UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

ITINERARIO DE INGENIERÍA DE SOFTWARE
Cod.31105128

TRABAJO DE INVESTIGACIÓN

"ARCHE: ARQUITECTURAS Y TOMA DE DECISIONES"

Estudiante: Rafael Elías Cárdenas Rodríguez

Director: José Félix Estívariz López

Curso 2013/2014. Convocatoria de defensa: septiembre

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

ITINERARIO DE INGENIERÍA DE SOFTWARE
Cod.31105128

"ARCHE: ARQUITECTURAS Y TOMA DE DECISIONES"

Trabajo Fin de Máster tipo A

Estudiante: Rafael Elías Cárdenas Rodríguez

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

Dentro del contexto de la ingeniería del software, el objetivo del diseño arquitectónico es derivar una representación arquitectónica de un sistema a partir del análisis y especificación de requisitos. Esta representación o modelo del sistema puede evaluarse respecto de la calidad antes de su construcción, lo cual permite al arquitecto verificar si la opción de diseño contemplada cumple con los requisitos de calidad. En caso contrario, el arquitecto refina la arquitectura aplicando tácticas arquitectónicas, puesto que los cambios que se hagan a posteriori suponen un mayor costo.

Los requisitos de atributos de calidad son satisfechos por las diversas estructuras diseñadas en la arquitectura, y el comportamiento e interacciones de sus elementos. Debido a la repercusión que conlleva una decisión de diseño, sobre la arquitectura de un sistema en el logro de los atributos cualitativos, resulta fundamental contar con métodos y herramientas que guíen al arquitecto durante la toma de decisiones.

En el trabajo de investigación realizado se ha hecho uso de la herramienta ArchE, un asistente de diseño arquitectónico que ofrece soporte a la toma de decisiones. Permite evaluar de un modo sistemático, mediante marcos de razonamiento, la arquitectura a través de escenarios de calidad.

Palabras clave.- Arquitectura software, Atributos de calidad, Tácticas arquitectónicas, Diseño sistemático, Attribute-Driven Design, ArchE

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

El presente trabajo se sitúa dentro de la línea del Desarrollo de Software y de las Arquitecturas Software. La estructura es la siguiente:

En el **primer punto** del documento, se tratan diversos aspectos relacionados con la Arquitectura Software, como la importancia de los atributos de calidad. Otro aspecto tratado son los escenarios de atributos de calidad, usado como medio para caracterizar los atributos de calidad. Se trata también la relación entre los atributos de calidad y las tácticas arquitectónicas. Por último, se dedica un apartado a una metodología para diseñar una arquitectura (Attribute-Driven Design). Todos estos aspectos se encuentran en el libro "*Software Architecture in Practice*" (Bass, Len; Clements, Paul; Kazman, Rick, 2003).

El **segundo punto** está dedicado a la herramienta de asistencia al diseño arquitectónico ArchE. En los distintos subapartados se repasa los conceptos fundamentales en los que se basa, los marcos de razonamientos de los que consta por defecto, cómo es la interacción con el usuario, su arquitectura, el procedimiento de instalación y su puesta en marcha.

En el **tercer punto** se realiza la evaluación del sistema software *Pharma* mediante la herramienta ArchE. Mediante la aplicación de tácticas se obtiene una mejora del sistema, cumpliendo así el **objetivo fundamental del trabajo**.

En el **cuarto punto**, se describen las conclusiones y perspectivas de continuación.

1. ARQUITECTURA SOFTWARE

Los sistemas software son construidos para satisfacer los objetivos de negocio de las organizaciones. La arquitectura es un puente entre los objetivos de negocio (a menudo abstractos) y el resultado del sistema (algo concreto).

1.1 ¿Qué es la Arquitectura del Software?

La arquitectura del software de un sistema es un conjunto de estructuras que comprenden elementos software, relaciones entre ellos y propiedades de ambos.

Una **estructura** es un conjunto de elementos que se mantienen juntos por una relación. Los sistemas software están compuestos de muchas estructuras y una estructura que no es única puede considerarse arquitectura. Pueden existir tanto en software como en hardware.

Hay tres **categorías de estructuras arquitectónicas**, que juegan un importante papel en el diseño, documentación, y análisis de arquitecturas. Dependiendo de la naturaleza de los elementos que muestran.

- **Estructuras modulares**: Son estructuras estáticas, dividen el sistema en unidades de implementación. La funcionalidad del sistema puede dividirse en módulos y asignarse a los equipos de desarrollo.

- **Estructuras de componentes y conectores**: Son dinámicas, se centran en la forma en la que los elementos interactúan con otros en tiempo de ejecución para llevar a cabo las funciones del sistema.

- **Estructuras de asignación**: Describen las relaciones desde las estructuras software hasta la organización del sistema, desarrollo, instalación y ejecución de los entornos. Muestran la relación entre los elementos software y elementos de uno o más entornos externos en el que el software es creado y ejecutado.

Estas principales categorías corresponden a los tres **tipos de decisiones** que implica el diseño arquitectónico.

- Estructuras modulares: incluye decisiones sobre cómo el sistema es estructurado como un conjunto de unidades de código o datos que tienen que ser construidas o conseguidas. En cualquier estructura modular, los elementos son módulos de algún tipo. Las vistas sirven para cuestionarse acerca de la modificabilidad del sistema.
- Estructuras de componentes y conectores: incluye decisiones sobre cómo el sistema es estructurado como un conjunto de elementos que tienen un comportamiento en tiempo de ejecución (componentes) e interacciones (conectores). Las vistas son especialmente importantes para cuestionarse acerca de propiedades en tiempo de ejecución del sistema como el rendimiento, seguridad o disponibilidad.
- Estructuras de asignación: incluye decisiones sobre cómo el sistema se relacionará con estructuras no software de su entorno.

Una estructura es arquitectónica si soporta razonamiento sobre el sistema y las propiedades del sistema. El razonamiento debería ser sobre un atributo del sistema que es importante para algún cliente.

Debido a que la arquitectura está compuesta de estructuras y las estructuras están compuestas de elementos y relaciones, una arquitectura abarca elementos software y cómo los elementos se relacionan entre ellos. Esto significa que de forma específica omite cierta información sobre los elementos y lo que no es útil para la lógica del sistema. De esta manera, una arquitectura es una **abstracción** de un sistema que selecciona ciertos detalles y suprime otros.

La abstracción es esencial para dominar la complejidad de un sistema.

Sistemas modernos son con frecuencia demasiado complejos para entenderlos del todo. Resulta mejor, centrarse en un número pequeño de estructuras del sistema. Para comunicar el significado sobre la arquitectura, debemos dejar claro qué estructura o estructuras estamos discutiendo, qué vista tomamos de la arquitectura.

Una **vista** es una representación de un conjunto congruente de elementos arquitectónicos y de relaciones entre ellos, escrita para que sea entendida por el grupo de personas interesadas en el sistema. Una vista es una representación de una estructura.

Las estructuras juegan un papel importante en la perspectiva de la arquitectura del software debido a la facultad analítica y de ingeniería que poseen. Cada estructura proporciona una perspectiva para razonar sobre alguno de los principales atributos de calidad.

Algunas **estructuras modulares útiles** incluyen las siguientes:

- Estructura de descomposición. Las unidades son módulos que están relacionados por la relación *es-un-submodulo-de*, mostrando cómo los módulos son descompuestos de forma recursiva en módulos más pequeños hasta que los módulos sean lo suficiente pequeños hasta ser fácilmente comprendidos. Esta estructura determina la modificabilidad del sistema, garantizando que los posibles cambios son localizados.
- Estructura de usos. En esta estructura las unidades son además de módulos, clases tal vez. Las unidades están relacionadas por la relación uses, como especializada forma de dependencia. Una unidad de software usa otra si la corrección de la primera requiere la presencia de una versión correcta de la segunda. La habilidad para crear fácilmente un subconjunto de un sistema permite el desarrollo incremental.
- Estructura capas. Los módulos en esta estructura son llamados capas. Una capa es una "máquina virtual" abstracta que proporciona un conjunto de servicios a través de una interfaz. Esta estructura es usada para favorecer la portabilidad de un sistema.
- Estructura de clases (o generalización). Las unidades de módulo en esta estructura son llamadas clases. La relación es de *herencia de* o *es una instancia de*. Soporta el razonamiento sobre colecciones de similar comportamiento o capacidad. Permite razonar sobre la reutilización y la adición incremental de funcionalidad.
- Estructura de modelo de datos. El modelo de datos describe la información estática de la estructura en términos de entidades de datos y sus relaciones.

Algunas **estructuras de componentes y conectores útiles**:

- Estructura de servicios. Las unidades son los servicios que interoperan con otros mediante mecanismos de coordinación de servicios como puede ser SOAP. La

estructura de servicio ayuda a dirigir la construcción de un sistema compuesto de componentes que pueden haber sido desarrollados de forma anónima e independiente entre sí.

- Estructura de concurrencia. Esta estructura permite al arquitecto determinar oportunidades para el paralelismo y las situaciones donde puede ocurrir el origen de un conflicto. Las unidades son componentes y los conectores son sus mecanismos de comunicación. Los componentes son organizados en hilos lógicos de ejecución.

Algunas **estructuras de asignación útiles**, son:

- Estructura de despliegue. Muestran cómo el software es asignado a los procesos de hardware y a los elementos de comunicación. Los elementos son elementos software, entidades hardware y canales de comunicación. Las relaciones son del tipo *asignado-a,* mostrando en qué unidades físicas residen los elementos software, y del tipo *migra-a* si la asignación es dinámica. Esta estructura puede ser usada para razonar sobre el rendimiento, integridad de datos, seguridad, y disponibilidad. Resulta interesante en sistemas distribuidos y paralelos.
- Estructura de implementación. Esta estructura muestra cómo los elementos software son mapeados a estructura/s de ficheros en el desarrollo del sistema, integración, o control de configuración de los entornos. Esto es crítico para la gestión de las actividades de desarrollo y construcción de procesos.
- Estructura de asignación de trabajo. Esta estructura asigna la responsabilidad para implementar e integrar los módulos a los equipos que lo llevarán a cabo. El arquitecto conocerá la pericia requerida de cada equipo. Esta estructura determinará la mayoría de las comunicaciones entre los equipos.

Cada una de estas estructuras proporcionan una diferente perspectiva pero no son independientes entre sí. Elementos de una estructura estarán relacionados con elementos de otras estructuras.

A la hora de elegir entre las diferentes estructuras disponibles, es necesario tener en cuenta aquellas que favorezcan a los atributos de calidad más importantes del sistema.

En algunos casos, los elementos arquitectónicos se componen de forma que resuelven problemas específicos. Estas composiciones de elementos arquitectónicos, son consideradas **patrones arquitectónicos**, proporcionan estrategias preparadas para resolver alguno de los problemas a los que se enfrenta un sistema.

Un patrón arquitectónico describe los tipos de elemento y la forma en la que interaccionan para resolver el problema.

Los patrones pueden ser caracterizados según el tipo de elementos arquitectónicos que ellos usan. Por ejemplo el patrón por capas (*Layered pattern*) ayuda a estructurar las aplicaciones que pueden ser descompuestas en grupos de tareas en las cuales cada grupo de tareas se encuentra en un nivel de abstracción diferente.

1.2 ¿Por qué es importante la Arquitectura del Software?

Algunos de los principales **motivos** por los que se considera importante, son:

1. Los atributos de calidad son determinados, esencialmente, por su arquitectura. Una buena arquitectura es necesaria, pero no suficiente, para asegurar la calidad.

Para lograr los atributos de calidad debe tenerse en cuenta además el diseño, la implementación y el despliegue.
2. Las decisiones tomadas en una arquitectura te permiten razonar y gestionar cómo el sistema evoluciona.
3. Es posible hacer predicciones de atributos de calidad sobre un sistema basándonos en la evaluación de sus arquitectura. Resulta beneficioso conocer el tipo de decisiones arquitectónicas que conducen a los atributos de calidad en un sistema.
4. Una arquitectura documentada mejora la comunicación entre todas las personas interesadas. La arquitectura es lo suficientemente abstracta para que la mayoría del personal no técnico pueda comprenderla adecuadamente (o con cierta ayuda del arquitecto). La arquitectura puede ser refinada en especificaciones técnicas lo suficiente para guiar en la implementación, integración, pruebas y despliegue.
5. La arquitectura software es una manifestación de las decisiones tempranas de diseño sobre el sistema, suponen un peso importante.
6. Una arquitectura define un conjunto de restricciones sobre la implementación posterior.
7. La arquitectura dicta la estructura de la organización o viceversa.
8. Una arquitectura puede proporcionar la base para un prototipo en evolución.
9. Una arquitectura es el elemento clave que permite al arquitecto y al director del proyecto razonar sobre el coste y la planificación.
10. Una arquitectura puede ser creada como modelo reutilizable que sea el corazón de una línea de productos.
11. El desarrollo basado en arquitecturas se centra en componer y ensamblar elementos que sean probables desarrollarlos por separado.
12. Los patrones arquitectónicos guían al arquitecto, centrándose en gran parte en los atributos de calidad, reduciendo la complejidad del diseño y el sistema.
13. Una arquitectura puede ser la base de entrenamiento a un nuevo miembro del equipo. La arquitectura, incluye una descripción de cómo los elementos interactúan entre sí y su comportamiento.

1.3 La arquitectura en el contexto del ciclo de vida de un proyecto

Los procesos de desarrollo software son propuestas estándar para desarrollar sistemas software. Hay cuatro **procesos** dominantes **del desarrollo software**, los cuales se describen en el orden en el que aparecieron:

1. **Ciclo de vida en cascada**. El modelo organiza el ciclo de vida en una serie de actividades secuenciales conectadas. El proceso empieza con la especificación de requisitos, seguido por el diseño, la implementación, la integración, pruebas, instalación y mantenimiento.
2. **Ciclo de vida iterativo**. Considera el desarrollo del software como una serie de ciclos cortos, en la que cada iteración se acerca a la última solución software para el problema planteado. Un proceso iterativo conocido es *Rational Unified Process* (Proceso Racional Unificado). Este proceso define cuatro fases en cada iteración: inicio, elaboración, construcción y transición.
3. **Desarrollo ágil**. Las metodologías de desarrollo más conocidas son Scrum, Extreme Programming y Crystal Clear. Todas son incrementales e iterativas. Se centran en el trabajo en equipo, la adaptabilidad y la estrecha colaboración (dentro del equipo y con el cliente/usuario final). Estas metodologías suponen que los requisitos cambian durante el ciclo de vida del proyecto.
4. **Desarrollo dirigido por modelos (MDD)**. Se basa en la idea de que las personas no deberían escribir código en lenguajes de programación, sino crear modelos de dominio, desde el cual el código es generado automáticamente. La persona crea el

Modelo Independiente de la Plataforma, *Platform Independent Model* (PIM), que es combinado con el Modelo Específico de la Plataforma, *Platform Specific Model* (PSM), para generar el código ejecutable. De esta forma el modelo PIM es una realización de los requisitos funcionales mientras que el modelo PSM dirige la plataforma específica y los atributos de calidad.

No importa el proceso de desarrollo software o modelo de ciclo de vida que uses, hay un número de **actividades** que están **implicadas en la creación de una arquitectura software**. El proceso determinará cómo de a menudo realizas cada una de las siguientes actividades:

- Crear un caso de negocio para el sistema.
- Entender los requisitos significativos de la arquitectura.
- Crear o elegir la arquitectura.
- Documentar y comunicar la arquitectura.
- Analizar o evaluar la arquitectura.
- Implementar y probar el sistema basado en la arquitectura.
- Asegurar que la implementación se ajusta a la arquitectura.

En cualquier proceso de diseño habrá varios candidatos de diseño a considerar. Elegir entre los candidatos de una forma racional es uno de los grandes retos del arquitecto.

Evaluar una arquitectura para el soporte de atributos de calidad, es esencial para asegurarse que el sistema construido desde esa arquitectura satisface las necesidades del grupo de personas interesadas. La técnica basada en escenarios proporciona una de las más efectivas estrategias para evaluar una arquitectura. La estrategia más madura es Método de Análisis de Acuerdos de Arquitectura, *Architecture Tradeoff Analysis Method (ATAM).*

Los sistemas son creados para satisfacer los objetivos de negocio de una o más organizaciones. Muchos de estos objetivos tendrán una profunda influencia en la arquitectura.

Una arquitectura software es el resultado de influencias del negocio, sociales y técnicas. La existencia de una arquitectura afecta por este orden a la técnica, negocio, y entorno social que a su vez influye en futuras arquitecturas.

El contexto técnico, del ciclo del proyecto, de negocio y profesional influyen en el arquitecto y la arquitectura como se puede ver en el siguiente esquema:

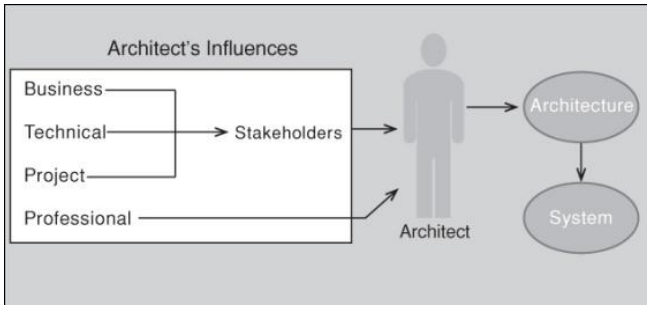


Figura 1. Influencias en el arquitecto y la Arquitectura

Los cuatro contextos referidos son:

1. **Técnico**. Incluye el logro de los requisitos de los atributos de calidad y la tecnología actual.
2. **Ciclo de vida del proyecto**. Debe hacerse los casos de uso del sistema, entender los requisitos significativos de la arquitectura, crear o seleccionar la arquitectura, documentarla, analizarla o evaluarla, implementar y probar el sistema basado en la arquitectura, y asegurarse que la implementación cumple con la arquitectura.
3. **Negocio**. El sistema creado a partir de la arquitectura debe satisfacer los objetivos de negocio del grupo de personas interesadas en él.
4. **Profesional**. Debe tener ciertas habilidades y conocimientos para ser un arquitecto.

1.4 Atributos de calidad y escenarios

Muchos factores determinan las cualidades que debe ser proporcionadas a la arquitectura de un sistema. Estas cualidades van más allá de la funcionalidad, declaración básica de las capacidades, servicios y comportamiento del sistema. El mapeo de una funcionalidad del sistema a estructuras software, es lo que determina el soporte de la arquitectura para las cualidades.

Un **atributo de calidad** es una propiedad medible de un sistema que es usada para indicar cómo de bien el sistema satisface las necesidades de sus interesados.

El papel que juega los **requerimientos** de un sistema en la arquitectura. Todos los requerimientos se incluyen en las siguientes **categorías**:

Requisitos funcionales. Estos requerimientos declaran lo que el sistema debe hacer, y cómo comportarse o reaccionar ante un estímulo en ejecución.

Requisitos de atributos de calidad. Estos requisitos son capacidades de los requisitos funcionales o del producto en su conjunto. Una capacidad de un requerimiento funcional es por ejemplo la rapidez con la que una función debe realizarse. Una capacidad del producto en su conjunto es por ejemplo el tiempo para desplegar el producto.

Restricciones. Una restricción es una decisión de diseño con limitaciones. Las limitaciones pueden venir impuestas por la competencia del arquitecto o por factores externos.

Los requisitos funcionales son satisfechos asignando una secuencia apropiada de responsabilidades a través del diseño. La asignación de responsabilidades a elementos de la arquitectura es una decisión fundamental en el diseño arquitectónico.

Los requisitos de atributos de calidad son satisfechos por las diversas estructuras diseñadas en la arquitectura, y el comportamiento e interacciones de los elementos que forman esas estructuras.

Las restricciones son satisfechas cumpliendo la decisión de diseño y conciliándola con otras decisiones que afectan al diseño.

El logro de atributos de calidad causa responsabilidad. Se puede identificar las responsabilidades que van a ser asociadas con un conjunto de requisitos particular.

Los atributos de calidad han sido de interés para la comunidad del software desde 1970. Existe una gran variedad de definiciones y clasificaciones publicadas hasta hoy.

