Universidad Nacional Educación a Distancia
Escuela Técnica Superior de Ingeniería Informática
Master Investigación Ingeniería del Software
Proyecto Fin de Máster

ArchE: evaluación de arquitecturas y toma de decisiones

Autor**:** *Pablo García Díaz*

Tutor: *José Félix Estívariz López*

Salamanca, Septiembre 2013

ÍNDICE

ÍNDICE DE ILUSTRACIONES

1. Introducción

Este documento se presenta como memoria del proyecto de fin de Máster denominado **"ArchE: evaluación de arquitecturas y toma de decisiones"**. Tal proyecto se realiza como trabajo de finalización del máster denominado en Investigación en Ingeniería del Software para la Universidad Nacional de Educación a Distancia.

La idea del proyecto surge directamente de la universidad de la mano del tutor del mismo, José Félix Estívariz, la cual propone un estudio sobre cómo influyen requisitos externos dados por actores que podríamos decir "no principales" al desarrollo de una arquitectura, de la cual siempre se tiene en cuenta los requisitos de la aplicación a la que va a servir.

El trabajo consiste en realizar un ensayo sobre el libro **"Software Architecture in Practice"**, *2ª Edition, by Len Bass, Paul Clmentes and Rick Kazmank.* El libro trata de cómo intervienen los requisitos globales o externos, atributos de calidad, a la hora de diseñar la arquitectura de un producto, tenerlos en cuenta y como aplicar su influencia en el desarrollo de la arquitectura. También se incluyen en el documento pautas indicadas en el artículo que se propone como apoyo: *Relating Business Goals to Architecturally Significant Requeriments for Software Systems* de cómo aplicar esos atributos de calidad.

La segunda parte del documento tiene como fin aplicar dichas pautas sobre un ejemplo concreto, para ello se ha elegido la herramienta **ArchE**, desarrollada por el SEI, que consiste en una aplicación de evaluación de arquitecturas en base a los atributos de calidad. Se tomará como ejemplo de aplicación CTAS, proyecto de ejemplo que trae consigo ArchE. Además de estos se incluye una guía de instalación y de uso de ArchE.

2. Objetivos del proyecto

El objetivo principal del proyecto es el estudio y comprensión de cómo los requisitos de calidad, o atributos de calidad, impactan sobre la arquitectura final, medir dicho impacto y aplicar reglas para dotar a dicha arquitectura del cumplimiento de los máximos atributos posibles para la arquitectura final este alineada con el mayor número de los objetivos de los *stakeholders* posible. A partir de este objetivo global se puede extraer el siguiente listado desglosando los objetivos del trabajo:

a) Estudio y comprensión de qué son los atributos de calidad y su importancia en las arquitecturas software.

b) Comprensión de las pautas de los textos propuestos de cómo aplicar dichos atributos a la arquitectura.

c) Estudio de la instalación y el manejo de la herramienta ArchE.

d) Aplicación de dichas pautas y demás conocimientos al mejorar una arquitectura mediante la herramienta ArchE.

e) Comprensión de la aplicación CTAS para la aplicación de las pautas.

f) Justificación de los objetivos anteriores mediante este documento.

3. Análisis y Ensayo: *Software Architecture in Practice*

3.1. El ciclo de Arquitecturas de Negocio ABC

El libro al que hará referencia el documento es *Software Architecture in Practice, Second Edition Len Bass; Paul Clements; Rick Kazman* donde se detalla el ciclo de vida de una arquitectura así como a está le afectan los llamados atributos de calidad, que son los requisitos de alto nivel de la compañía tales como, cuántos trabajadores hacen falta para mantener dicha arquitectura?, cuando tiempo de media se tarda en realizar una modificación?, en qué grado afectará al volumen de ventas del producto?, … se detallarán más adelante a lo largo del escrito. Pero antes de comenzar a escribir sobre todo esto es incuestionable dar una definición de arquitectura, que es una arquitectura? se puede tener claro que es el esqueleto de una vivienda si se refiere a una arquitectura de de edificios, pero que es una arquitectura software? en el libro se nos ofrece la siguiente:

La arquitectura de un software es la estructura de estructuras del sistema que comprende los elementos del software, las propiedades de dichos elementos vistas desde el exterior y las relaciones entre ellas.

