchE estamos generando un modelo de una de las vistas más importantes que definirán una arquitectura. Es inevitable pensar por tanto que el asistente al diseño podría generar un modelo del sistema para ayudar a su estudio y evaluación. Precisamente la elaboración de modelos ejecutables de sistemas COTS es lo que los promotores de EPIC propugnan como método de construcción del conocimiento. Una herramienta con esta doble orientación ADD+MDA sería de una utilidad enorme para el estudio de propuestas arquitectónicas, especialmente las que incluyan sistemas COTS. Autores como Gorton en [11] ya han mencionado la relación entre Atributos de Calidad como Portabilidad, Interoperabilidad y Reutilización y los postulados MDA que siguen algunas herramientas como AndroMDA, pero no he encontrado una herramienta que proponga una doble orientación ADD+MDA.

Finalmente también me ha resultado atractiva la propuesta en ADD 2.0 de clasificar los impulsos arquitectónicos en 3 niveles de interés para un interesado en el sistema. La pregunta está en si podríamos llevar este esquema a un asistente al diseño y si dicho esquema ayudaría a la resolución de escenarios quizás no de forma exacta, pero sí con márgenes de incertidumbre. Es una aproximación que podría ayudar a encontrar soluciones más rápidamente en los escenarios.

Referencias

[1] “Software Architecture in Practice, 2nd Edition”. Len Bass, Paul Clements y Rick Kazman. SEI Series in Software Engineering, Ed. Addison-Wesley 2003.

[2] “Software Architecture in Practice, 3rd Edition”. Len Bass, Paul Clements y Rick Kazman. SEI Series in Software Engineering, Ed. Addison-Wesley 2012.

[3] “Arquitecturas Software: Gestión de los atributos de calidad de un sistema y su incorporación en ArchE para la mejora del análisis, evaluación y soporte en la toma de decisiones durante el desarrollo arquitectónico”. Trabajo de investigación del alumno Jesús Martín Fernández, 2010. Escuela Técnica Superior de Ingeniería Informática Departamento: Ingeniería de Software y Sistemas Informáticos de la UNED.

[4] “An Introduction to Software Architecture”. David Garlan and Mary Shaw. School of Computer Science, Carnegie Mellon University. January 1994 (CMU-CS-94-166).

[5] “Quality Attribute Design Primitives and the Attribute Driven Design Method”. Len Bass, Mark Klein y Felix Bachmann. Software Engineering Institute, Carnegie Mellon University, 2001. Disponible online en:

http://www.sei.cmu.edu/library/assets/bilbao_paper.pdf

[6] “Attribute-Driven Design (ADD), Version 2.0”. Rob Wojcik, Felix Bachmann, Len Bass, Paul Clements, Paulo Merson, Robert Nord, Bill Wood. TECHNICAL REPORT 2006 (CMU/SEI-2006-TR-023). Software Engineering Institute, Carnegie Mellon University. Disponible online:

http://www.sei.cmu.edu/library/abstracts/reports/06tr023.cfm

[7] “A Practical Example of Applying Attribute-Driven Design (ADD) Version 2.0”. William G. Wood. TECHNICAL REPORT 2007 (CMU/SEI-2007-TR-005). Software Engineering Institute Carnegie Mellon University. Disponible online:

http://www.sei.cmu.edu/library/abstracts/reports/07tr005.cfm

[8] “ArchE Version 2.1 User’s Guide Version 1.0”. Software Engineering Institute, Carnegie Mellon University, 2007. Distribuido con la herramienta.

[9] “Using ArchE in the Classroom: One Experience”. Felix Bachmann, Len Bass, Philip Bianco, Mark Klein. TECHNICAL NOTE 2007 (CMU/SEI-2007-TN-001). Software Engineering Institute Carnegie Mellon University. Disponible online:

http://www.sei.cmu.edu/library/abstracts/reports/07tn001.cfm

[10] “Preliminary Design of ArchE: A Software Architecture Design Assistant”. Felix Bachmann, Len Bass y Mark Klein. TECHNICAL REPORT 2003 (CMU/SEI-2003-TR-021). Software Engineering Institute Carnegie Mellon University. Disponible online:

http://www.sei.cmu.edu/library/abstracts/reports/03tr021.cfm

[11] “Essential Software Architecture, 2nd Edition”. Ian Gorton. Ed. Springer, 2011.