Desde la perspectiva de un arquitecto, hay tres problemas en relación a los atributos de calidad de un sistema:

- La definición dada para un atributo no se puede probar.
- A qué particular interés pertenece una cualidad.
- Cada atributo puede ser referido usando diferentes términos.

Una solución a las dos primeras cuestiones es usar **escenarios de atributos de calidad** como medio de caracterizar los atributos de calidad. Una solución a la tercera cuestión es ilustrar los conceptos que son fundamentales para ese atributo.
Dentro de los sistemas complejos, los atributos de calidad nunca pueden ser logrados de forma aislada. La consecución de uno de ellos tendrá un efecto, algunas veces positivo o negativo, sobre el logro de otros atributos de calidad.

Para que los requisitos de un atributo de calidad no resultaran ambiguos y se pudieran probar, los autores del texto base definieron una forma común para especificar todos los requisitos de atributos de calidad.

El escenario de un atributo de calidad es un requisito específico de un atributo de calidad. Consiste en seis **partes**:

Estímulo.- Es una condición que requiere una respuesta cuando llega al sistema.

Fuente del estímulo.- Un estímulo tiene un origen. Alguna entidad (persona, sistema informático, etc.) que genera el estímulo.

Entorno.- El entorno de un requisito es un conjunto de circunstancias en las que el escenario tiene lugar.

Artefacto.- Es la porción del sistema a la que se aplica el requisito. Puede ser a una colección de sistemas, a un sistema completo o a partes específicas del sistema.

Respuesta.- Debe ser especificado cómo el sistema debería responder al estímulo. La respuesta consiste en las responsabilidades que el sistema (para cualidades en tiempo de ejecución) o desarrolladores (para cualidades en tiempo de desarrollo) deberían actuar en respuesta al estímulo.

Medida de la respuesta.- Cuando una respuesta ocurre, debería ser medible de alguna forma para que el requisito pueda ser probado.

Se distingue el escenario general de un atributo de calidad, que puede pertenecer a cualquier sistema, del escenario concreto de un atributo de calidad, que son específicos de un sistema particular.

Los atributos de calidad se pueden caracterizar como un colección del escenario general. Para traducir estas caracterizaciones genéricas de atributo a requisitos de un sistema particular, el escenario general necesita ser adaptado al sistema específico.

Figura muestra las partes de un escenario de atributo de calidad.

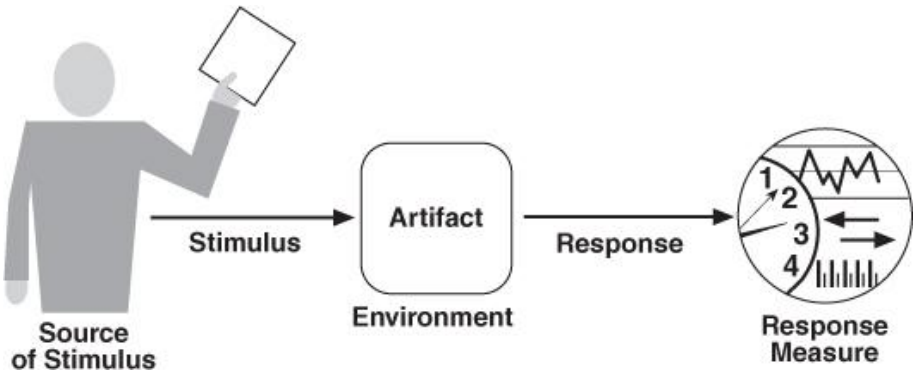


Figura 2. Partes de un escenario de atributo de calidad

1.5 Atributos de calidad y tácticas arquitectónicas

Los requisitos de un atributo de calidad especifican las respuestas del sistema que comprende los objetivos de negocio. Las técnica que un arquitecto puede usar para lograr los atributos de calidad requeridos son llamadas **tácticas arquitectónicas**. Una táctica es una decisión de diseño que influye en la consecución de un atributo de calidad, las tácticas afectan directamente a la respuesta del sistema a algún estímulo.

Una táctica se centra en la respuesta de un único atributo de calidad. Las tácticas son técnicas de diseño que los arquitectos han usado durante años.

Un diseño de un sistema consiste en una colección de decisiones. Algunas de estas decisiones ayudan a controlar las respuestas del atributo de calidad; otras aseguran la consecución de la funcionalidad del sistema.

En la siguiente figura se representa la relación entre estímulo, tácticas y respuesta.

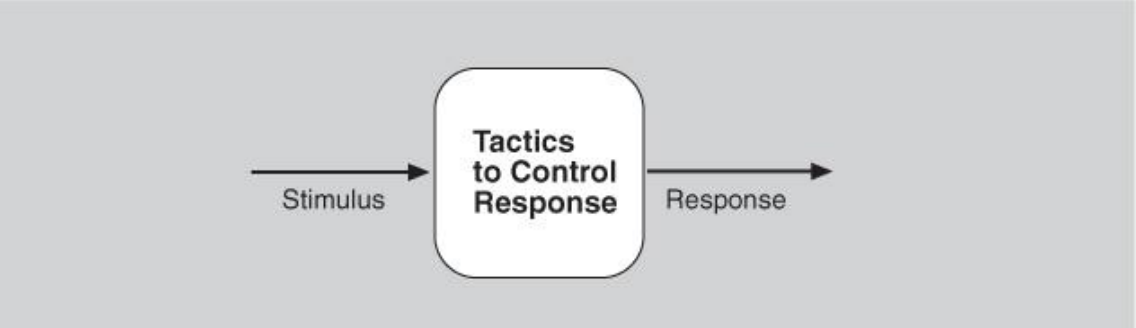


Figura 3. Tácticas para controlar la respuesta al estímulo

Las tácticas suelen organizarse en jerarquías y existen para cada uno de los atributos de calidad. A continuación se describe siete de los más importantes atributos de calidad, por su ocurrencia, en los sistemas software modernos.

Availability (disponibilidad).- Se refiere a la habilidad del sistema para estar disponible para el uso, especialmente después de que un fallo ocurra.

Interoperability (interoperabilidad).- Trata acerca del acuerdo entre dos o más sistemas que de forma útil intercambian información significativa por medio de interfaces en un contexto particular.

Modifiability (modificabilidad).- Trata sobre el cambio, el interés se centra en el coste (tiempo y/o dinero) y riesgo de hacer cambios que puedan afectar a otras funciones o atributos de calidad.

Performance (rendimiento).- Trata sobre habilidad del sistema software para cumplir con los requisitos en un tiempo.

Security (seguridad).- Es una medida de la habilidad del sistema para proteger datos e información de accesos no autorizados mientras que se proporciona acceso a la gente y sistemas que son autorizados.

Testability (testabilidad).- Trata sobre la facilidad con la que se puede comprobar el funcionamiento de un sistema, en especial durante la fase de desarrollo para encontrar errores.

Usability (usabilidad).- Se preocupa de cómo de fácil es para el usuario completar una tarea deseada y el tipo de soporte que el sistema proporciona al usuario.

Esta clasificación se ha tomado de la 3ª Ed. del libro *Software Architecture in Practice*, en ella aparece un nuevo atributo que no estaba incluido en la 2ª Ed., que es el atributo de interoperabilidad. Su consideración se debe a que los sistemas cada vez más tienden a intercambiar información.

Por este motivo, para ejemplificar los conceptos de escenario general, escenario concreto y tácticas de un atributo de calidad, he elegido el de interoperabilidad. Por otro lado, en los siguiente apartados de la memoria se van a ver con más detalle estos conceptos para los atributos de modificabilidad y rendimiento.

Las partes de un **escenario general de interoperabilidad** son:

- Fuente del estímulo.- Un sistema inicia una petición para interoperar con otro sistema.
- Estímulo.- Una petición para intercambiar información entre sistema/s.
- Artefacto.- El sistema que desea interoperar.
- Entorno.- Los sistemas que desean interoperar son descubiertos en tiempo de ejecución o son conocidos antes.
- Respuesta.- La solicitud para interoperar es manejada en el intercambio de información. La información sintáctica y semántica recibida es comprendida. De forma alternativa, la petición es rechazada y son notificadas las entidades apropiadas. En cualquier caso, la petición puede ser trazada.
- Medida de la respuesta.- El porcentaje de información intercambiada procesada correctamente o el porcentaje de información intercambiada rechazada correctamente.

En la siguiente figura se muestra un **ejemplo** de un **escenario concreto de interoperabilidad**:

"Nuestro sistema de información del vehículo envía la localización al sistema de monitorización de tráfico. El sistema de monitorización de tráfico combina nuestra localización con otra información, cubriendo esta información en Google Map, y la transmite. La información de la localización es correctamente incluida con una probabilidad del 99.9%".

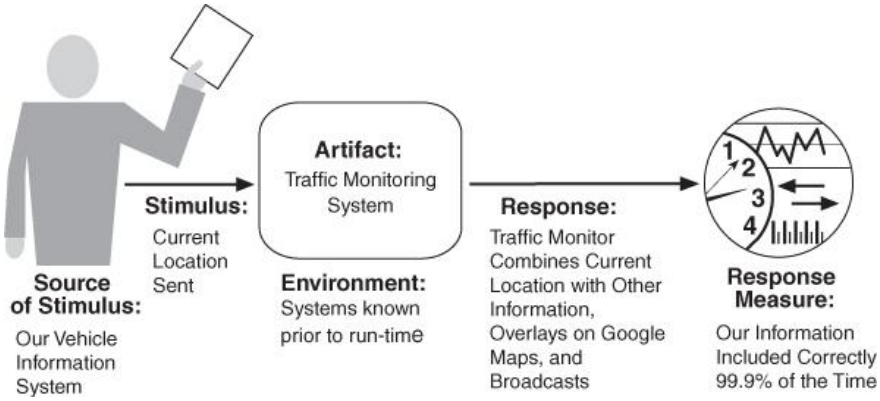


Figura 4. Ejemplo de un escenario concreto de interoperabilidad

Se identifican dos categorías de tácticas de interoperabilidad.

1. Localizar.- Hay una sola táctica en esta categoría. Es usada cuando el sistema que interopera debe ser descubierto en tiempo de ejecución.

    - Descubrir servicio. Localiza un servicio buscando un conocido directorio del servicio. El servicio puede ser localizado por el tipo de servicio, por nombre, por localización o por otro atributo.

2. Manejar interfaz.- Consiste en dos tácticas.

    - Orquestación. Es una táctica que usa un mecanismo de control para coordinar y gestionar la invocación de servicios particulares. La orquestación es usada cuando la situación de interoperabilidad entre sistemas es compleja.
    - Interface a medida. Es una táctica que añade o quita capacidades a una interfaz. El patrón decorador es un ejemplo de este tipo de táctica.

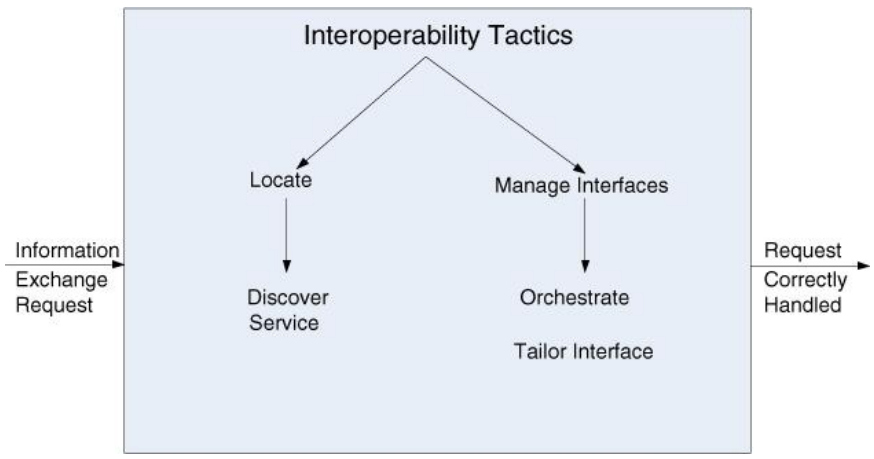


Figura 5. Resumen de las tácticas para lograr la interoperabilidad

Catalogando las tácticas, proporcionamos una manera de hacer el diseño más sistemático dentro de algunas limitaciones. De forma frecuente será necesario elegir entre varias tácticas para mejorar un atributo de calidad particular. La elección depende de factores como la mejora que ofrece de los atributos de calidad y el coste para implementarlo.

Puede que la aplicación de una táctica para mejorar un atributo de calidad, suponga empeorar otro atributo de calidad. Las tácticas se pueden usar de forma provechosa estimando si el efecto final de emplear una táctica, supone algún beneficio en el atributo de calidad que nos interesa renunciando a algo más.

1.6 Tácticas y Patrones Arquitectónicos

Las tácticas y patrones arquitectónicos son formas de capturar el buen diseño de estructuras ya probadas, para que puedan ser reutilizado. Constituyen las principales herramientas del negocio de la arquitectura del software.

Un **patrón arquitectónico**:

- Es un paquete de decisiones de diseño que son encontradas en la práctica de forma repetida.
- Tiene propiedades que permiten su reutilización, y
- Describe una clase de arquitecturas.

Las tácticas son más sencillas que los patrones. Las tácticas típicamente usan una única estructura o mecanismo de computación. Por este motivo dan un mayor control a un arquitecto cuando toma decisiones de diseño que los patrones, los cuales combinan múltiples decisiones de diseño en un paquete.

Las tácticas son los "bloques de construcción" de diseño de los patrones arquitectónicos que son creados.

Un patrón arquitectónico establece una relación entre:

- Un **contexto**. Situación común en el mundo que presenta un problema.
- Un **problema**. El problema, apropiadamente generalizado, que se presenta en el contexto dado. La descripción del problema a menudo incluye atributos que deben ser encontrados.
- Una **solución**. Una exitosa resolución arquitectónica al problema, abstracta. La solución describe las estructuras arquitectónicas que resuelven el problema, las responsabilidades y las relaciones entre los elementos.

La descripción de la solución debería además dejar claro que atributos de calidad son proporcionados por los elementos estáticos y los elementos configurados en tiempo de ejecución.

Un patrón es descrito como una solución a una clase de problemas en un contexto general. Cuando un patrón es elegido y aplicado, el contexto de su aplicación se hace muy específico.

Los patrones pueden ser categorizados por el tipo de elemento predominante que ellos muestran: los patrones de módulos muestran módulos, los patrones de componentes y conectores muestran componentes y conectores, y los patrones de asignación muestran una combinación de elementos software (módulos, componentes, conectores) y elementos no software.

La mayoría de los patrones consisten en varias tácticas diferentes, y aunque estas tácticas podrían cumplir con una finalidad común, a menudo son escogidas para fomentar diferentes atributos de calidad.

La siguiente figura muestra patrones arquitectónicos descritos en el libro *Pattern-Oriented Software Architecture Volume 1: A System of Patterns,* by Buschmann et al., y cuáles tácticas de modificabilidad emplean:

	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							

Figura 6. Correspondencia entre tácticas de modificabilidad y patrones

Las tácticas se pueden usar para mejorar patrones en general. Si un patrón tiene una debilidad respecto a cierto atributo de calidad (p.e. rendimiento) en una situación más específica de lo habitual, usar tácticas puede ser útil.

1.7 Diseñar una Arquitectura

Una estrategia presentada por los autores para diseñar una arquitectura que satisfaga los requisitos de calidad y los requisitos funcionales, es el método Attribute-Driven Design (Diseño Dirigido por Atributos de calidad).

Este método puede ser visto como una extensión de otros métodos de desarrollo, como RUP (Rational Unified Process).

Es un proceso de descomposición recursiva donde, en cada iteración, tácticas y patrones arquitectónicos son elegidos para satisfacer un conjunto de escenarios de calidad y entonces la funcionalidad es asignada para instanciar los tipos de módulos proporcionados por el patrón.

ADD puede comenzar cuando un conjunto de requerimientos (funcionales, de calidad, y restricciones) significativos de la arquitectura son conocidos.

ADD es un método de cinco pasos:

1. Elegir el elemento del sistema a diseñar. Para diseños no iniciados, la parte para comenzar es el sistema completo. Para diseños que ya han sido parcialmente completados (por restricciones externas o por anteriores iteraciones del proceso), la parte es un elemento que aún no ha sido diseñado.

2. Identificar los requerimientos significativos de la arquitectura para el elemento elegido.
3. Generar una solución de diseño para el elemento elegido, usando el diseño de sistemas existentes, frameworks, patrones, tácticas y listas de verificación.
4. Verificar y refinar los requisitos y generar la entrada para la siguiente iteración. Algún diseño del paso 3 cumplirá todos los requisitos de los elementos elegidos o no lo hará. Si no lo cumple, entonces alguno de ellos puede ser asignado a los elementos que serán elaborados en futuras iteraciones del proceso ADD, o si el existente diseño es inadecuado, debemos volver atrás. Los requisitos no significativos, serán satisfechos, asignados a elementos hijos o marcados como no alcanzables.
5. Repetir los pasos 1–4 hasta que todos los requisitos significativos hayan sido satisfechos o hasta que la arquitectura haya sido elaborada suficientemente para que sea usada por los que vayan a implementarla.

La salida del método ADD no es una arquitectura completa detallada, pero es una arquitectura en la que el principal diseño que se aproxima ha sido seleccionado y examinado. Esto produce una rápida y temprana arquitectura operativa, que puede ser repartida a los equipos del proyecto para que puedan empezar su trabajo mientras se sigue refinando la arquitectura.

2. LA HERRAMIENTA ARCHE

El prototipo de investigación Architecture Expert (ArchE) fue creado, para el ámbito educativo, por el instituto Software Engineering Institute (SEI). La herramienta ArchE es un asistente de diseño que ayuda a los arquitectos software y diseñadores a explorar alternativas eficaces para un conjunto particular de requisitos. Las alternativas son propuestas en términos de tácticas (transformaciones) arquitectónicas sugeridas por marcos de razonamiento.

Un marco de razonamiento es una abstracción que encapsula el conocimiento necesario para entender y estimar el comportamiento de un sistema con respecto a un atributo de calidad particular, para que este conocimiento pueda ser aplicado por personal no experto. Tener encapsulados modelos para los atributos de calidad tiene ventajas en cuanto a nivel de detalles, porque ayuda al arquitecto a manejar las relaciones entre varios modelos de atributos de calidad mientras diseña una arquitectura.

ArchE fue implementada para poder integrarse con la plataforma Eclipse. De esta manera incorpora como plugins dos marcos de razonamiento (Reasoning Frameworks – RF) correspondientes a los atributos de calidad de modificabilidad y rendimiento. En su última versión, la 3.0 desarrollada en 2008, permite extender la herramienta con nuevos marcos de razonamiento sobre otros atributos de calidad implementando la interfaz habilitada para ello, ArchE Reasoning Framework Interface (ArchE RFI).

2.1 Conceptos esenciales en ArchE

El funcionamiento de ArchE se basa en los siguientes conceptos:

- **Escenarios**: Para cada escenario concreto de un atributo de calidad se especifica mediante un formulario las seis partes que lo componen (estímulo, fuente del estímulo, entorno, artefacto, respuesta y medida de la respuesta). Representan los requisitos de calidad del sistema.

- **Responsabilidades**: Son las actividades que lleva a cabo el software que está siendo diseñado. Se usa las responsabilidades para expresar los requisitos funcionales como parte de los escenarios de atributos de calidad. Las responsabilidades pueden incluir propiedades específicas de un atributo de calidad o formar parte de una relación (entre responsabilidades o escenarios). Toda esta información es recogida por los marcos de razonamiento para crear la arquitectura inicial.

- **Marcos de razonamiento (RF)**: A partir de los escenarios y las responsabilidades introducidas como entradas, cada marco de razonamiento sobre un atributo de calidad evalúa si la arquitectura cumple o no los requisitos definidos, proponiendo tácticas en el caso de encontrar soluciones para mejorar la arquitectura inicial.

- **Tácticas arquitectónicas**: Son el medio para satisfacer la medida de respuesta esperada de un atributo de calidad, manipulando algún aspecto de un atributo de calidad del modelo a través de decisiones de diseño.