Como puede observarse la arquitectura software puede asemejarse con la base y el esqueleto sobre el que asienta el producto final para el que es destinado, tal como el esqueleto de una vivienda pero está comprendida por elementos tipo software. También se hace referencia a una frase muy importante *propiedades vistas desde el exterior,* ya que la arquitectura va a ser "utilizada" por usuarios, sistemas, otros elementos software, es muy importante y parte inherente en ella el cómo se ve desde el exterior y como puede utilizarse.

También cabe destacar la importancia de las relaciones definidas entre los elementos software de los que componen la arquitectura ya que estos elementos pueden ser paquetes software, librerías de código fuente, protocolos de red, bases de datos, etc…por lo que dicha relación definirá a largo plazo su comportamiento.

ABC son las siglas, en inglés, de *Architecture Business Cycle* que es el ciclo de vida del software en el caso particular de las arquitecturas; como todo software se puede intuir cuáles serán las etapas de dicho pero el caso de las arquitecturas se tienen en cuenta singularidades que no se tienen para lo que puede ser conocido como "software estándar", a continuación se exponen dichas etapas o actividades del ciclo:

- **Oportunidad de negocio**

Ya en el primer punto del ciclo de vida es diferente al estándar, donde este punto de nacimiento sería simplemente el captar una necesidad en uno o varios clientes potenciales. En el caso de las arquitecturas software esa oportunidad debe ser conceptualmente algo de mayor volumen, se puede identificar si se contestan preguntas como tiene mercado objetivo?, cuál es ese mercado objetivo?, con que usuarios/sistemas interactuará?...

En un mundo ideal en esta fase estarían involucrados los arquitectos software, pero esto poquísimas veces pasa, para que desde el primer momento el software final este alineado con estas preguntas y no se tengan que bajar expectativas en fases posteriores.

- **Recolección de requisitos**

  Como es sabido existen muchas técnicas/lenguajes para representar los requisitos, como una matriz, como casos de uso, etc…, estas técnicas también son válidas para el caso de las arquitecturas pero hay que incorporar otras tales como las lecciones aprendidas de sistemas anteriores o el aprovechamiento de piezas ya creadas son muy válidas para las arquitecturas.

  El punto donde es similar a los sistemas comunes es la gran importancia de esta fase, que los requisitos sean completos y totalmente entendibles, por ambas partes, es clave para que el desarrollo y la arquitectura final redunde en satisfacción tanto en la parte cliente como en los desarrolladores y arquitectos.

- **Creación o selección de arquitectura**

  El título de la fase ya explica todo, en esta fase se realiza un estudio de las soluciones que haya en ese momento en el mercado en cuanto a arquitectura se refiere y realiza un estudio o comparación sobre que resultaría mejor, si adquirir una arquitectura o crearla.

- **Comunicación de la arquitectura**

  En este punto se debe entender y detallas por los arquitecturas cuales será la comunicación de la arquitectura con el producto software, interfaces de entrada y salida y demás. Este punto es clave para la fase pura de diseño y programación

porque hay que estar seguro que necesita la arquitectura para devolver los datos que se esperan de ella.

- **Análisis o evaluación de la arquitectura**

  Pura fase de análisis y diseño de un sistema software estándar, siempre con las singularidades que tiene una arquitectura y se van relatando en el documento. Cuando se habla de evaluación se refiere a que sobre dicho análisis y mediante diferentes procesos se valora el grado de cumplimiento de la arquitectura con los requisitos de calidad que se esperan.

  Una de estas técnicas de evaluación es la conocida por “de escenarios” de la que se expondrá en detalle este documento en la parte de ArchE.

- **Implementación**

  Fase de desarrollo y pruebas típica de cualquier aplicación informática, ya que se han definido los protocolos de comunicación con la arquitectura se pueden codificar a la vez tanto la arquitectura como el producto final para poder probarlos juntos en las llamadas pruebas integradas.

- **Garantizar la conformidad de la arquitectura**

  Aunque el nombre utilizado en el libro sea esté se asemeja totalmente a la fase de mantenimiento de una solución software, tener una vigilancia constante del buen funcionamiento, solucionar posibles incidencias y por supuesto demostrar que la arquitectura creada se asemeja a los requisitos. Sobre esto último se está trabajando mucho en los últimos años aunque el trabajo todavía se puede considerar inmaduro.