[12] “Object Design: Roles, Responsibilities, and Collaborations”. Rebecca Wirfs-Brock, Alan McKean. Ed. Addison Wesley, 2003.

[13] “Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview. Key Elements in Building, Fielding, and Supporting Commercial-off-the-Shelf (COTS) Based Solutions”. Cecilia Albert, Lisa Brownsword. TECHNICAL REPORT (CMU/SEI-2002-TR-009) 2002. Software Engineering Institute Carnegie Mellon University. Disponible online:

http://www.sei.cmu.edu/library/abstracts/reports/02tr009.cfm

[14] “ArchE – An Architecture Design Assistant”. Len Bass. Presentación en SEI Software Architecture Workshop for Educators, 2007. Disponible online:

http://www.sei.cmu.edu/library/abstracts/presentations/arche07.cfm

[15] “Making Practical Use of Quality Attribute Information”. Ipek Ozkaya, Len Bass, Robert L. Nord y Raghvinder S. Sangwan. March/April 2008 IEEE Software.

[16] “Patterns of Enterprise Application Architecture”. Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee y Randy Stafford. Ed. Addison-Wesley, 2002.

[17] “Documenting Software Architectures. Views and Beyond. 2nd Edition”. Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord y Judith Stafford. SEI Series in Software Engineering. Ed. Addison-Wesley, 2011.

[18] “Enterprise Software Architecture and Design. Entities, Services and Resources”. Dominic Duggan. IEEE Computer Society, Ed. Wiley, 2012.

[19] “Software Architecture. Foundations, Theory and Practice”. Richard N. Taylor, Nenad Medvidovic, Eric M. Dashofy. Ed. Wiley, 2010.

[20] “Who needs an Architect?”. Martin Fowler. IEEE Software, July/August 2003.

[21] “Accelerating COTS Middleware Acquisition: The i-Mate Process”. Anna Liu, Ian Gorton. IEEE Software, March/April 2013.

[22] “Rigorous Evaluation of COTS Middleware Technology”. Anna Liu, Ian Gorton y Paul Brebner. Computer, IEEE Computer Society, March 2003.

[23] “A Principled Way to Use Frameworks in Architecture Design”. Humberto Cervantes, Perla Velasco-Elizondo y Rick Kazman. IEEE Software, March/April 2013.

[24] “Relating Business Goals to Architecturally Significant Requirements for Software Systems”. Paul Clements, Len Bass. SEI TECHNICAL NOTE (CMU/SEI-2010-TN-018). Software Engineering Institute Carnegie Mellon University, 2010. Disponible online:

http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=9347

Acrónimos

COTS: Commercial Off-The-Shelf

CUCM: Appliance de tipo Call Manager de una marca concreta (Cisco)

EJB Timer: Enterprise Java Bean (EJB) de tipo Timer. Forma parte del estándar Java Enterprise Edition y cumplen la función de realizar una actividad de forma periódica, al igual que un cron de sistema operativo, pero en el contexto de ejecución del contenedor EJB del servidor de aplicaciones Java.

MAC: Media Access Control o identificador único asignado a un interfaz de red en tecnologías como Ethernet. Típicamente son 6 grupos de 2 dígitos hexadecimales, como 00:05:9A:3C:7A:00

MRS: Appliance de Mensajería Unificada de una marca concreta (Sycos)

LDAP: (Lightweight Directory Access Protocol) lo forma un protocolo de acceso a datos en una base de datos jerárquica denominada directorio. El carácter jerárquico de la base de datos lo hace ideal para albergar estructuras organizativas empresariales, motivo por el cual suele ser el repositorio de usuarios, grupos y roles dentro de una organización. Actualmente proporcionan uno de los mecanismos más extendidos de dar servicio de autenticación y autorización de usuarios. Un registro LDAP contiene un grupo de atributos que habitualmente están organizados en grupos según su funcionalidad. Unos dan información personal (cuando el registro LDAP es de una persona), otros pueden dedicarse a configuraciones, etc…

ToIP: Telephony over Internet Protocol (IP)