2.2 Marcos de razonamiento en ArchE

El SEI desarrolló dos marcos de razonamiento como plugins usando la interfaz *ArchE-RF Interface*.

2.2.1 Modifiability Reasoning Framework

Este marco de razonamiento se basa en el análisis del impacto que supone realizar cambios.

La modificabilidad se considera como la habilidad de la arquitectura software para adaptarse a los cambios. Dado un conjunto de escenarios de cambios, el nivel de modificabilidad de una arquitectura va en función de cómo la funcionalidad es asignada a los módulos y cómo estos módulos interactúan entre sí.

La arquitectura es interpretada como un grafo, en el cual los nodos corresponden a unidades de cambio (las responsabilidades) mientras que los enlaces representan las dependencias entre los nodos (dependencias funcionales).

La relación de dependencia para módulos tiene un comportamiento similar a la dependencia de responsabilidades, teniendo asociado probabilidades de efecto de cambio para las entradas y salidas.

Mientras el marco de razonamiento se ejecuta, el procedimiento de interpretación filtrará esos elementos y las relaciones de diseño de la vista de módulos que están relacionados a responsabilidades de un escenario específico, para construir el grafo de la arquitectura. Este grafo será evaluado según una fórmula de coste.

En la siguiente ilustración se muestra un ejemplo de la aplicación de la táctica que separa una responsabilidad en dos responsabilidades hijas para reducir el costo de la modificación.

El proceso de interpretación-evaluación-rediseño continúa hasta que el análisis de los escenarios alcanzan valores que satisfacen las expectativas del arquitecto.

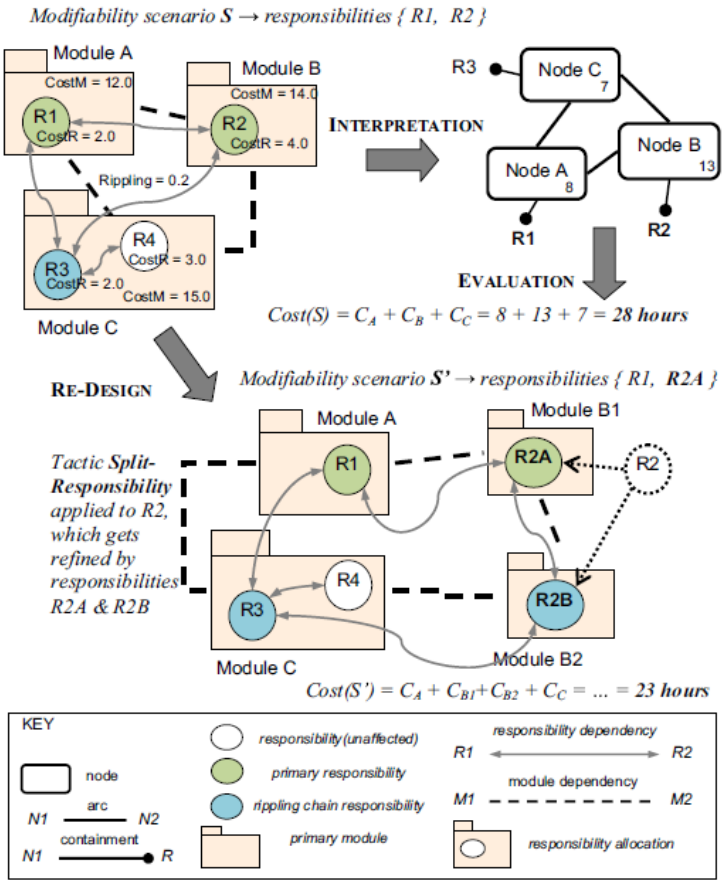


Figura 7. Interpretación, evaluación y rediseño para Modificabilidad

Las tácticas de modificabilidad que podemos encontrar en ArchE para su aplicación, son las siguientes:

- Reducir el costo de modificar una única responsabilidad
  - Separación o división de responsabilidades (para habilitar nuevos flujos de responsabilidades y creando así nuevos módulos con sus respectivos costos y atributos).

- Par reducir acoplamientos
  - Encapsulación
  - Usar un Wrapper (envoltorio)
  - Aumentar el nivel de abstracción
  - Usar un intermediario

- Para incrementar la cohesión
  - Mantener la coherencia semántica
  - Servicios abstractos comunes

2.2.2 Performance Reasoning Framework

El marco de razonamiento de rendimiento fue concebido originalmente por sus desarrolladores como un "analizador" sin soporte para tácticas. Evalúa la arquitectura en cuanto a su rendimiento, ofreciendo al usuario los resultados. Encapsula la suite MAST (*Modeling And Analysis Suite For Real-Time Applications*) para proveer la funcionalidad de analizar.

La implementación convierte los escenarios de rendimiento y sus responsabilidades en tareas, considerando las relaciones entre responsabilidades como eventos entre tareas. Lo que hace es transformar la descripción de la arquitectura a un modelo intermedio ICM (*Intermediate Constructive Model*) para poder ser analizado. El modelo de tareas es enviado a un fichero como entrada a la herramienta MAST. Los resultados de respuesta en el peor de los casos, son comparados con los requisitos de tiempo para determinar la capacidad de planificación del sistema.

Durante la experiencia de uso con este framework he confirmado este comportamiento en la versión por defecto de la herramienta ArchE v3. Por otra parte, he podido encontrar una publicación que habla sobre el logro de la inclusión de tácticas de rendimiento en ArchE v3.

2.3 Interacción de un arquitecto con ArchE

Cuando se utiliza la herramienta ArchE como soporte al diseño de una arquitectura que debe cumplir una serie de requisitos, se da el siguiente flujo de actividades:

1. El arquitecto introduce como entrada los requisitos funcionales por medio de funciones.
2. De forma automática, por cada función ArchE crea una responsabilidad, elemento que sí es manejable por la herramienta y que posee atributos adicionales (según los marcos de razonamiento cargados).
3. Se debe indicar, si existe, las relaciones entre las responsabilidades (tipo de relación disponibles según los RF cargados).
4. Se especifica los requisitos de calidad a través de los escenarios concretos de atributos de calidad. En la creación de cada escenario se elige el RF del atributo de calidad en cuestión.
5. Se indican las relaciones existentes entre escenarios-responsabilidades.
6. A partir de estos datos ArchE presenta un diseño inicial.
7. Del resultado de la evaluación, identifica aquellos escenarios que no cumplen con los requisitos de calidad exigidos.
8. Presenta al usuario un listado de tácticas procedentes del RF para mejorar el diseño con respecto a los requisitos de atributos de calidad.
9. El arquitecto escoge aplicar una o varias tácticas para la mejora del diseño. Si es requerido, proporciona además información adicional para los nuevos elementos de diseño.
10. ArchE aplica dichas tácticas para producir un nuevo modelo de diseño.

Este proceso se repite hasta que se alcanzan todos los requisitos de atributos de calidad, el arquitecto considere que la solución obtenida es aceptable o ArchE no realice más propuestas para mejorar el modelo.

Como se ha podido comprobar, ArchE pretende ser usado de un modo colaborativo con el arquitecto software. Existe una alta interacción. ArchE tiene conocimiento de atributos de calidad y cómo relacionar requisitos de calidad a la arquitectura de diseño,

pero no tiene conocimiento de la semántica del sistema que está siendo diseñado. Por este motivo, el arquitecto es el que decide entre las alternativas de diseño propuestas, considerando las más apropiadas en circunstancias particulares.

2.4 Infraestructura de ArchE

El funcionamiento de ArchE sigue el estilo arquitectónico blackboard (pizarra), en el cual diferentes actores colaboran para producir una solución a un problema. Cada actor puede leer información de la pizarra que fue desarrollada por otros actores, y a la inversa, cada actor puede introducir nueva información a la pizarra que podría ser de interés a otro.

Una representación de este estilo es que los marcos de razonamiento pueden ser considerados como fuentes de conocimiento, y ArchE es el componente de control que maneja la interacción entre ellos, para asegurar el progreso en el proceso de diseño.

La siguiente figura muestra una vista simplificada de la interacción de ArchE y los marcos de razonamiento:

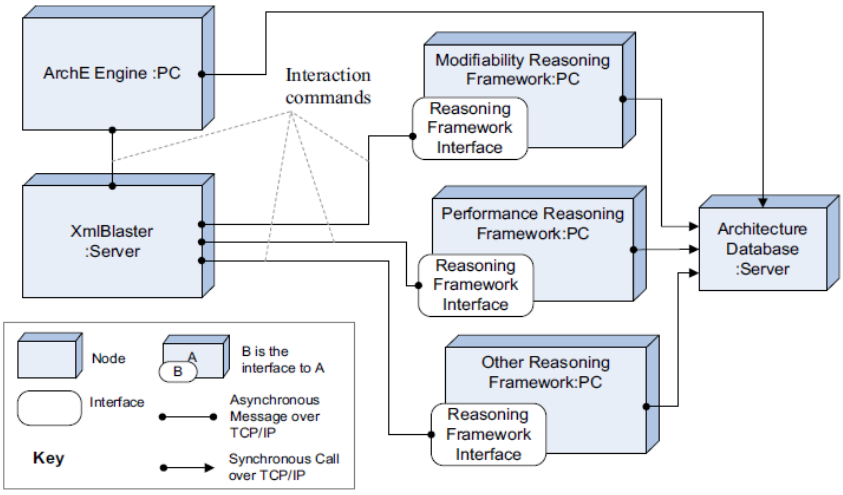


Figura 8. Integración de un marco de razonamiento externo con ArchE Engine

La infraestructura de colaboración depende de cuatro componentes:

- ***ArchE Engine***. Este componente conserva la funcionalidad de la primera versión con respecto a la estructura general de la búsqueda de alternativas para la arquitectura. Delega el trabajo de diseño a los marcos de razonamiento. Este motor tiene muy pocos conocimientos de algunas de las técnicas de diseño para atributos de calidad o de la semántica del sistema que está siendo diseñado. Las responsabilidades del motor son: procesar las entradas de usuario, actualizar la interfaz, coordinación de los marcos de razonamiento, presentación de sus resultados, y mostrar cuestiones al usuario.

- **XmlBlaster**. Es un middleware para la gestión de mensajes entre los participantes de la red de comunicación.

- **Reasoning Framework Interface**. Es la interfaz que debe implementar cada marco de razonamiento (Modifiability, Performance u otros). Abstrae de los detalles sobre el funcionamiento con XmlBlaster, el protocolo de comunicación entre ArchE

y el marco de razonamiento, y de los algoritmos que ejecutan los comandos de interacción.

- **Architecture Database**. Repositorio usado para manejar la persistencia de datos que necesitan ser compartidos por ArchE y los marcos de razonamiento participantes (p.e., los escenarios). Es la base de datos MySQL.

ArchE emplea el motor de reglas Jess que le permite actuar como un sistema experto. Jess mantiene la estructura de datos en un FactBase (repositorio de datos) y mantiene un conjunto de reglas para operar sobre el FactBase. Cada conjunto de reglas es disparada por los datos del repositorio y el motor de reglas coloca, a su vez, nuevos datos en el mismo.

Como ya se ha comentado, ArchE se integra con la plataforma Eclipse. Esto se ha conseguido gracias a la aplicación del patrón arquitectónico *Publish-subscribe* (variante del estilo arquitectónico *blackboard*). Donde varios componentes independientes interactúan mediante eventos e intercambian información.

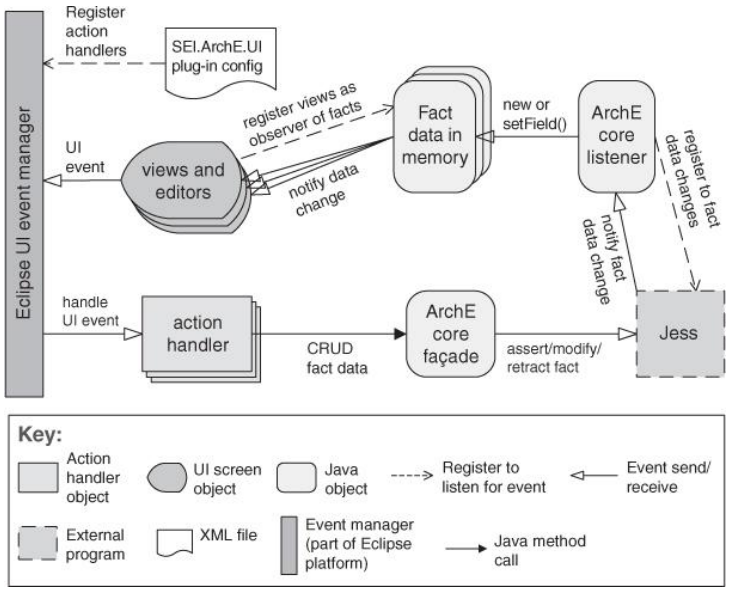


Figura 9. Eclipse y ArchE: Publish-subscribe

2.5 Procedimiento de instalación

La página web del Trabajo Fin de Máster, publicada por el equipo docente, proporciona información fundamental acerca de los pasos y los paquetes que son necesarios instalar, para poder ejecutar ArchE. Esto es esencial, ya que la herramienta dejó de ser soportada por el SEI en años anteriores.

El funcionamiento de ArchE se basa en el sistema experto JESS, la plataforma de desarrollo Eclipse, MySql y el servidor de comunicaciones XmlBlaster. El sistema operativo donde se ha instalado todo es Windows 7 de 64 bits.

La instalación de los distintos componentes o paquetes, puede realizarse de forma satisfactoria en el siguiente orden:

1.- La última versión Java SE 5.0, jdk-1_5_0_22-windows-i586-p.exe. ArchE no es compatible con versiones superiores.

2.- IDE Eclipse Europa SDK 3.3, Eclipse-SDK-3.3-win32.zip.

3.- Framework GEF, GEF-runtime-3.3.zip.

4.- El sistema de gestión de bases de datos MySQL, Mysql-essential-5.0.67-win32.msi.

5.- Sistema Experto JESS, Jess71p1.zip. Este archivo corresponde a una licencia educativa, puesta a disposición del alumnado.

6.- Gestor de comunicaciones basado en xml, xmlBlaster, xmlBlaster_REL_1_6_1.zip.

7.- El último paso requerido es ejecutar el instalador de ArchE que se encuentra en ArchE_3_0_0_Installer.zip.

8.- Paquete de librerías Hibernate para Eclipse, HibernateTools-3.2.3.GA.zip . No es estrictamente necesario, se ha instalado para hacer uso del marco de razonamiento Availability RF.

Se describe a continuación con más detalle el proceso de instalación en cada paso.

Paso 1, Instalar Java SE 5.0.

Para poder ejecutar las aplicaciones de Eclipse y ArchE, que utilizan tecnología Java, es necesario tener instalado el componente JRE (Java Runtime Environment) o JDK (Java Development Kit). El JRE contiene la JVM (Java Virtual Machine) y el JDK contiene el JRE y permite además desarrollar aplicaciones basadas en Java.

Se ha comprobado que versiones superiores a la JDK 5, no son totalmente compatibles con ArchE. Para comenzar la instalación se ejecuta el archivo ejecutable antes señalado:

Figura 10. Archivo de instalación JDK

El asistente muestra la ruta de instalación por defecto (*C:\Archivos de Programa\Java\... o similar*) del JDK y el JRE. Si se desea, esta ruta puede cambiarse por otra (como en este caso).

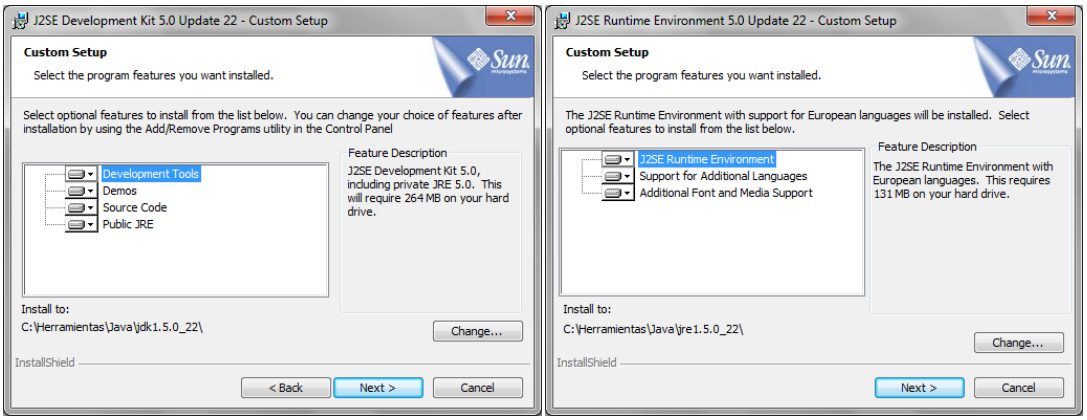


Figura 11. Ruta de instalación JDK

Después de finalizar la instalación del JDK y el JRE es necesario definir la variable de entorno del sistema *JAVA_HOME* para que pueda ser referenciada a nivel de sistema operativo:

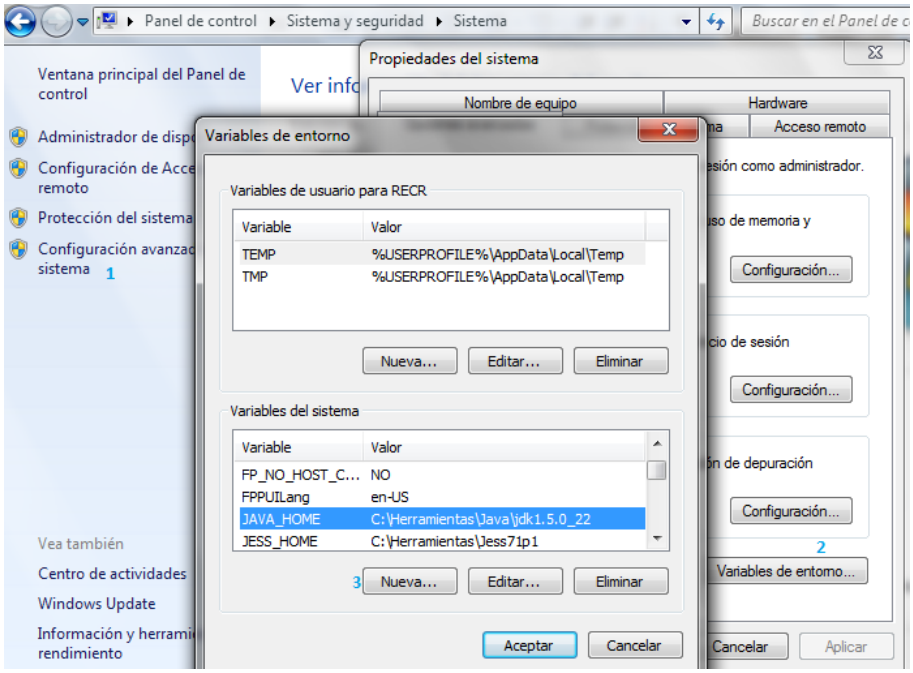


Figura 12. Definición variable JAVA_HOME

Lo siguiente es editar la variable del sistema *Path*, para concatenarle el directorio **%JAVA_HOME%\bin;** . Esto permite ejecutar comandos java por consola desde cualquier directorio o por una aplicación que use Java.

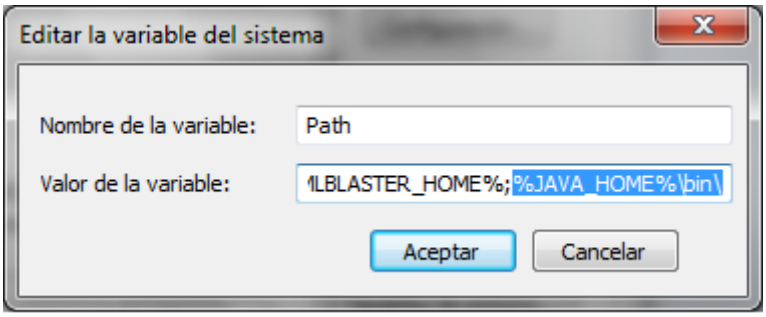


Figura 13. Edición de variable Path

Paso 2, Instalación del IDE Eclipse

Descomprimir el contenido del archivo Eclipse-SDK-3.3-win32.zip en el directorio deseado. Al terminar existirá una nueva carpeta con el nombre 'eclipse', que contiene a su vez otras carpetas y archivos. En este caso, *C:\Herramientas\IDE**eclipse***.