En este punto desde el libro se lanza una pregunta.. **que hace una buena arquitectura?**, la respuesta obvia es que cumple con el propósito para la que se ha creado, una respuesta totalmente valida. Pero si uno se imagina a dos arquitectos de dos empresas diferentes creando una arquitectura para los mismos requisitos seguramente creen dos productos distintos, como saber cuál es la mejor de las dos?, esta sería una pregunta muy difícil de responder y habría que realizar una gran evaluación para determinarlo.

Se nos ofrecen varias recomendaciones para que, a lo largo del ciclo de vida, se establezca la mejor arquitectura final posible, hay dos tipos: recomendaciones de proceso y recomendaciones de producto.

Recomendaciones de producto

1. La arquitectura debe ser liderada por un solo arquitecto, independientemente de que pueda tener un grupo a su cargo.

2. El arquitecto debe tener desde el primer momento los requisitos funcionales del producto software así como la lista de requisitos de calidad o atributos de calidad de la arquitectura (como la seguridad o modificabilidad) priorizados.

3. La arquitectura debe estar bien documentada, con una visión estática y una visión dinámica como mínimo, se explicarán más adelante. El objetivo de la documentación es que sea entendible por una persona que no ha tenido nada que ver con un esfuerzo mínimo.

4. La arquitectura y la documentación de la misma debe ser distribuida a los actores del sistema para que sean partícipes activos en su revisión.

5. La arquitectura debe ser medida por el mayor número de medidas cuantitativas (tiempo de espera) para que exista una evaluación con algo que se podrían llamar “valores reales”.

6. El diseño de la arquitectura debe comenzar por un esqueleto mínimo con las vías de comunicación para irlo evolucionando poco a poco de forma incremental, esto permite una integración con menor esfuerzo.

7. La arquitectura debe adaptarse a un conjunto de restricciones de recursos de la compañía, se recomienda que este sea el menor de los posibles, tales como presupuesto, utilización de la red, etc.

Recomendaciones estructurales

1. La arquitectura debe estar organizada en módulos, tales que estos actúen como cajas negras con sus responsabilidades funcionales. Mediante esta división se minimiza el esfuerzo y la dependencia ante cambios.

2. Cada módulo citado anteriormente debe tener especificada una interfaz con sus entradas y salidas.

3. La arquitectura debe cumplir los atributos de calidad mediante tácticas arquitectónicas, las cuáles serán mentadas posteriormente.

4. La arquitectura no debe depender de una versión particular de un producto comercial, para no atarse solamente a una compañía.

5. Los módulos que producen datos deben estar separados lo máximo posible de los que consumen datos, ya que disminuye el esfuerzo ante un cambio, modificabilidad.

6. Para sistemas de procesamiento paralelo la arquitectura debe ofrecer procesos bien definidos que, aunque toquen varios módulos, sean transparentes al sistema.

7. Dichos procesos no deben estar sujetos a ejecutarse en un procesador particular, debe poder procesarse en cualquiera.

8. La arquitectura debe hacer las mismas cosas de la misma manera en todos los módulos, es decir, seguir un conjunto de patrones claro.

3.2. Atributos de calidad software

Que sería para una arquitectura software un atributo de calidad? simplemente sería una cualidad o característica que pueda ser medible dentro de lo posible que determine si la arquitectura cumple o no unos estándares de calidad. Un ejemplo de atributo de calidad sería la seguridad, que protocolo de seguridad cumple la arquitectura? que nivel de seguridad en las contraseñas permite?, están cifrados los mensajes entre los módulos del proyecto?, respondiendo a estas preguntas y teniendo un baremos que determine un mínimo o máximo de calidad podríamos determinar si la arquitectura es segura o no, o como mínimo en qué grado cumple el atributo de seguridad.

A menudo ocurre que estos atributos de calidad "chocan" con los requisitos o atributos funcionales del producto software, entendemos requisito funcional a la capacidad del sistema para hacer el trabajo para el que ha sido concebido. Porque chocan? pues porque en la mayoría de ocasiones la funcionalidad restringe llegar al nivel de calidad de un atributo necesario, por ejemplo la utilización de una gran base de datos, porque se necesita el almacenamiento de una gran cantidad de información, penalizará seguramente al rendimiento. Por supuesto que siempre hay maneras de que este impacto sea el menor posible y, para ello, es necesario implicar tanto al arquitecto como a los responsables del negocio en el desarrollo del producto.

Si ya se ha tenido experiencia realizando proyectos software se puede pensar que, para cumplir con la funcionalidad exigida, el producto final puede realizarse en un solo módulo sin ningún tipo de organización interna ni documentación, en cambio se piensa en que el sistema debe ser mantenible a futuro, atributo de calidad de mantenibilidad, y por ello se realiza una documentación asociada para hacerlo más entendible y se descompone en módulos, normalmente, para realizar el desarrollo mediante varias personas que trabajen de forma paralela y bajar el tiempo de entrega del producto.

El alcanzar en un grado los atributos de calidad no es tarea fácil, por ende hay que considerarlos a lo largo de las fases de análisis, diseño, implementación y despliegue, pero hay que tener en cuenta que ningún atributo de calidad es dependiente de una sola fase si no que en todas las fases se tienen que tener en cuenta a todos los atributos de calidad deseados.

Un ejemplo podría ser el rendimiento, que tiene que ver con los mensajes que habría entre componentes arquitectónicos y no arquitectónicos, de los algoritmos usados, definidos en el diseño, y como se codifiquen dichos algoritmos, fase de implementación.

La arquitectura es clave para alcanzar los atributos de calidad, ya que supone la base donde se asienta el producto software final. Por ello cuando se define y se implementa un arquitectura hay que tener en cuenta dichos atributos y, a ser posible, de forma priorizada para poder ponderar la importancia de unos sobre otros.

El estudio de los atributos de calidad es una discusión que lleva perenne en la comunidad tecnológica desde la década de los 70, de esta discusión se han identificado tres problemas o cuestiones bastante arraigadas ya:

1. No tiene sentido definir un sistema con los atributos de calidad, por ejemplo no tiene sentido decir que un sistema es modificable. Sin embargo se puede decir que es más o menos modificable con respecto a otros.

2. Otro foco es la catalogación de una característica dentro de un atributo de calidad, un fallo en el sistema concierne a la seguridad, la disponibilidad o la usabilidad?, posiblemente sea a los tres..

3. Cada atributo de calidad tiene su propio "vocabulario", el rendimiento tiene *eventos*, la seguridad tiene *ataques* y la disponibilidad tiene *fallos* cuando parece que podrían ser lo mismo.

Para las dos primeras cuestiones una posible solución es usar escenarios para la caracterización de los atributos de calidad, también se podría asignar un grado de cumplimiento de un atributo en un sistema. Para el tercero claramente sería necesario un glosario de términos para estar seguro que todos los *stakeholders* estén hablando en el mismo idioma.

Se acaba de hablar de **escenarios**, concepto clave en la evaluación de atributos de calidad, que no son más que una situación que afecta a un atributo de calidad que se compone de seis partes:

- Estímulo: condición que cumple el sistema para que comience el escenario.

- Fuente del estímulo: entidad que genera el estímulo anterior.

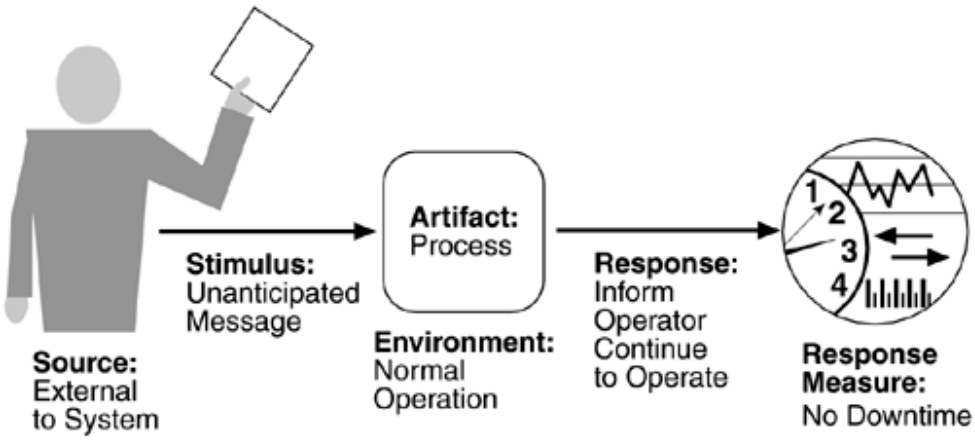
- Medio ambiente: condiciones del sistema en las que puede darse el estímulo.

- Artefacto: se refiere a que si el sistema entero es afectado o la lista de partes del mismo.

- Respuesta: actividad que realiza el sistema inmediatamente después de llegar el estímulo.

- Medición de la respuesta: manera en la que se mide la respuesta.