Paso 3, Instalación Framework GEF

Graphical Editing Framework (GEF) proporciona la tecnología para crear vistas y editores gráficos de gran valor.

El archivo GEF-runtime-3.3.zip contiene una carpeta eclipse. Para su instalación, extraerlo en la carpeta superior a la de Eclipse *(C:\Herramientas\IDE\)* y confirmar la fusión del contenido de ambas carpetas eclipse.

Paso 4, Instalación de MySQL

Ejecutar el paquete instalador Mysql-essential-5.0.67-win32.msi. En los pasos de la ventana de Wizard se ha elegido el setup 'Typical', luego como directorio de instalación *C:\Herramientas\BBDD\MySQL\MySQL Server 5.0* y se ha dejado desmarcado el check *Configure the MySQL Server* (se podrá configurar más adelante mediante el instalador de ArchE).

Paso 5, Instalación JESS

El sistema experto JESS (Java Expert System Shell), es un motor de reglas para la plataforma Java y da soporte para desarrollar sistemas expertos basados en reglas que estén totalmente integrados con aplicaciones escritas en el lenguaje Java.

JESS fue desarrollado para poder integrarse con el entorno IDE Eclipse. Para ello:

Descomprimir el archivo Jess71p1.zip, en cualquier directorio (en este caso *C:\Herramientas*). Definir la variable de sistema JESS_HOME, cuyo valor es el directorio de instalación de JESS (*C:\Herramientas\Jess71p1*). Editamos la variable del sistema *Path*, para concatenarle el directorio **%JESS_HOME%\bin;** .

Dentro de la carpeta 'Jess71p1' existe un directorio 'eclipse' con varios archivos comprimidos.

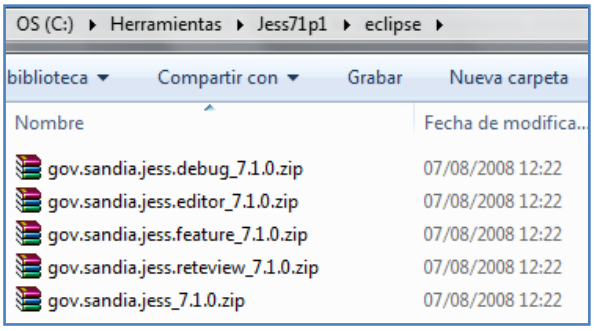


Figura 14. Directorio eclipse de JESS

Al seleccionar todos y descomprimirlos en el mismo directorio, se originan las carpetas 'features' y 'plugins'.

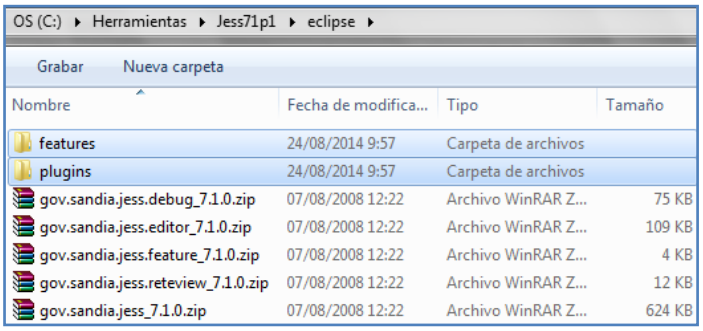


Figura 15. Descomprimir archivos JESS

Trasladamos estas carpetas desde el directorio eclipse de la instalación de JESS al directorio eclipse donde se encuentra instalado el IDE Eclipse (C:\Herramientas\IDE\eclipse\) y confirmamos la fusión de las carpetas 'features' y 'plugins'.

Paso 6, Instalación xmlBlaster

Descomprimir el archivo xmlBlaster_REL_1_6_1.zip, en cualquier directorio (en este caso *C:\Herramientas*). Definir la variable de sistema XMLBLASTER_HOME , cuyo valor es el directorio de instalación de xmlBlaster (*C:\Herramientas\xmlBlaster*).

XmlBlaster debe estar en marcha siempre antes que ArchE. Para lanzar el arranque de xmlBlaster se ha hecho uso del archivo facilitado arranca_xmlblaster.bat .

Paso 7, Instalación de ArchE

Descomprimir el contenido del archivo ArchE_3_0_0_Installer.zip en cualquier directorio. Para iniciar el asistente que nos guiará durante la instalación, se ejecuta el instalador 'ArchE_3_0_0_Installer.exe'.

Comienza con este cuadro de diálogo:

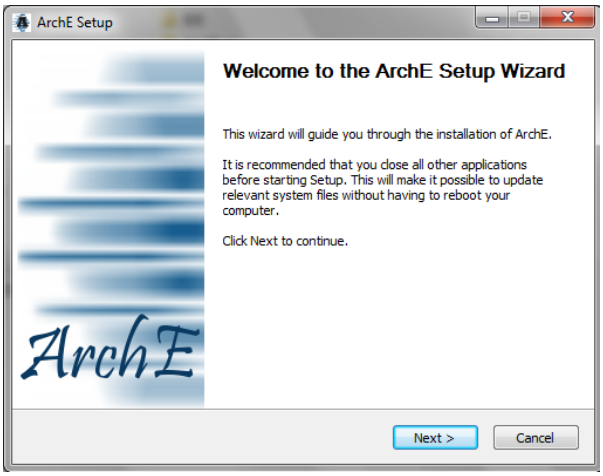


Figura 16. Inicio instalación ArchE

En el siguiente paso *(Next)*, se indica como tipo de instalación: 'Completa'. Después se elige la carpeta donde va a ser instalado ArchE, siendo la misma donde está instalado Eclipse.

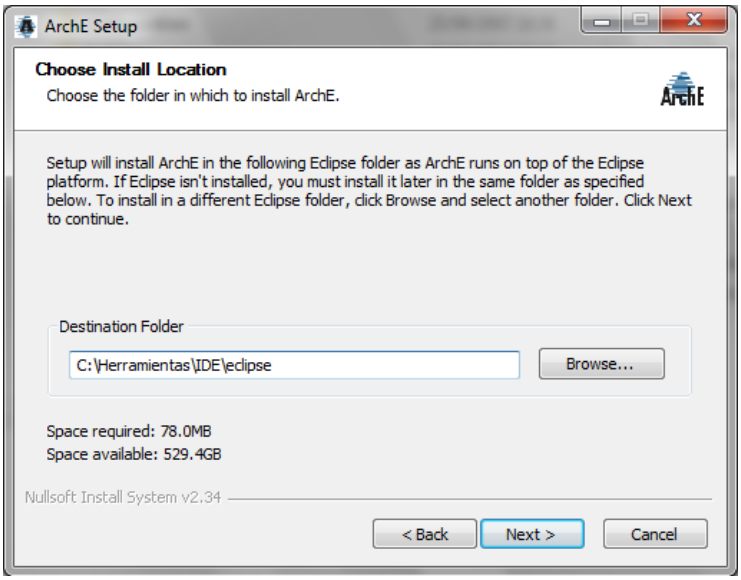


Figura 17. Directorio instalación ArchE

En el siguiente paso el asistente muestra un resumen donde ha verificado si todos los componentes externos usados por ArchE están instalados.

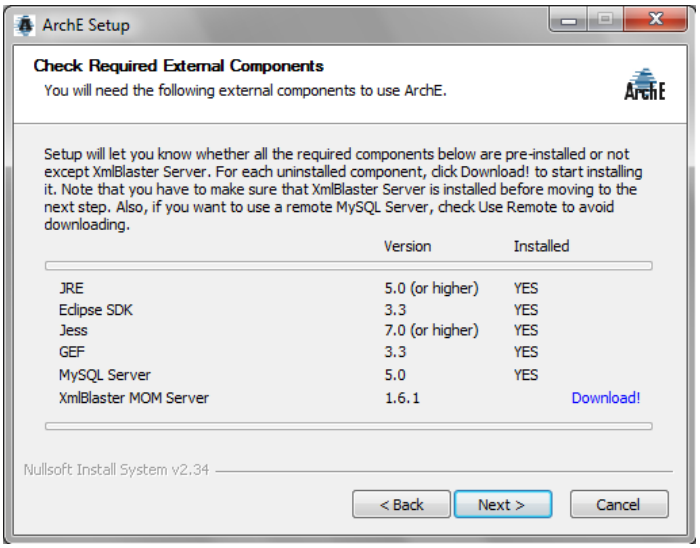


Figura 18. Componentes requeridos por ArchE

Estamos en condiciones de continuar con la instalación. Configuramos en el siguiente cuadro de diálogo el puerto de conexión del servidor MySQL y la password del root:

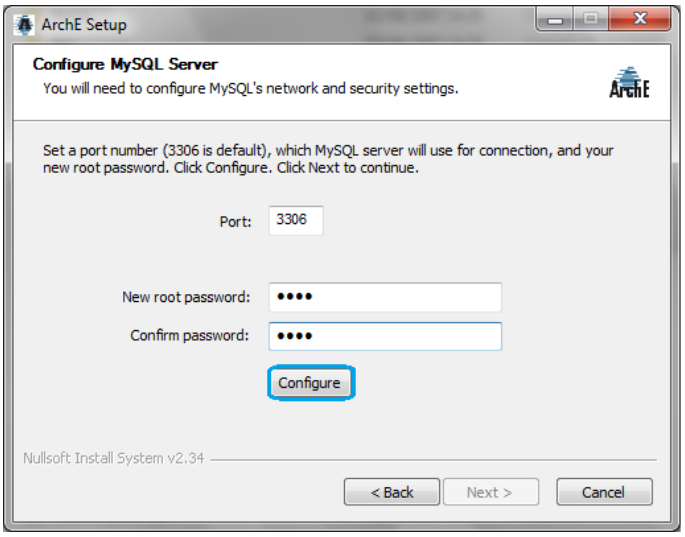


Figura 19. Configuración de MySQL

Ahora se establecerá conexión con el servidor MySQL y se creará la base de datos que utilizará ArchE:

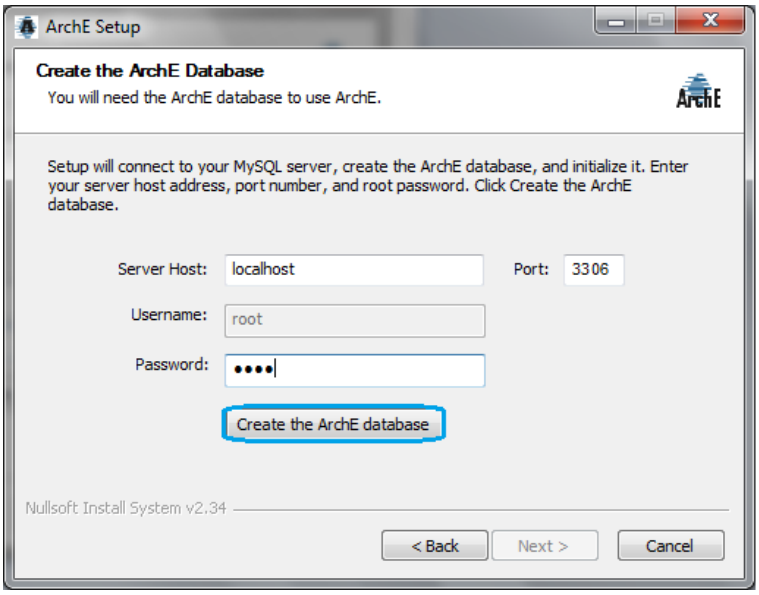


Figura 20. Creación ArchE database

Cuando termine la creación de la base de datos, aparecerá el mensaje 'ArchE database was created successfully' y se habilitará el botón 'Next'.

Por último se configura la conexión con el servidor XmlBlaster para permitir a ArchE interactuar con los marcos de razonamiento:

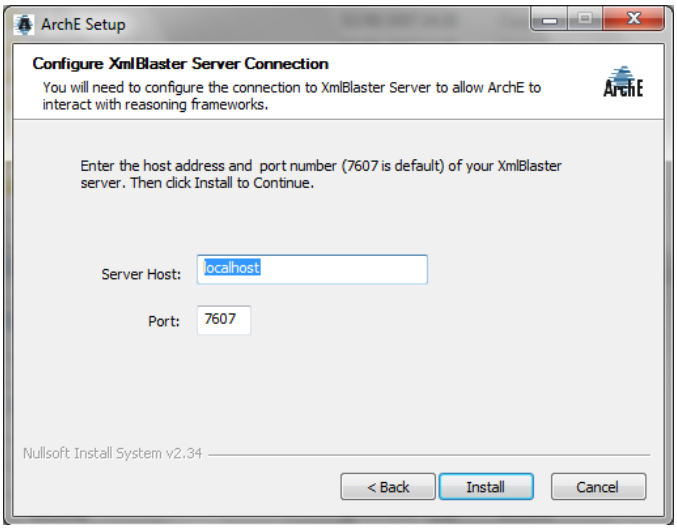


Figura 21. Configuración XmlBlaster

Pulsando el botón Install, comenzará el proceso de instalación de ArchE con las características que se han configurado. Tras unos segundos, el asistente nos informa de que ArchE ha sido instalado.

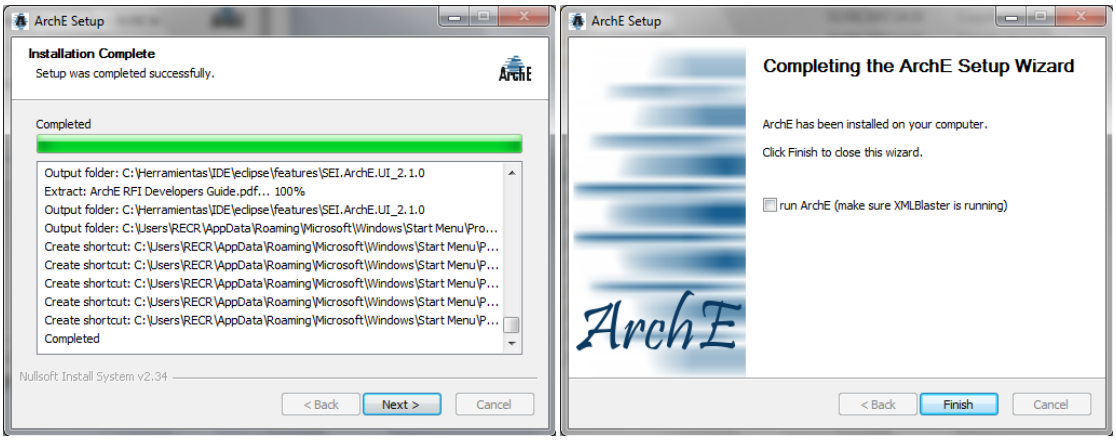


Figura 22. ArchE instalado con éxito

Paso 8, paquete Hibernate Tools

Paquete de librerías Hibernate para Eclipse, HibernateTools-3.2.3.GA.zip. No es estrictamente necesario.

2.6 Puesta en marcha

La primera comprobación para verificar que ArchE se ha integrado correctamente con el entorno de desarrollo Eclipse, es iniciar la herramienta. Podemos hacerlo situándonos en el directorio de instalación de Eclipse y ejecutar el acceso directo con nombre 'ArchE' (el mismo dado durante la configuración): ArchE

En un primer contacto, se puede ver algunos elementos propios del suite ArchE:

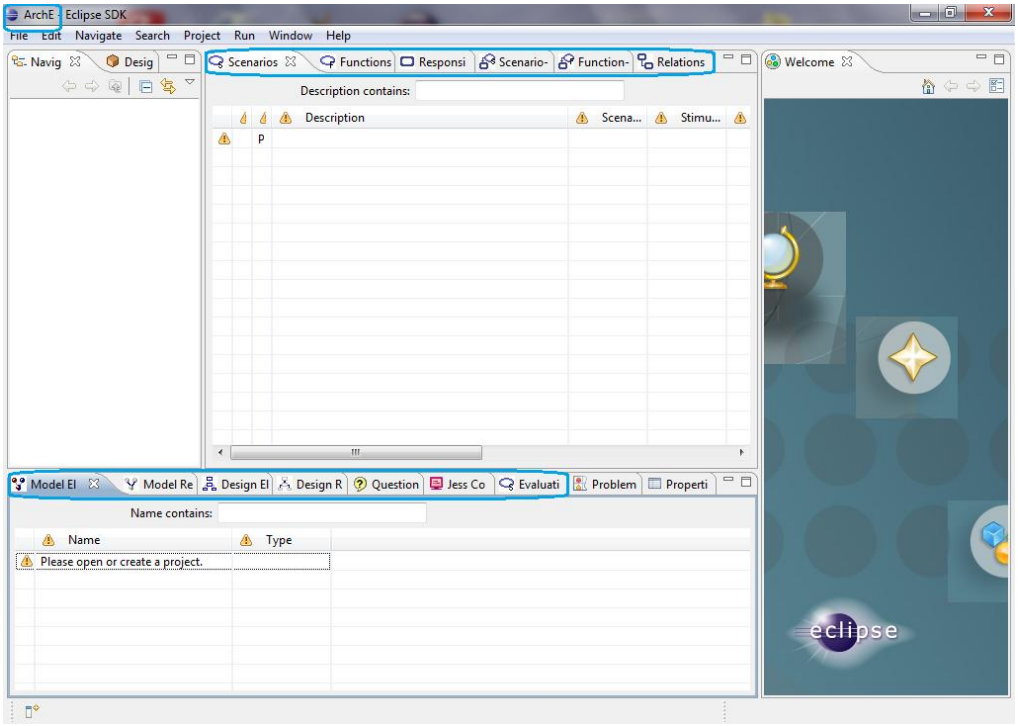


Figura 23. ArchE en Eclipse

Se va a mostrar ahora los pasos para cargar los marcos de razonamiento de los que dispone la herramienta para poder hacer uso de ellos. Es necesario crear previamente un proyecto tipo 'ArchE Project', como se ve en la imagen:

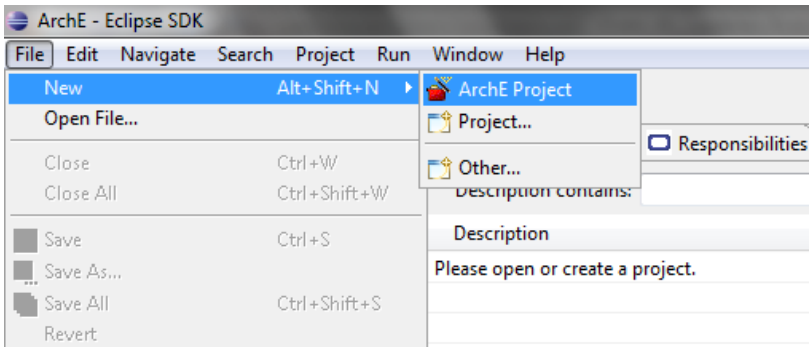


Figura 24. Creación ArchE Project

Mediante la opción de menú *Window-> Show View -> Other...* buscamos los RF Externos.