Existen escenarios generales, que afectan a cualquier tipo de sistema y escenarios concretos, que son los que son específicos del sistema en particular. A continuación el libro propone un ejemplo de escenario para el atributo de calidad de disponibilidad:



PONER LO DE LA FIGURA

En este caso el escenario tendría las siguientes características:

- Estímulo: mensaje no previsto.

- Fuente del estímulo: externo al sistema.

- Medio ambiente: sistema en condiciones normales.

- Artefacto: proceso cualquiera.

- Respuesta: informar del mensaje y seguir con la operativa normal.

- Medición de la respuesta: que no haya tiempo de inactividad.

El momento de la creación de dichos escenarios es en la fase de captura de requisitos, pero en la práctica raramente se cumple esto y se realiza más adelante. Una buena práctica es tener mediante tablas todos los escenarios que afecten a cada atributo de calidad con sus seis partes bien definidas para tenerlos presente durante todo el ciclo del proyecto ya que el olvido o la omisión de algún escenario puede conllevar a consecuencias graves.

A continuación se listan los principales atributos de calidad que se comentan en el libro y que aplican la mayoría de sistemas/arquitecturas software:

Disponibilidad

Referente a los fallos de un sistema y a las consecuencias derivadas de dicho fallo. Un fallo puede apreciarse cuando un servicio o funcionalidad que el sistema debe ofrecer no funciona correctamente y es perceptible por un usuario o un sistema.

En lo que a este atributo se refiere son muy importantes la forma en que se detecta un fallo, la frecuencia con que ocurre un fallo, el tiempo que se permite al sistema estar sin funcionar correctamente, el tiempo en que tarda el sistema en recuperarse y recuperar su actividad normal..

Modificabilidad

Coste o esfuerzo de realizar un cambio en el sistema, aquí se nos incita a pensar en dos variantes, que puede cambiar y quien y cuando realiza el cambio. Hay que tener en cuenta que puede cambiar **todo**, módulos de software donde se realizarían cambios en el código fuente, protocolos de comunicación, herramientas externas que son consumidoras o productoras de información del sistema, etc.., y en cuanto a la segunda cuestión puede ser el mismo equipo que desarrollo el proyecto, un equipo con personas diferentes, o el propio usuario final si se le ha dejado esa opción, se puede imaginar una página web con la funcionalidad de permitir al usuario ingresar los contenidos de la misma.

Rendimiento

Tiempo que tarda el sistema en responder cuando se produce la llegada de un evento, un evento puede ser una petición de un usuario, un sistema externo, la llegada de un determinado día u hora.

Relacionado con este atributo están los recursos consumidos, se supone que a menor tiempo mayor número de recursos pero esto no siempre es así, además si para cada respuesta se ocupan una gran cantidad de recursos puede penalizar a la disponibilidad.

Conceptos relacionados con la latencia, tiempo entre la llegada del estímulo y la respuesta del sistema, plazos de procesamiento, restricciones del sistema, por ejemplo el número de transacciones que es capaz de ejecutar una base de datos de forma paralela.

Seguridad

Es la capacidad de un sistema de resistir y protegerse ante el uso no autorizado de sus propios recursos sin dejar de ofrecer los servicios para el que se ha concebido.

Un ataque sería un intento de un uso no autorizado, las principales características de un sistema *seguro* serían:

- No rechazo: se dice que se cumple el “no rechazo” si una transacción no puede ser negada por ninguna de las partes involucradas.
- Confidencialidad: cuando los datos y servicios contratados no están disponibles para el acceso no autorizado.
- Integridad: significa que los datos se encuentran tal y como se dejaron en la última transacción registrada.
- Disponibilidad: indica que el sistema estará disponible aunque esté ocurriendo un ataque.
- Auditoría: el sistema podrá ser reconstruido en base a todas las transacciones ocurridas.

Capacidad para ser probado

Proviene del inglés que se expresa en una sola palabra, *testeability*, y se trata de la facilidad con la que el producto puede ser probado. Normalmente las pruebas se llevan la mayor parte del tiempo de coste del desarrollo del software.