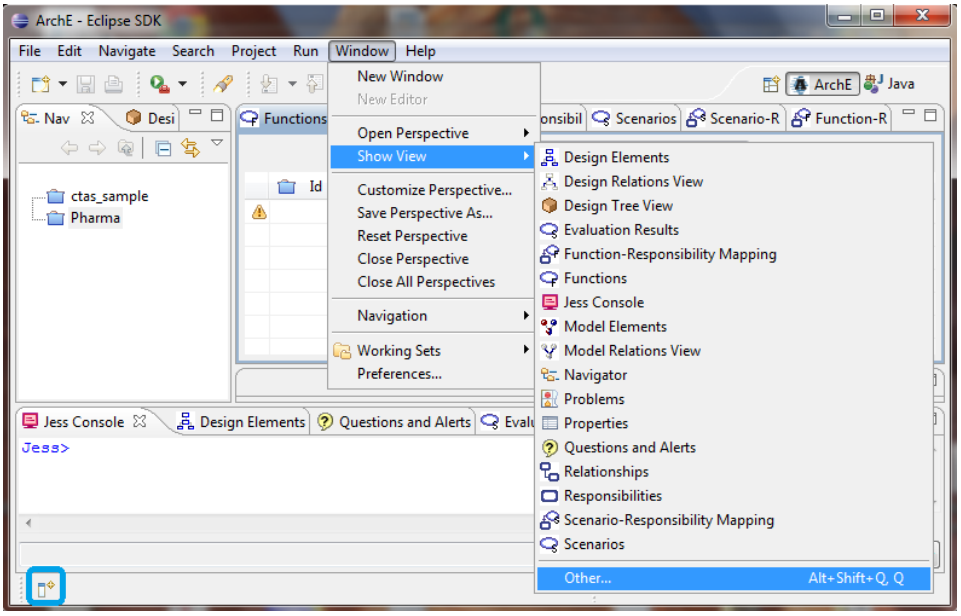


Figura 25. Seleccionar View

Se muestra una ventana donde se puede seleccionar los marcos de razonamiento a cargar: ChangeImpact Modifiability RF (el de modificabilidad) y ICM Performance RF (el de rendimiento).

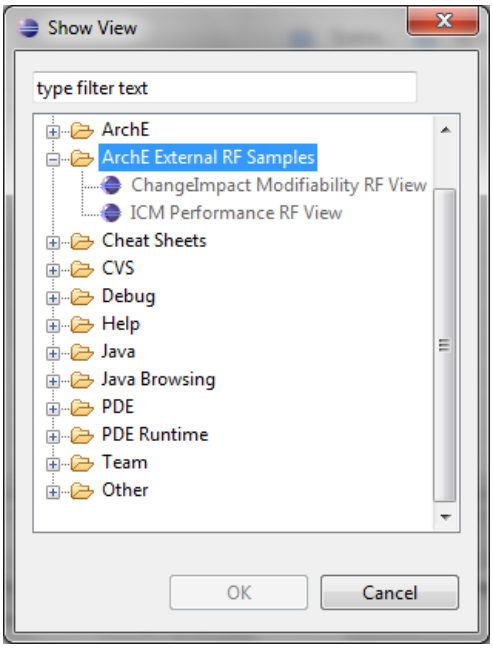


Figura 26. Selección RF

Hay que cargarlos de uno en uno, por ejemplo si seleccionamos el de modificabilidad, se carga el marco de razonamiento, tendremos una vista con la opción de arrancarlo ('Start') y una consola para ver la traza de ejecución del RF.

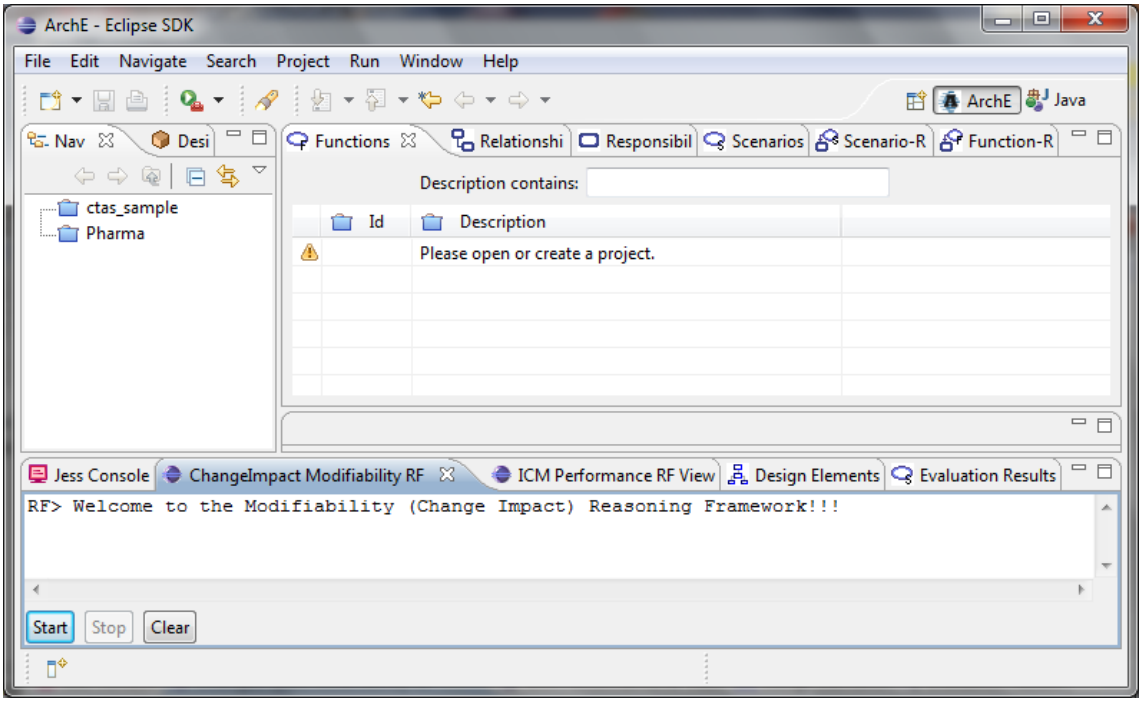


Figura 27. RF cargado

Es importante, antes de iniciar un marco de razonamiento, arrancar el gestor de comunicaciones XmlBlaster, para su buen funcionamiento.

En los casos prácticos expuestos más adelante, se verá con profundidad las distintas opciones que ofrece la herramienta, así como el uso de los marcos de razonamiento disponibles.

3. EVALUACIÓN DE UN SISTEMA SOFTWARE EN ARCHE

3.1 Descripción del caso práctico propuesto: Sistema Pharma

El caso propuesto ha sido extraído del marco general de un sistema de receta electrónica.

Este tipo de sistemas permite a los facultativos de los centros de salud realizar prescripciones electrónicas a un paciente. Posteriormente, el paciente puede retirar los medicamentos que tiene prescritos desde cualquier oficina de farmacia sin necesidad de presentar una receta en papel. Para ello, el/la farmacéutico/a realiza la identificación automática del paciente a través de la tarjeta sanitaria y registra la dispensación de aquellas recetas pendientes de recogida.

Los principales objetivos de un sistema de receta electrónica son:

- Eliminar la receta en papel.
- Mejora la calidad de la atención sanitaria, al reducir el número de visitas a los centros de salud para renovar recetas, aumentando el tiempo disponible de consulta de los médicos.
- Acceso electrónico a las órdenes de prescripción de medicamentos desde cualquier punto de dispensación del país.
- Automatizar los procesos de identificación, prescripción, control y dispensación de medicamentos.

La arquitectura de un sistema de receta electrónica está compuesta por varios subsistemas que se comunican entre sí. Debido a la complejidad de realizar el estudio del caso práctico sobre el sistema completo, se ha decidido centrar el análisis sobre el subsistema de dispensación, al que se le ha denominado 'Sistema Pharma'.

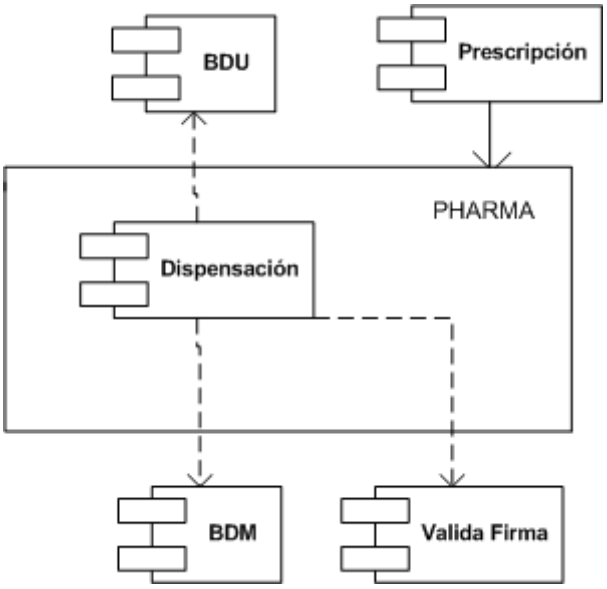


Figura 28. Diagrama subsistemas - Caso práctico

3.2 Casos de uso del sistema

Los requisitos funcionales del sistema Pharma pueden verse representados en los siguientes diagramas de casos de uso:

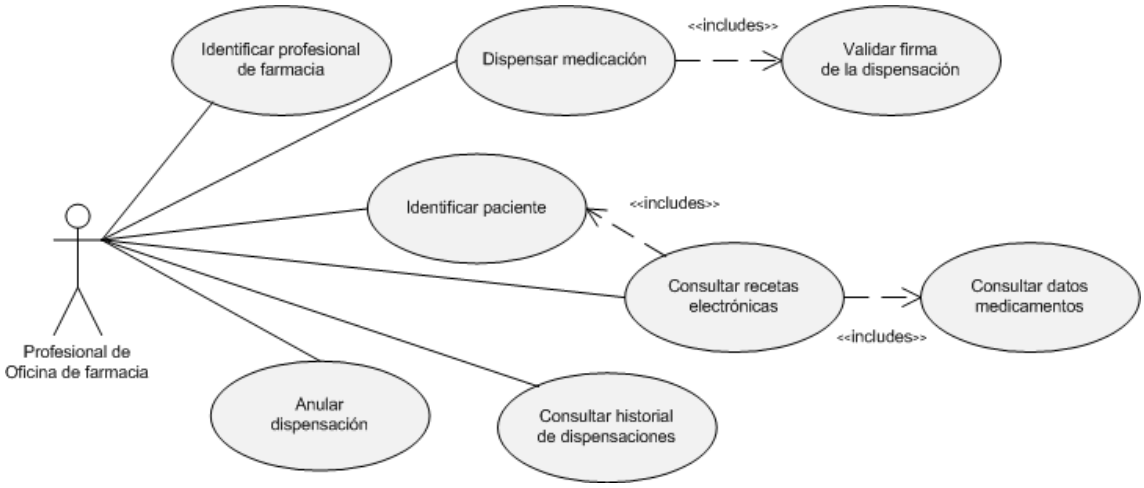


Figura 29. Diagrama CU Pharma - Farmacéutico

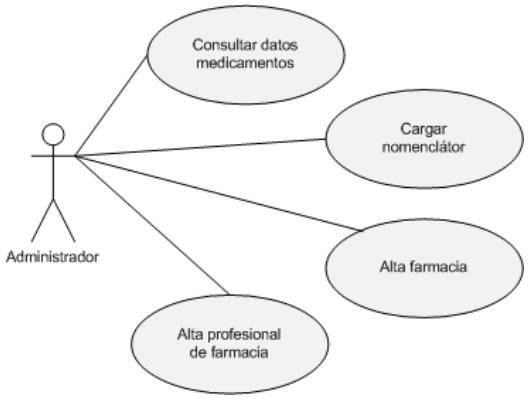


Figura 30. Diagrama CU Pharma - Administrador

Definición de actores que intervienen en el sistema:

- Profesional de Oficina de Farmacia.- El usuario de la aplicación de dispensación.
- Administrador.- Se encarga de las tareas de administración del sistema.

Definición de los casos de uso del sistema Pharma:

Tabla 1. Caso de Uso: Identificar paciente

CU-IPA	**IDENTIFICAR PACIENTE**
Descripción	Este caso de uso hace referencia a la operación por la que el paciente que acude a la oficina de farmacia es identificado en el Módulo de Dispensación para permitir el acceso a su medicación.
Actores	Profesional de Oficina de Farmacia
Precondiciones	Los pacientes que se identifiquen en las oficinas de farmacia deberán

	estar dados de alta en el repositorio de ciudadanos (servicio externo).
Poscondiciones	El paciente es identificado.
Flujo Normal	
1. Este caso de uso se inicia cuando un paciente acude a la oficina de farmacia. 2. El paciente se identificará en la Oficina de Farmacia mediante la lectura de su Tarjeta Sanitaria Individual. 3. Se devuelven los datos requeridos del paciente.	

Tabla 2. Caso de uso: Identificar profesional de farmacia

CU-IPF	**IDENTIFICAR PROFESIONAL DE FARMACIA**
Descripción	Este caso de uso hace referencia a la operación por la que se valida que un profesional de farmacia tiene permitido el acceso al sistema Pharma. Desde la oficina de farmacia se enviará a Pharma las credenciales del profesional de farmacia, y se validará que está dado de alta para el acceso a Pharma.
Actores	Profesional de Oficina de Farmacia
Precondiciones	El profesional de farmacia ha sido dado de alta.
Poscondiciones	El profesional de farmacia es identificado.
Flujo Normal	
1. Este caso de uso se inicia cuando un profesional de farmacia accede al módulo de Dispensación de Pharma. 2. El Módulo de Dispensación solicita los datos de acceso. 3. El profesional de farmacia proporciona credenciales. 4. El Módulo de Dispensación valida en el Módulo de Administración si los datos de acceso son correctos y está permitido el acceso a Pharma.	

Tabla 3. Caso de uso: Consultar recetas electrónicas

CU-CRE	**CONSULTAR RECETAS ELECTRÓNICAS**
Descripción	Este caso de uso hace referencia a la operación por la que el Módulo de Dispensación muestra en la oficina de farmacia los datos de las recetas dispensables del paciente (aquéllas para las se ha alcanzado la fecha de la receta) o que están pendientes de dispensar (fecha de receta a futuro).
Actores	Profesional de Oficina de Farmacia
Precondiciones	El profesional de farmacia debe de ser un usuario autorizado. El paciente ha sido correctamente identificado.
Poscondiciones	Se obtienen las recetas dispensables del paciente.

Flujo Normal
1. Este caso de uso se inicia cuando un profesional de farmacia solicita la consulta de medicación de un paciente. 2. Se obtienen las recetas dispensables del paciente.

Tabla 4. Caso de uso: Dispensar medicación

CU-DIM	**DISPENSAR MEDICACIÓN**
Descripción	Este caso de uso hace referencia a la operación por la que el profesional de farmacia registra la dispensación de uno o más productos que son retirados por un paciente.
Actores	Profesional de Oficina de Farmacia
Precondiciones	El profesional de farmacia debe de ser un usuario autorizado desde el Módulo de Administración. El paciente ha sido correctamente identificado.
Poscondiciones	Las dispensaciones realizadas son almacenadas por el Sistema.
Flujo Normal	
1. Este caso de uso se inicia cuando un profesional de farmacia selecciona los productos dispensables que un paciente va a retirar en la oficina de farmacia. 2. Introduce la información necesaria, (nº envases, la firma, etc). 3. Se almacena en Pharma las dispensaciones correspondientes	

Tabla 5. Caso de uso: Consultar historial de dispensaciones

CU-CHD	**CONSULTAR HISTORIAL DE DISPENSACIONES**
Descripción	Pharma informa a la oficina de farmacia de las dispensaciones realizadas de la medicación recetada.
Actores	Profesional de Oficina de Farmacia
Precondiciones	El profesional de farmacia debe de ser un usuario autorizado.
Poscondiciones	Se obtiene el historial de dispensaciones que sirve como seguimiento del cumplimiento del tratamiento por parte del paciente.
Flujo Normal	
1. El profesional de farmacia accede a la consulta del historial de dispensaciones de un paciente. 2. Pharma devuelve, por cada receta, los datos: fecha, producto y número de envases.	

Tabla 6. Caso de uso: Validar firma de dispensación

CU-VFI	**VALIDAR FIRMA DE DISPENSACIÓN**
Descripción	Este caso de uso hace referencia a la operación por la que se valida la firma de una dispensación generada por el aplicativo de farmacia.
Actores	Profesional de Oficina de Farmacia

Precondiciones	Se ha generado una firma en el aplicativo de farmacia para una dispensación.
Poscondiciones	Se valida la firma.
Flujo Normal	
1. Este caso de uso se inicia durante la ejecución de 'Dispensar medicación'. 2. Pharma validará (invocando, el método de validación de firma) la firma comunicada por las oficinas de farmacia en el momento de la dispensación.	

Tabla 7. Caso de uso: Anular dispensación

CU-AND	**ANULAR DISPENSACIÓN**
Descripción	Este caso de uso hace referencia a la operación por la que los usuarios de una oficina de farmacia pueden anular dispensaciones realizadas desde esa misma oficina de farmacia.
Actores	Profesional de Oficina de Farmacia
Precondiciones	El profesional de farmacia debe de ser un usuario autorizado.
Poscondiciones	Las dispensaciones quedan anuladas.
Flujo Normal	
1. Este caso de uso se inicia cuando un profesional de farmacia accede a la funcionalidad para anular dispensaciones. 2. El profesional de farmacia selecciona las dispensaciones que desea anular. 3. Las dispensaciones quedan anuladas en el sistema.	
Excepciones	
Si se supera el límite máximo de tiempo (configurado en base de datos) que, tras la dispensación, ésta puede ser eliminada, no podrá llevarse a cabo la anulación solicitada por el usuario de farmacia.	

Tabla 8. Caso de uso: Consultar datos medicamentos

CU--CDM	**Consulta Datos Medicamentos**
Descripción	Este caso de uso abarca la funcionalidad de consultar los datos de los productos en la Base de Datos de Medicamentos.
Actores	• Profesional de Oficina de Farmacia • Operador Administrador
Precondiciones	Operador autorizado con permiso para consultar.
Poscondiciones	Se devuelve datos.
Flujo Normal	
1. Se solicita consultar los datos de un producto. 2. Se devuelve información sobre el producto consultado.	

Tabla 9. Caso de uso: Cargar Nomenclátor

CU-CAN	**Cargar Nomenclátor**
Descripción	La carga del Nomenclátor se basa en la actualización de los datos almacenados en la Base de Datos de Medicamentos a partir de una fuente de información (ficheros).
Actores	Operador Administrador
Precondiciones	Operador autorizado con permiso para consultar y actualizar datos.
Poscondiciones	No aplica.
Flujo Normal	
1. El operador selecciona los ficheros de datos de medicamentos necesarios para la carga. 2. Se realizarán comprobaciones sobre la estructura contenida de los ficheros de carga, para comprobar su validez. 3. Si todo es correcto, el operador podrá comenzar a realizar la carga del Nomenclátor a partir de los ficheros de datos seleccionados.	

Tabla 10. Caso de uso: Gestionar Alta Usuario de Oficina de Farmacia

CU-GAU	**GESTIONAR ALTA USUARIO DE OFICINA DE FARMACIA**
Descripción	Este caso de uso permite la gestión de los usuarios de oficinas de farmacia.
Actores	Operador Administrador
Precondiciones	El usuario debe de ser un operador autorizado.
Poscondiciones	No aplica.
Flujo Normal	
1. El operador realiza la gestión (alta, baja o modificación) de los usuarios de oficinas de farmacia.	

Tabla 11. Caso de uso: Gestionar Alta de Oficina de Farmacia

CU-GAF	**GESTIONAR ALTA FARMACIA**
Descripción	Este caso de uso permite gestionar el alta de una oficina de farmacia.
Actores	Operador Administrador
Precondiciones	Para poder acceder a esta funcionalidad, el usuario debe de ser un operador autorizado.
Poscondiciones	No aplica.
Flujo Normal	
1. El operador realiza el alta de oficina de farmacia.	

3.3 Especificación de requisitos en ArchE

Para que la herramienta ArchE pueda evaluar la arquitectura de un sistema software y asistir en el diseño al arquitecto, es necesario especificar mediante la propia herramienta, los requisitos funcionales y los requisitos de atributos de calidad que el sistema debe cumplir.

Previamente a la ejecución de ArchE, ha de iniciarse la base de datos MySQL y el gestor XmlBlaster, para su buen funcionamiento. Realizaremos la especificación sobre un proyecto 'ArchE Project' (es necesario crearlo si no existe).

El **primer paso** consiste en definir el conjunto de funcionalidades básicas del sistema desde la vista 'Functions'. El diagrama de casos de uso, antes expuesto, ha servido para identificar las funciones.

Situando el cursor sobre cualquiera de las líneas y haciendo clic en el botón derecho del ratón, nos aparece un menú contextual con la opción de creación 'New function'.

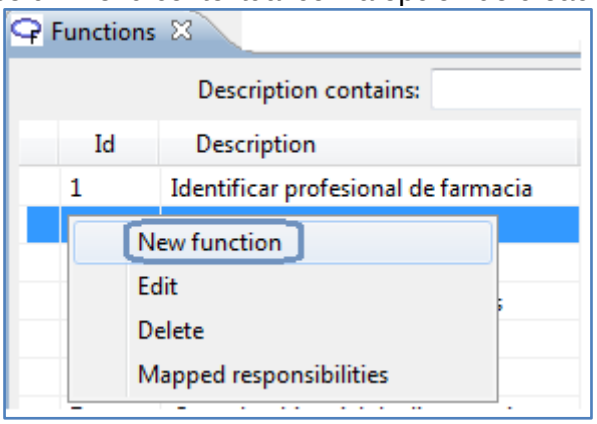


Figura 31. Definición de funciones en ArchE

Cada función queda definida por un identificador numérico y una descripción:

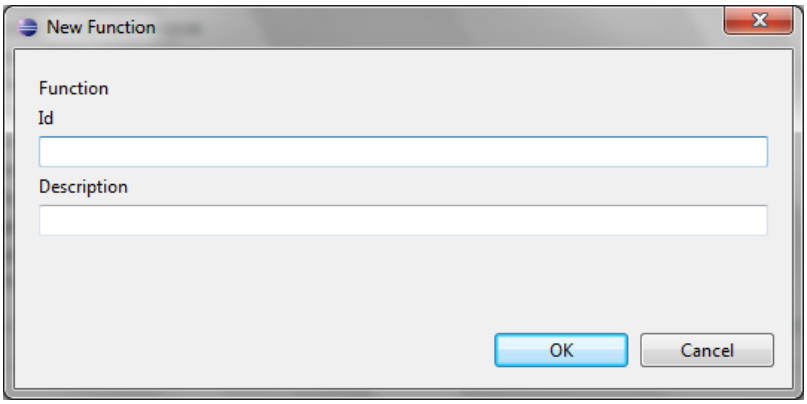


Figura 32. Ventana New Function

La lista de funciones creadas en el proyecto ArchE para el sistema 'Pharma' puede verse en la siguiente figura:

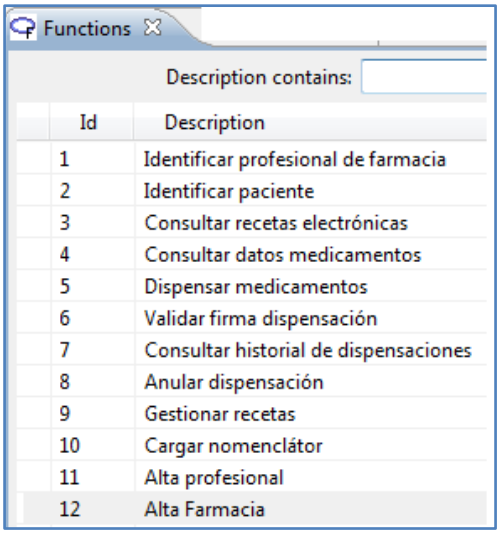


Figura 33. Funciones del sistema Pharma

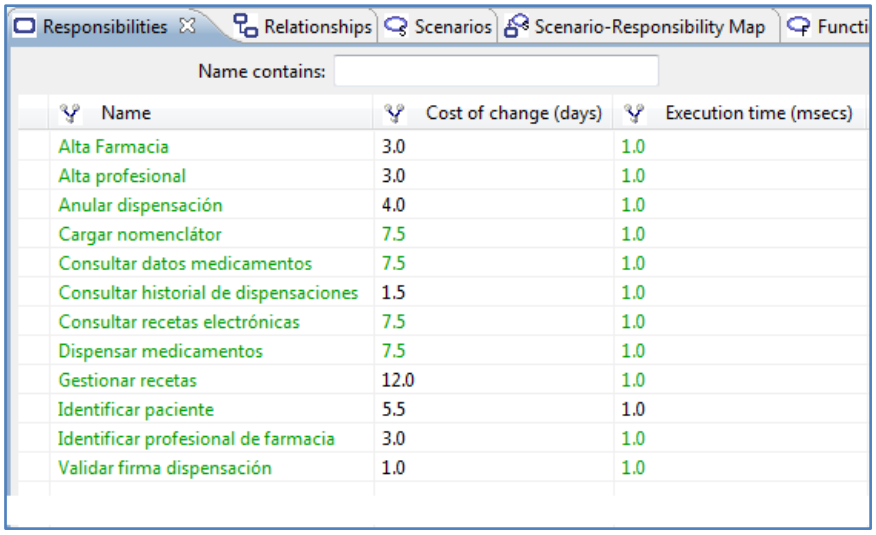
Es importante, si no queremos perder los datos que hayamos introducido en el proyecto, persistir los cambios con la opción de la herramienta Persist fact base. Esta opción puede ser seleccionada desde la pestaña 'Project' o desde la vista 'Navigator' pulsando botón derecho.

En este momento es preciso iniciar los dos marcos de razonamiento disponibles en ArchE: Modificabilidad (ChangeImpact Modifiability RF) y Rendimiento (ICM Performance RF). Se van a utilizar en los pasos siguientes.

El **segundo paso** consiste en definir las responsabilidades.

Una responsabilidad es una actividad llevada a cabo por el sistema que está siendo diseñado. Inicialmente ArchE crea automáticamente una responsabilidad por cada función, hecho reflejado en la vista 'Responsibilities'. A medida que el proceso de diseño avanza, nuevas responsabilidades pueden ser añadidas, separadas o modificadas como resultado de aplicar tácticas para alcanzar escenarios concretos de atributos de calidad. ArchE usa las responsabilidades como medio de expresión de requerimientos funcionales.

Las responsabilidades, a diferencia de las funciones, poseen propiedades relativas a los marco de razonamiento cargados en la herramienta. Como se puede ver en la figura, las responsabilidades poseen valores para los atributos del costo del cambio (modificabilidad) y tiempo de ejecución (rendimiento).

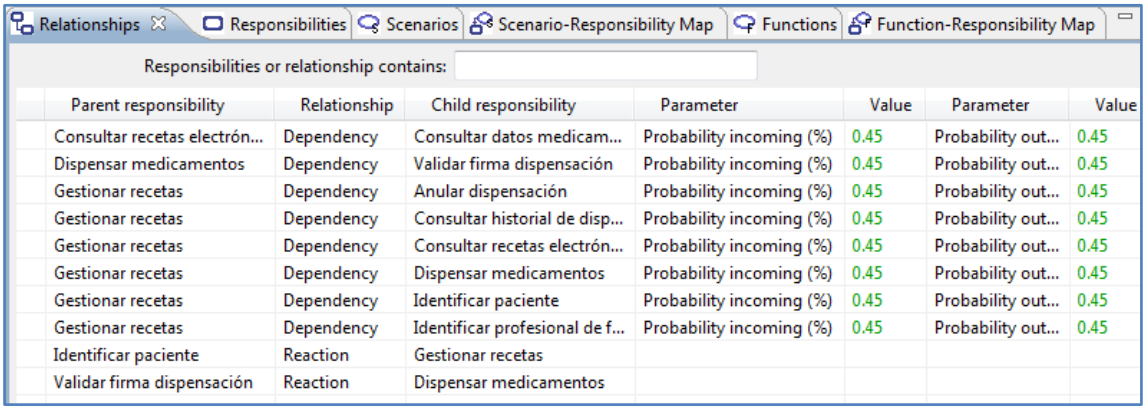


Name	Cost of change (days)	Execution time (msecs)
Alta Farmacia	3.0	1.0
Alta profesional	3.0	1.0
Anular dispensación	4.0	1.0
Cargar nomenclátor	7.5	1.0
Consultar datos medicamentos	7.5	1.0
Consultar historial de dispensaciones	1.5	1.0
Consultar recetas electrónicas	7.5	1.0
Dispensar medicamentos	7.5	1.0
Gestionar recetas	12.0	1.0
Identificar paciente	5.5	1.0
Identificar profesional de farmacia	3.0	1.0
Validar firma dispensación	1.0	1.0

Figura 34. Vista inicial de Responsabilidades

ArchE representa en color verde aquello que es generado de forma automática y en color negro lo personalizado por el usuario.

El **tercer paso** consiste en introducir las relaciones entre las responsabilidades identificadas. Tipo de relaciones disponibles: *Dependency* (modificabilidad), *Reaction* (rendimiento).



Parent responsibility	Relationship	Child responsibility	Parameter	Value	Parameter	Value
Consultar recetas electrón...	Dependency	Consultar datos medicam...	Probability incoming (%)	0.45	Probability out...	0.45
Dispensar medicamentos	Dependency	Validar firma dispensación	Probability incoming (%)	0.45	Probability out...	0.45
Gestionar recetas	Dependency	Anular dispensación	Probability incoming (%)	0.45	Probability out...	0.45
Gestionar recetas	Dependency	Consultar historial de disp...	Probability incoming (%)	0.45	Probability out...	0.45
Gestionar recetas	Dependency	Consultar recetas electrón...	Probability incoming (%)	0.45	Probability out...	0.45
Gestionar recetas	Dependency	Dispensar medicamentos	Probability incoming (%)	0.45	Probability out...	0.45
Gestionar recetas	Dependency	Identificar paciente	Probability incoming (%)	0.45	Probability out...	0.45
Gestionar recetas	Dependency	Identificar profesional de f...	Probability incoming (%)	0.45	Probability out...	0.45
Identificar paciente	Reaction	Gestionar recetas				
Validar firma dispensación	Reaction	Dispensar medicamentos				

Figura 35. Vista inicial Relationships

El **cuarto paso** consiste en expresar los requisitos de los atributos de calidad del sistema como escenarios. Cada escenario concreto de un atributo de calidad es descrito en seis partes: el estímulo, la fuente del estímulo, el entorno, artefacto que recibirá el estímulo, la respuesta y la medida de la misma, que será la que se utilizará para evaluar la arquitectura y ver si cumple o no el requisito

Desde la vista 'Scenarios', seleccionando la opción del menú contextual 'New scenario', nos permite crear escenarios introduciendo su descripción, valores para cada una de sus partes e indicando el tipo de escenario de atributo de calidad que va a ser evaluado por el marco de razonamiento correspondiente.

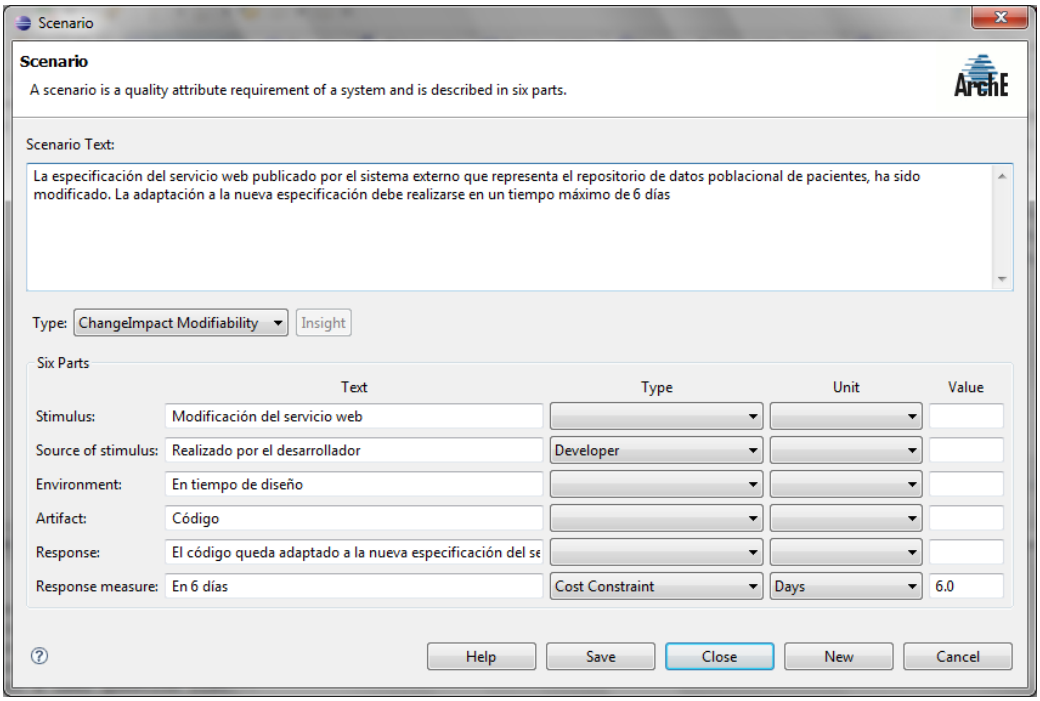


Figura 36. Creación de un escenario de Modificabilidad

Se han definido los siguientes escenarios de los atributos de calidad de modificabilidad y rendimiento (ICM Performance):

- MS1 (Modifiability Scenario 1)

La especificación del servicio web publicado por el sistema externo que representa el repositorio de datos poblacional de pacientes, ha sido modificado. La adaptación a la nueva especificación debe realizarse en un tiempo máximo de 6 días.

Tabla 12. Descripción de Modifiability Scenario 1

	Texto	Tipo	Unidad	Valor
Origen del estímulo	Realizado por un desarrollador	Developer ▼		
Estímulo	Modificación del servicio web			
Artefacto	Código			
Entorno	En tiempo de diseño			
Respuesta	El código queda adaptado a la nueva especificación del servicio web			
Medida de la respuesta	En 6 días	Coste	Días	6

- MS2 (Modifiability Scenario 2)

Se incluye una nueva funcionalidad, con el fin de que el sistema contemple distintos perfiles para los profesionales de las oficinas de farmacia.

Tabla 13. Descripción de Modifiability Scenario 2

	Texto	Tipo	Unidad	Valor
Origen del estímulo	Cambio realizado por un desarrollador	Developer ▼		
Estímulo	Añadir perfiles a los profesional de farmacia			
Artefacto	Código			
Entorno	En tiempo de diseño			
Respuesta	El sistema contempla distintos perfiles para los profesionales de farmacia			
Medida de la respuesta	En 7 días	Coste	Días	7

- MS3 (Modifiability Scenario 3)

El formato de la firma de una dispensación realizada por los farmacéuticos mediante su certificado digital ha cambiado. El método de validación de firma ha de contemplar el nuevo formato en un plazo máximo de 6 días.

Tabla 14. Descripción de Modifiability Scenario 3

	Texto	Tipo	Unidad	Valor
Origen del estímulo	Cambio realizado por el desarrollador	Developer ▼		
Estímulo	Adaptación del método de validación de firma			
Artefacto	Componente de validación			
Entorno	En tiempo de diseño			
Respuesta	Validación de la firma recibida atendiendo al nuevo formato			

Medida de la respuesta	En 6 días	Coste	Días	6

- PS1 (Performance Scenario 1)

Cuando un usuario de una oficina de farmacia introduce su tarjeta de profesional en el lector para identificarse, sus datos deben ser leídos en un tiempo medio de 4.5 segundos.

Tabla 15. Descripción de Performance Scenario 1

	Texto	Tipo	Unidad	Valor
Origen del estímulo	Usuario de farmacia	Externo ▼		
Estímulo	Leer chip tarjeta del farmacéutico	Periódico ▼	Segundos	4.5
Artefacto	Módulo lector del sistema	Módulo ▼		
Entorno	Modo de operación normal	Condición normal ▼		
Respuesta	La información del chip de la tarjeta es leída	Tarea		
Medida de la respuesta	La media de la respuesta en torno a 4.5 segundos	Media ▼	Segundos	4.5

- PS2 (Performance Scenario 2)

El farmacéutico solicita ver el historial de dispensaciones de un paciente y éste se le muestra, en condiciones normales, en un tiempo máximo de 6 segundos.

Tabla 16. Descripción de Performance Scenario 2

	Texto	Tipo	Unidad	Valor
Origen del estímulo	Usuario de farmacia	Externo ▼		
Estímulo	Mostrar historial de dispensación	Periódico ▼	Segundos	6
Artefacto	Componente de dispensación	Módulo ▼		
Entorno	Modo de operación normal	Condición normal ▼		

Respuesta	Se devuelve el historial de dispensación	Tarea		
Medida de la respuesta	Como máximo 6 segundos	Caso peor ▼	Segundos	6

El **último paso** de la especificación consiste en, establecer las relaciones entre las responsabilidades (requisitos funcionales) y los escenarios (requisitos de atributos de calidad). El mapeo escenarios-responsabilidades se realiza desde la vista 'Scenario-Responsability Mapping'. Para crear una relación, se elige la opción 'New Mapping' del menú contextual que aparece cuando pulsamos botón derecho, y se selecciona el escenario y la responsabilidad deseados.

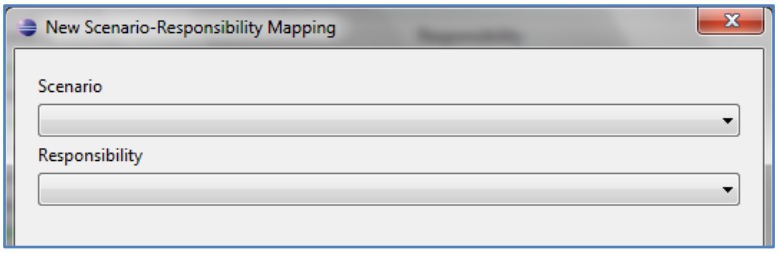


Figura 37. Creación relación Escenario-Responsabilidad

ArchE evaluará el escenario teniendo en cuenta sólo las responsabilidades asociadas al mismo.

Todas las relaciones creadas entre las responsabilidades y los escenarios del sistema 'Pharma' se muestran en la figura siguiente:

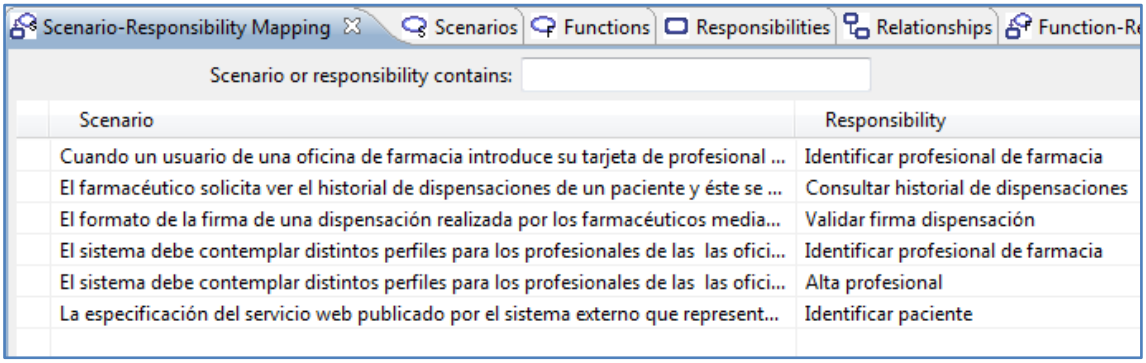


Figura 38. Mapeo de Escenarios-Responsabilidades

3.4 Evaluación realizada por ArchE y aplicación de las tácticas sugeridas

Mientras hemos estado introduciendo los escenarios y su relación con las responsabilidades, los marcos de razonamiento de los atributos de modificabilidad y rendimiento, así como el sistema experto Jess, han estado en ejecución analizando los datos. Cuando demos por finalizada la tarea de especificación, es el momento de considerar los resultados de la evaluación devueltos por ArchE.

A partir de la información facilitada, ArchE genera el diseño del modelo:

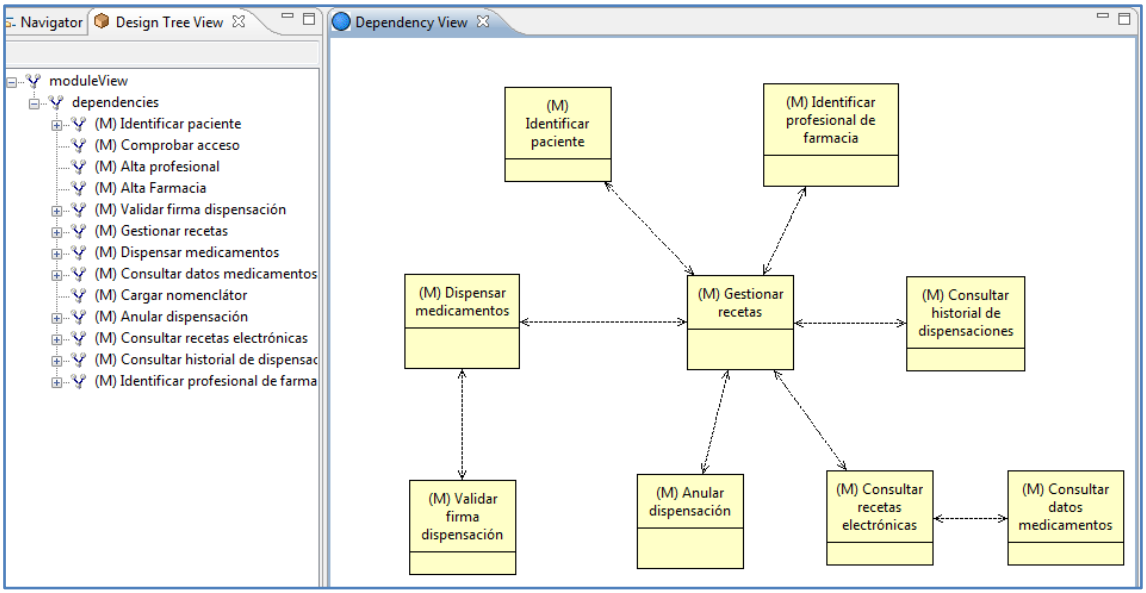


Figura 39. Vista de dependencias del diseño inicial

Desde la vista 'Scenarios' podemos comprobar si los escenarios que se han definido en base a los atributos de modificabilidad y rendimiento son o no satisfechos, es decir, si el diseño cumple con los requisitos de calidad exigidos. En el caso del sistema 'Pharma' esta es la evaluación inicial de los escenarios que se obtiene:

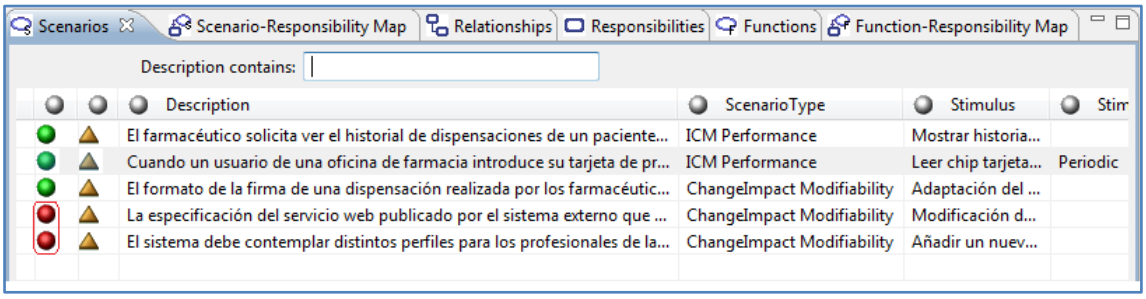


Figura 40. Vista Escenarios tras evaluación inicial

La primera columna de la izquierda representa, mediante esferas, el nivel de satisfacción del escenario. Si es de color verde, significa que el escenario es satisfecho. Si es de color gris, significa que el escenario no ha sido evaluado debido a que faltan aún parámetros por completarse o el framework de razonamiento correspondiente al atributo del escenario, no está en ejecución. Si es de color rojo, significa que el escenario no es satisfecho.

La segunda columna de la izquierda representa, mediante triángulos, si el último cambio realizado en el diseño tiene un impacto positivo o negativo sobre el escenario. Si es de color verde, significa que el último cambio tiene un efecto positivo. Si es de color amarillo, significa que el cambio no afecta al escenario. Si es de color rojo, significa que el último cambio tiene un impacto negativo.

Por tanto, de los seis escenarios definidos hay dos que no han sido satisfechos.

En la vista 'Questions and Alerts' puede verse un listado de alertas y preguntas dirigidas al usuario ordenadas por prioridad. Su función es sugerir la aplicación de tácticas arquitectónicas para mejorar el diseño actual.

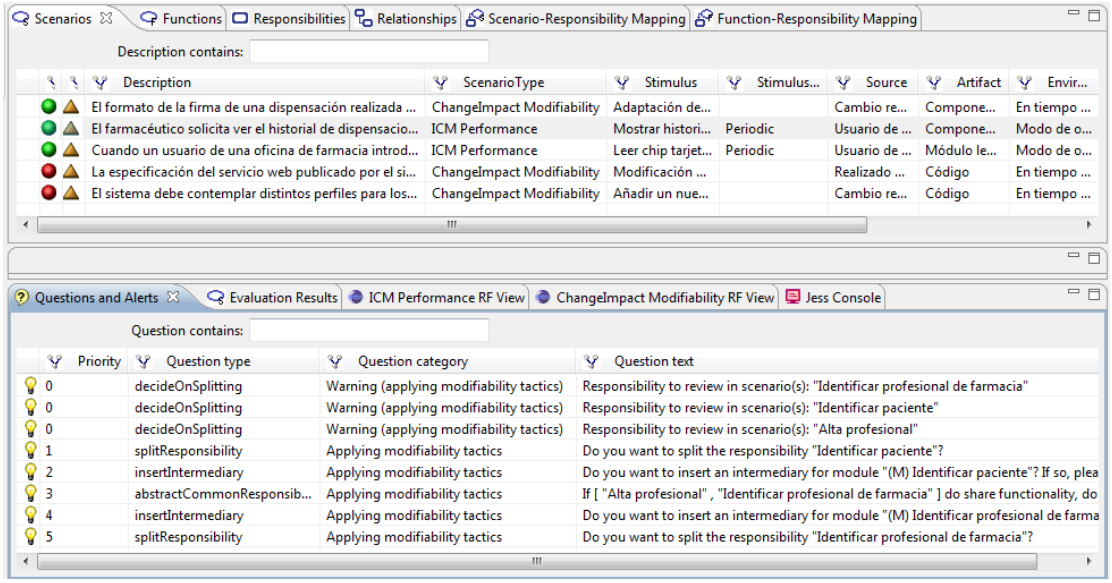


Figura 41. Vista inicial Alertas y Preguntas

El tipo de táctica propuesta a aplicar, deriva del atributo de calidad del escenario afectado. En este caso, todas las cuestiones están relacionadas con la aplicación de tácticas de modificabilidad, propuestas por el marco de razonamiento de modificabilidad.

Es importante aclarar que el marco de razonamiento de rendimiento desarrollado por el SEI no propone tácticas, simplemente determina si el sistema cumple o no con los requisitos de rendimiento exigidos.

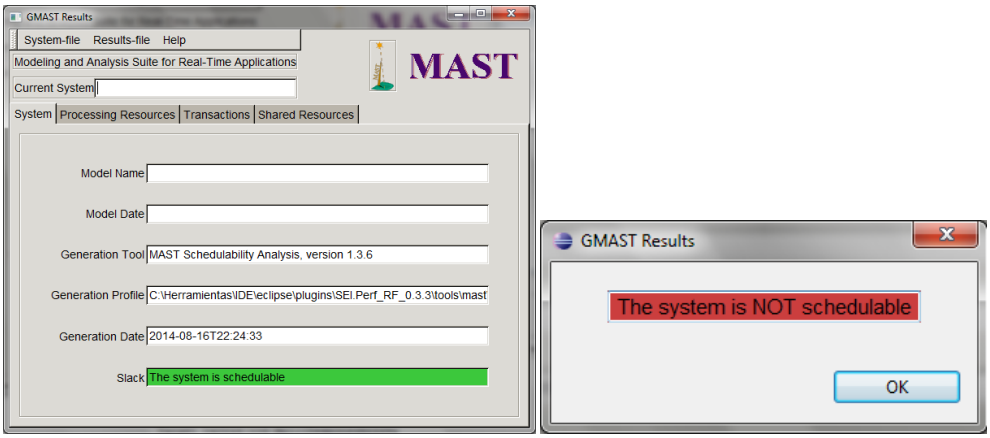


Figura 42. GMAST Results

Las cuestiones codificadas con prioridad 0, son advertencias sobre la aplicación de una táctica. Su utilidad es meramente informativa. A modo de ejemplo, si pulsamos dos veces sobre una cuestión se nos abre una ventana de este tipo:

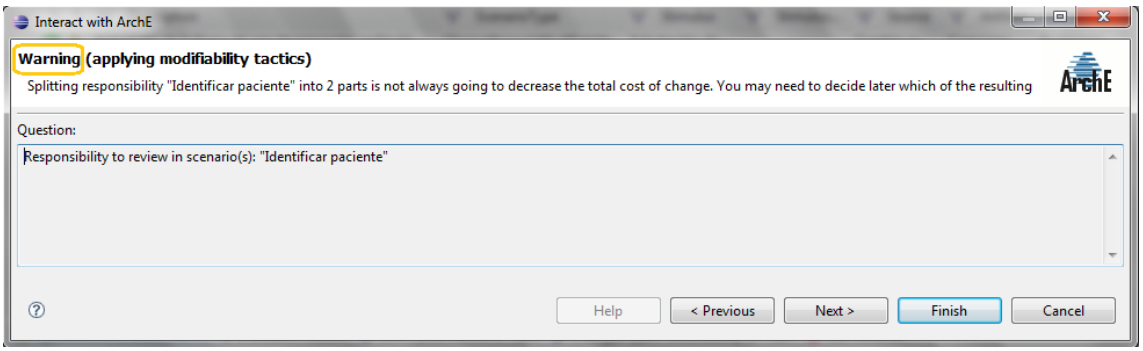


Figura 43. Advertencia (al aplicar tácticas de modificabilidad)

Otra vista de la herramienta muy práctica, es la vista 'Evaluation Results' que resume la utilidad que tendrá la aplicación de cada táctica sobre cada uno de los escenarios.

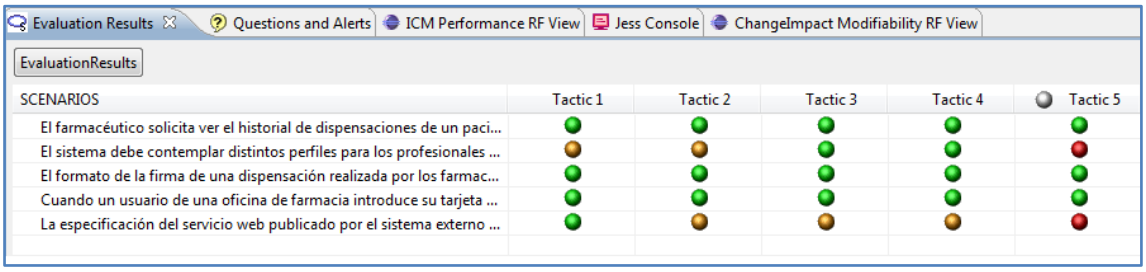


Figura 44. Vista inicial Evaluation Results

Una esfera de color verde indica que, si se aplica la táctica, el escenario se va a satisfacer. De color amarillo indica que al aplicarla tiene cierta utilidad. De color rojo, indica que al aplicarla no tiene utilidad sobre el escenario.

Lo primero que se puede deducir de la evaluación de resultados, es que la táctica 5 no mejora los escenarios que pretendemos satisfacer, con lo cual se descarta su aplicación y a su vez, se ahorra tiempo en el análisis.

Se observa que de las tácticas sugeridas, las dos más prioritarias afectan al mismo escenario, con lo cual aplicaremos en primer lugar, una de las dos. Se detallan en la siguiente tabla:

Tabla 17. Tácticas sugeridas inicialmente

Prioridad	Tipo	Categoría	Descripción
1	Split Responsability	Modificab.	La responsabilidad "Identificar paciente" tiene fuertes dependencias con otras responsabilidades. Por tanto, podría ser una buena idea dividirla en dos responsabilidades hijas para minimizar las dependencias. Una estimación sugiere que podría

			reducirse el costo del escenario de "7.319615384615384" a "5.835375" persona/día.
2	Insert Intermediary	Modificab.	Dependencias desde responsabilidades al módulo "(M) Identificar paciente", tienen un impacto sobre un escenario. Puede resultar beneficioso insertar un módulo intermediario para que los cambios sean menos costosos. Un intermediario de un coste de "15.500000000000004" persona/día podría reducir el coste de cambiar el escenario de "7.319615384615384" persona/día a "6.455946428571429" persona/día.

La primera táctica sugiere dividir la responsabilidad "Identificar paciente" en otras, funcionalmente para el sistema real carece de sentido puesto que en el proceso de identificación de paciente se va a llamar un servicio web externo, siempre de la misma forma.

La segunda táctica propone un intermediario, funcionalmente es aplicable porque permite desacoplar al sistema del servicio web cliente, de forma que si el cliente cambia, no hace falta cambiar el sistema o el cambio es mínimo. Aplicamos, por tanto, la 2ª táctica indicando el coste para el intermediario:

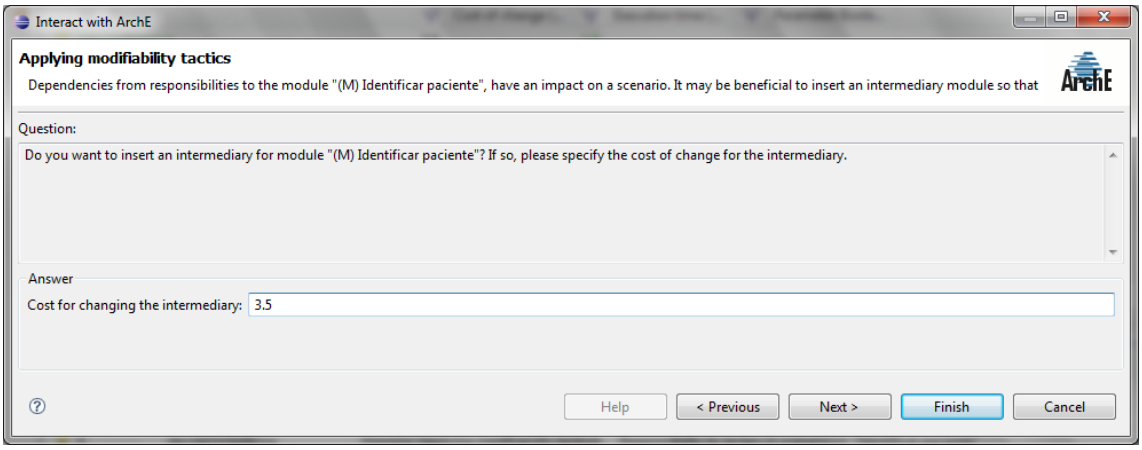


Figura 45. Cuestión para insertar intermediario

Regresando a la vista de escenarios podemos ver el efecto que ha producido la aplicación de la táctica:

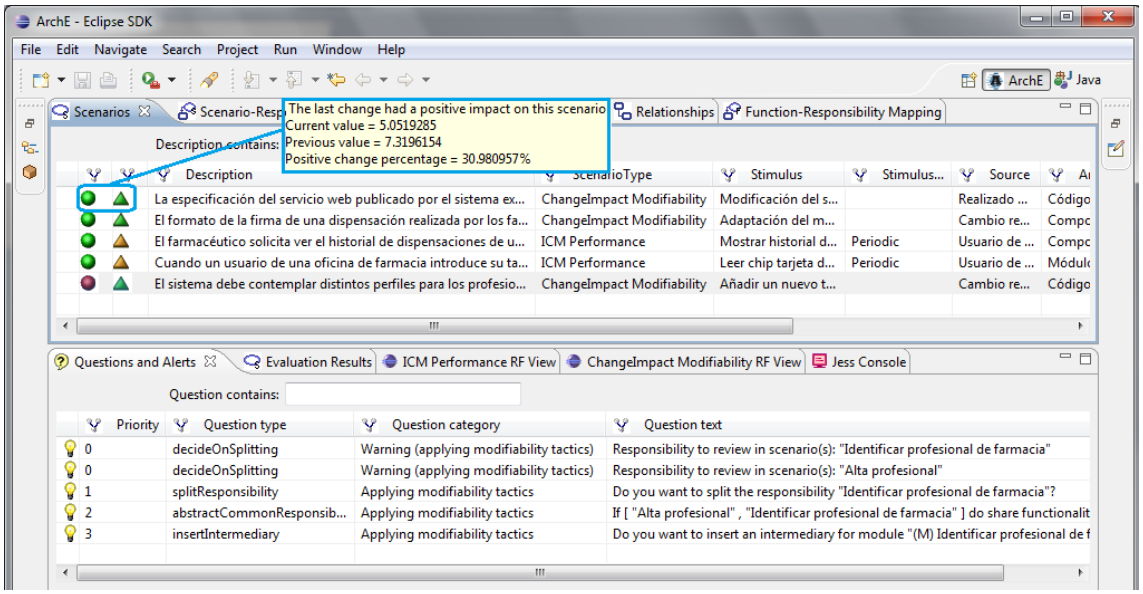


Figura 46. Escenarios tras insertar intermediario

Como se aprecia en la figura anterior, la aplicación de la táctica supone que el escenario de modificabilidad sea satisfecho y una mejora de un 30 %, respecto al diseño anterior. Este es el nuevo diseño propuesto por ArchE para el sistema 'Pharma':

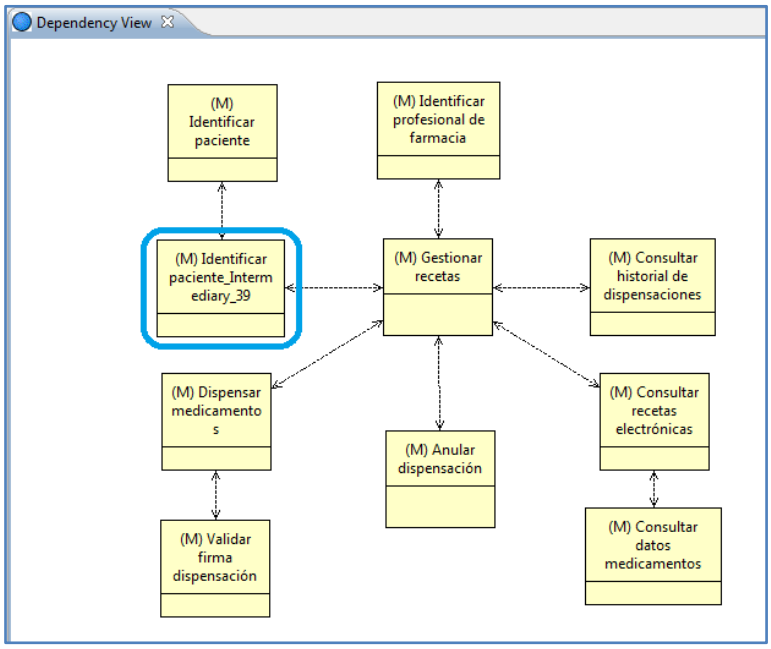


Figura 47. Vista de dependencias - Intermediario

Aún nos queda mejorar un escenario para cumplir con todos los requisitos de calidad requeridos. Volvemos a la vista 'Questions and Alerts' para revisar las tácticas sugeridas

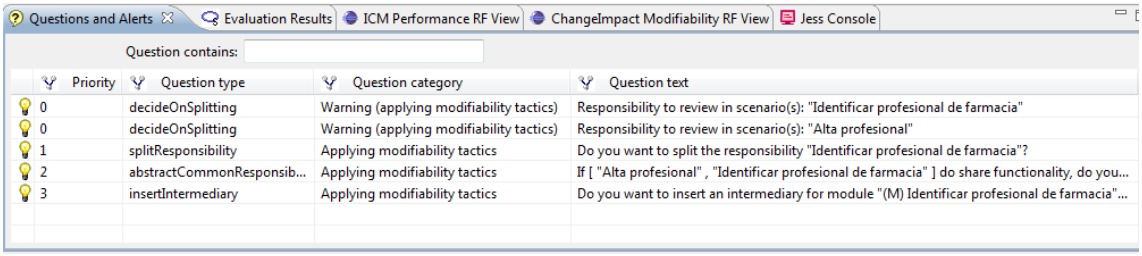


Figura 48. Vista Q&A tras Iteración I

La herramienta propone tres tácticas. En la vista 'Evaluation Results' se advierte que cualquier táctica que se aplique, va a hacer que se satisfagan todos los escenarios. Se determinará entonces cuál es la más adecuada.

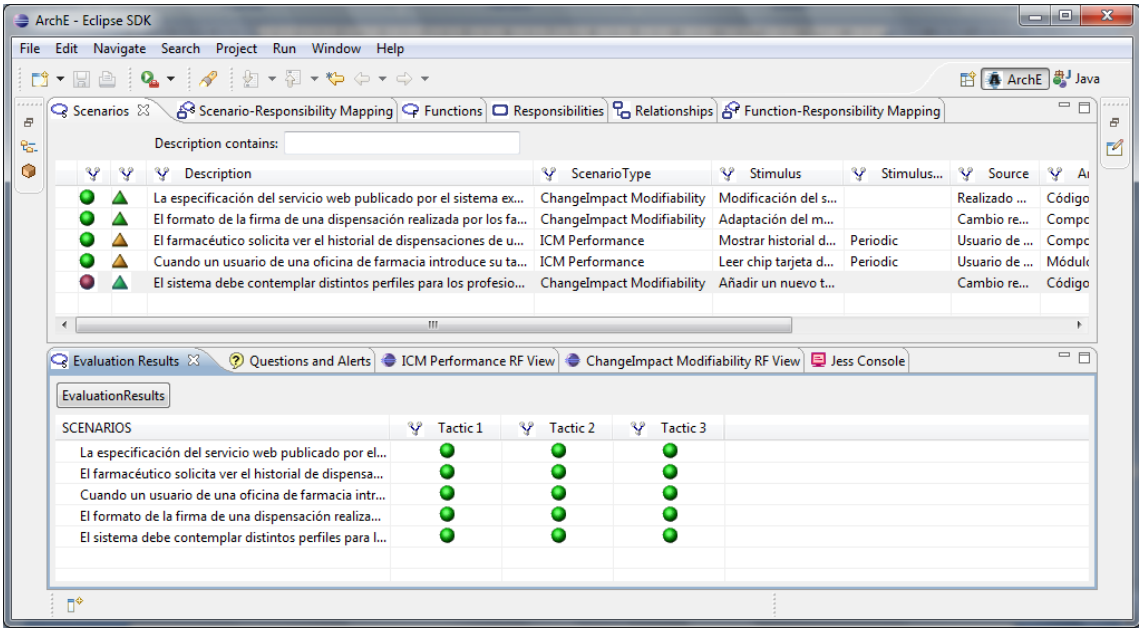


Figura 49. Vista Evaluation Results tras Iteración I

En esta vista se encuentra también un botón con el mismo nombre, que al pulsar sobre él, los iconos se muestran con forma triangular y codificados en colores. Estos iconos indican si la aplicación de la táctica, va a suponer una mejora o una degradación en la respuesta de los escenarios.

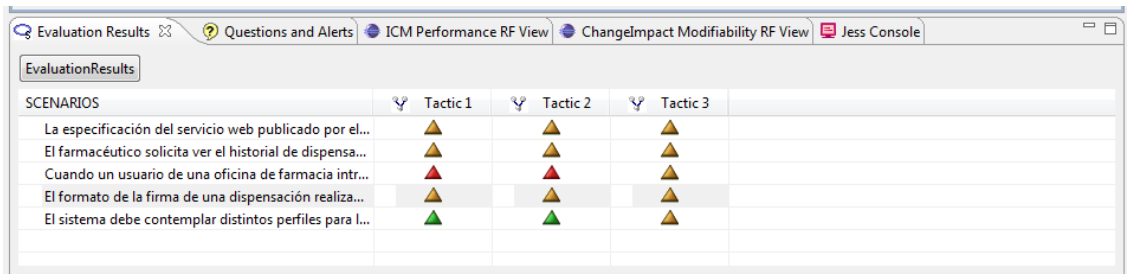


Figura 50. Botón EvaluationResults

Tabla 18. Tácticas tras Iteración I

Prioridad	Tipo	Categoría	Descripción
1	Split Responsability	Modificab.	La responsabilidad "Identificar profesional de farmacia" tiene fuertes dependencias con otras responsabilidades. Por tanto, podría ser una buena idea dividirla en dos responsabilidades hijas para minimizar las dependencias. Una estimación sugiere que podría reducirse el costo del escenario de "7.688571428571428" to "6.530375" persona/día.
2	Abstract Common Responsibility	Modificab.	Las responsabilidades "Alta profesional" e "Identificar profesional de farmacia" puede que compartan funcionalidad, y puede además estar en diferentes módulos. Por tanto, podría ser una buena idea trasladar la funcionalidad común a un módulo para mejorar la cohesión semántica. Una estimación sugiere que podría reducirse el costo de "7.688571428571428" a "6.5476473125" persona/día por este escenario de cambio.
3	Insert Intermediary	Modificab.	Dependencias desde responsabilidades al módulo "(M) Identificar profesional de farmacia", tienen un impacto sobre un escenario. Puede resultar beneficioso insertar un módulo intermediario para que los cambios sean menos costosos. Un intermediario de un coste de "15.500000000000004" persona/día podría reducir el coste de cambiar el escenario de "7.688571428571428" persona/día a "6.769041666666667" persona/día.

La táctica que mejor se ajusta a las necesidades del escenario de modificabilidad que se pretende satisfacer, es la segunda. Añadir perfiles a los profesionales de las oficinas de farmacia, afecta tanto al proceso de alta de un profesional como en la identificación del profesional. Por este motivo, se cree conveniente extraer la funcionalidad común para mejorar la cohesión, de forma que si cambia la funcionalidad común no se vean afectadas las responsabilidades implicadas. Aplicamos por tanto la segunda táctica.

Hay que destacar que de las tácticas sugeridas por la herramienta, no siempre la catalogada como más prioritaria es la que se debe aplicar. Es responsabilidad del arquitecto/diseñador analizar las propuestas y aplicar la táctica que mejor se adapte a la arquitectura del sistema evaluado.

Volviendo de nuevo a la vista de escenarios comprobamos el efecto que ha producido la aplicación de la táctica:

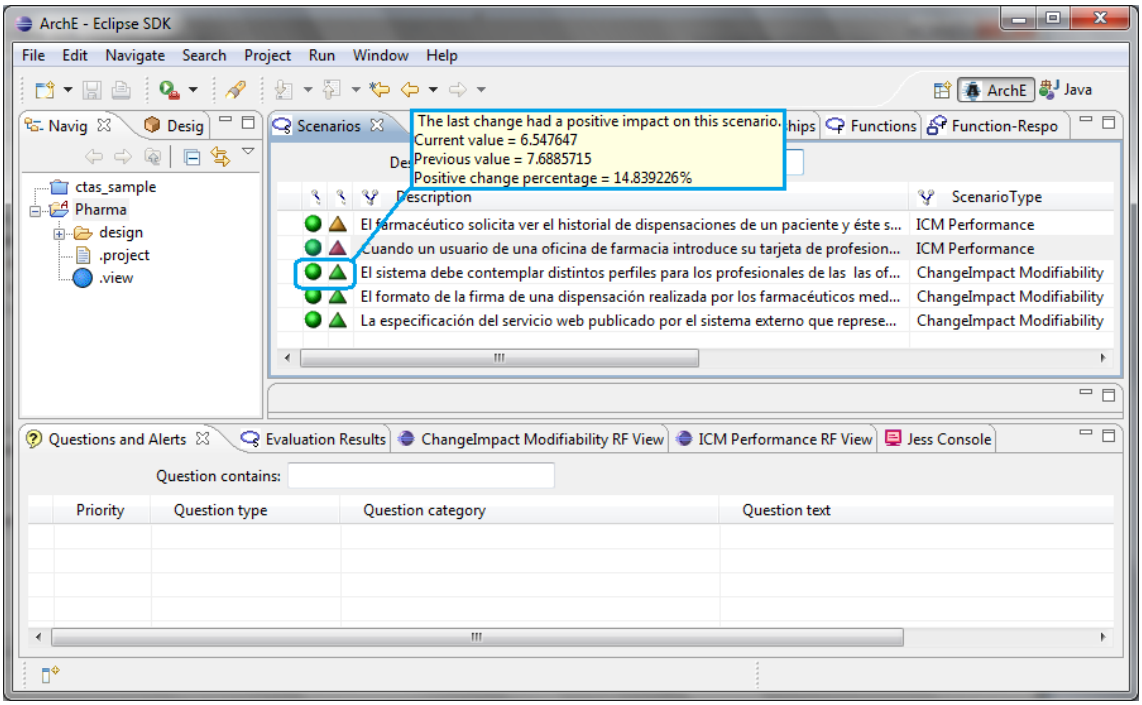


Figura 51. Escenarios tras Iteración II

Si pasamos el cursor encima del triángulo del escenario que se ha satisfecho, nos aparece la información de que la aplicación de la táctica supone una mejora del sistema de casi un 15 % respecto al diseño anterior.

En la figura previa se advierte que en la vista 'Questions and Alerts' el listado de preguntas y alertas está vacío, esto representa también que se ha resuelto de forma satisfactoria todos los escenarios de los atributos de calidad que se habían planteado.

El diseño resultante del sistema 'Pharma', de las decisiones que se han tomado siendo guiados por el asistente ArchE, es el siguiente:

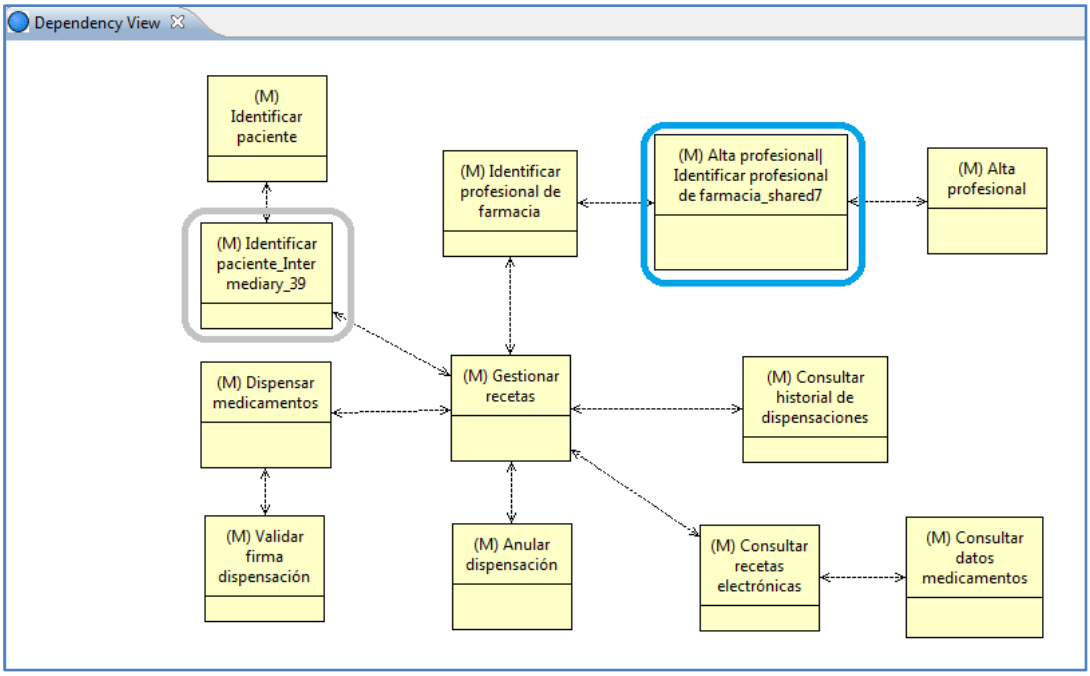
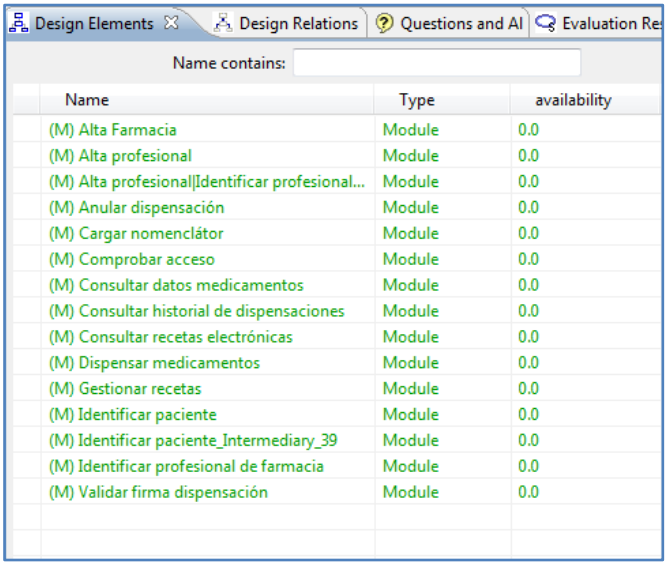


Figura 52. Vista de dependencias - Módulo común

Disponemos además de la vista “Design Elements”, que muestra los distintos módulos creados por el marco de razonamiento de modificabilidad tras los procesos de especificación de requisitos y aplicación de tácticas de mejora del sistema (en cuanto al atributo de calidad de modificabilidad).



Name	Type	availability
(M) Alta Farmacia	Module	0.0
(M) Alta profesional	Module	0.0
(M) Alta profesional|Identificar profesional...	Module	0.0
(M) Anular dispensación	Module	0.0
(M) Cargar nomenclátor	Module	0.0
(M) Comprobar acceso	Module	0.0
(M) Consultar datos medicamentos	Module	0.0
(M) Consultar historial de dispensaciones	Module	0.0
(M) Consultar recetas electrónicas	Module	0.0
(M) Dispensar medicamentos	Module	0.0
(M) Gestionar recetas	Module	0.0
(M) Identificar paciente	Module	0.0
(M) Identificar paciente_Intermediary_39	Module	0.0
(M) Identificar profesional de farmacia	Module	0.0
(M) Validar firma dispensación	Module	0.0

Figura 53. Vista de Elementos de Diseño

4. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURO

La realización del presente trabajo de investigación, me ha permitido profundizar en los elementos que intervienen dentro del proceso de diseño de arquitecturas software.

En la construcción de un sistema software la fase de diseño de la arquitectura es crucial, debido a que las decisiones que se hayan tomado sobre el modelo, va a repercutir en las fases posteriores de desarrollo software y en el grado de cumplimiento de los objetivos de negocio de la organización y de los requisitos subyacentes.

Los requisitos de calidad tienen especial importancia puesto que es una consecuencia directa del diseño de la arquitectura. Un buen diseño se consigue explorando las distintas alternativas arquitectónicas y seleccionando aquella que permita satisfacer los requisitos de calidad.

Esta tarea no es trivial para el arquitecto, por este motivo aparecieron métodos y herramientas que guiasen en el proceso de diseño.

Estos conceptos se han podido incorporar en la herramienta propuesta ArchE, que sirve para evaluar arquitecturas software de una forma sistemática. Esta herramienta posee la limitación de que sólo incorpora los marcos de razonamiento sobre los atributos de modificabilidad y rendimiento. Como comentan sus autores, es un asistente de diseño de arquitecturas en el ámbito educativo, que ofrece tácticas para cumplir con los escenarios de calidad y responsabilidades introducidos como entradas por el arquitecto. Como es lógico, no se puede valorar más allá del ámbito educativo.

Dentro de este contexto, puedo concluir que es una herramienta útil para la evaluación y diseño de arquitecturas. Particularmente cumplió con el objetivo de mejorar el diseño del caso práctico que propuse en relación al atributo de modificabilidad.

Como perspectivas de continuación se plantea:

- La adición de un nuevo marco de razonamiento sobre un atributo de calidad a la herramienta ArchE.
- Incorporar una nueva característica a la herramienta que le otorgue cierta capacidad semántica. Esto sería conveniente para evitar casos en los que hay una propuesta continua de aplicación de tácticas sobre el modelo, sin que ofrezca realmente una utilidad.

BIBLIOGRAFÍA

Builes, I. (2013). “ArchE: Arquitecturas y Toma de decisiones”. Trabajo de Investigación, UNED, ETS de Ingeniería Informática, Madrid - España.

Diaz, J., Kim, H., & Bianco, P. (2008). *The ArchE Reasoning Framework Interface Developer’s Guide*. Carnegie Mellon University. Pittsburgh - PA: Software Engineering Institute.

"INGENIERIA DEL SOFTWARE: UN ENFOQUE PRACTICO"
ROGER S. PRESSMAN , S.A. MCGRAW-HILL / INTERAMERICANA DE ESPAÑA, 2001

Fernández, J. M. (2010). *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, UNED, ETS de Ingeniería Informática, Madrid - España.

Gaitán, C. A. (2013). *Arquitecturas Software: Gestión de los atributos de calidad 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, UNED, ETS de Ingeniería Informática, Madrid - España.

“Preliminary Design of ArchE: A Software Architecture Design Assistant”. Felix Bachmann, Len Bass y Mark Klein. (TECHNICAL REPORT CMU/SEI-2003-TR-021). Disponible online:
http://www.sei.cmu.edu/library/abstracts/reports/03tr021.cfm

Pérez-Campanero, J. A. (2010). *Desarrollo de Software y de las Arquitecturas Software.* Tésis de Máster - Trabajo de Investigación, UNED, ETS de Ingeniería Informática, Madrid - España.

Felix Bachmann, Len Bass, and Robert Nord. “Modifiability Tactics,” CMU/SEI-2007-TR-002, September 2007.
http://www.sei.cmu.edu/reports/07tr002.pdf

"Integrating Quality-Attribute Reasoning Frameworks in the ArchE Design Assistant"
Andres Diaz-Pace, Hyunwoo Kim, Len Bass, Phil Bianco, and Felix Bachmann
https://resources.sei.cmu.edu/asset_files/WhitePaper/2009_019_001_29119.pdf

Página web del Trabajo Fin de Máster
http://www.issi.uned.es/Master_ISSI/TFdM/ArchE/index.htm

SEI. (2007). *ArchE Version 2.1 - User’s Guide Version 1.0.* Carnegie Mellon University. Pittsburgh - PA: Software Engineering Institute.

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

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

"Towards automation of architectural tactics application – an example with ArchE"
R. Champagne, S. Gagné (ÉTS, Montréal, Canada)
http://resources.sei.cmu.edu/asset_files/Presentation/2011_017_001_22700.pdf

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

SIGLAS, ABREVIATURAS Y ACRÓNIMOS

Fact Base (Base de Hechos): Los sistemas expertos son llamados así porque emulan el razonamiento de un experto en un dominio concreto. Una de las partes fundamentales que componen a un SE son las *bases de hechos*. Los hechos representan la estructura dinámica del conocimiento ya que su número puede verse incrementado a medida que se van relacionando las reglas.

Hibernate: Es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación.

IDE (*Integrated Development Environment*): Es un entorno de programación que ha sido empaquetado como un programa de aplicación; es decir, que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Middleware: Es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en los sistemas distribuidos. De esta forma se provee una solución que mejora la calidad de servicio, seguridad, envío de mensajes, directorio de servicio, etc.

PIM: El Modelo Independiente de Plataforma o "**P**latform **I**ndependent **M**odel" sirve como soporte para la generación de uno o varios modelos empleando lenguajes específicos de dominio o lenguajes de propósito general como Java, C#, Python, etc.

SOAP (siglas de *Simple Object Access Protocol*) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.

Vista: Es la representación concreta de un sistema en particular desde una perspectiva unitaria. Una vista es un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado.

Wrappers (Envolturas): Es una técnica de encapsulación. La diferencia de ésta radica en que los Wrappers son interpretados como los procesos generados o invocados tras cierta transformación y la encapsulación en sí es la estrategia para llegar a ellos. La técnica de Wrapper reduce el costo total de un cambio en una responsabilidad externa y los costos asociados con la propagación de la nueva transformación.

XML
XML, siglas en inglés de eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.