Para poder ser probado un software debe permitir que sus estados internos sean visibles desde el exterior, normalmente se utilizan plataformas software destinadas puramente a esto. Hay que tener en cuenta que hay varios tipos de pruebas, unitarias, integradas, de despliegue, de usuario…

Usabilidad

Facilidad que tiene el usuario en cuanto a manejo de la herramienta se refiere. Se trata de un atributo muy "subjetivo" o, dicho de otra manera, difícil de medir ya que depende de muchos factores sobre todo del usuario final. Dentro de la usabilidad se pueden diferenciar las siguientes áreas de acción:

- Aprendizaje de las características del sistema por parte del usuario.
- Minimizar el impacto de los errores, como hacer que un error de usuario tenga un impacto mínimo sobre el funcionamiento.
- Adaptar al sistema el usuario final.
- Aumento de la confianza y la satisfacción por parte del usuario final.

Para paliar estas áreas de acción es muy recomendable el uso de prototipos durante el ciclo de vida así como poner empeño en la documentación referente al manual de usuario.

A continuación se describen otros atributos de calidad que, al ser menos importantes y menos comúnmente encontrados, se detallan más resumidamente:

- **Escalabilidad:** modificación de la capacidad del sistema, por ejemplo un aumento del número de usuarios que actúan de forma paralela.
- **Portabilidad**: modificación en la plataforma del sistema, por ejemplo si el software funciona en varios sistemas operativos diferentes.
- **Interoperabilidad:** capacidad para comunicarse con nuevos sistemas externos.

Otro tipo de atributos de calidad que se escapan a lo puramente software son los llamados atributos de calidad del negocio, que tienen más que ver con objetivos estratégicos de la compañía, a continuación se muestran los más comunes:

- **Tiempo de mercado:** se trata del tiempo en el que el sistema tiene que estar a la venta por restricciones de competencia, es decir, si está desarrollando un tipo software dónde la primera compañía va a sacar un producto en 6 meses hay que ser capaz de acabar ese producto en menos tiempo para adelantarse.

- **Coste y beneficio:** la arquitectura y el producto final normalmente deben adaptarse a un presupuesto a menudo rígido, el cual está planificado para maximizar el beneficio lo máximo posible con la aplicación desarrollada.

- **Vida útil:** conocimiento de si el sistema está pensando para una larga duración con un mantenimiento pesado o si es algo de una duración puntual determina habitualmente la arquitectura que se le elige así como los atributos de calidad prioritarios.

- **Mercado objetivo:** si el producto final está dirigido a un determinado tipo de usuario o si el espectro es más amplio y se puede llegar al usuario común.

- **Programa de implementación:** básicamente se refiere a la posibilidad de vender el producto con una funcionalidad base y a partir de ahí adquirir funcionalidades extras de manera separada o si el producto se vende como un todo.

Para terminar con este punto se mostrarán las tres principales cualidades relacionadas puramente con las arquitecturas, existen más pero se quiere hacer un énfasis especial en estas tres:

- **Integridad conceptual:** indica si se unifica el diseño en todos los niveles de la arquitectura, es decir, si las acciones que realiza la arquitectura son similares en todas las partes de la misma. Dicho de otra manera es la capacidad de coherencia en las distintas partes de la arquitectura que la hará más entendible y mantenible a futuro maximizando el cumplimiento de la mayoría de atributos de calidad que se han mentado hasta ahora.

- **Exactitud y Completitud:** se refiere al cumplimiento por parte de la arquitectura de todos los requisitos habiendo tenido en cuenta las diferentes restricciones que puede tener el sistema.

- **Edificabilidad:** es la traducción directa al español de *buildability*, que tan raro suena para el español. Se refiere a la capacidad de desarrollar el sistema por el

equipo de personas pensado y en el tiempo planificado así como la posibilidad de absorber cambios durante las diferentes fases de desarrollo sin variar la fecha de entrega. Para esto hay que tener en cuenta muchos factores, tales como el conocimiento del arquitecto del sistema final, la claridad de los requisitos, la experiencia del equipo de trabajo, la descomposición con coherencia en módulos, ..

3.3. Tácticas para alcanzar los atributos de calidad

Una táctica es una decisión dentro de tiempo de diseño que influye en el grado de cumplimiento de uno o varios atributos de calidad. En este punto del documento se distinguirán las tácticas más comunes para los principales atributos de calidad presentados en el punto anterior.

Otros conceptos relacionados son los de estrategia arquitectónica, que se refiere a un conjunto de tácticas y patrón arquitectónico a un paquete de tácticas con una determinada finalidad.

Se puede pensar que una táctica es una opción de diseño del arquitecto, por ejemplo introducir redundancia para aumentar la disponibilidad pero esto mismo repercutirá negativamente en la modificabilidad, por esto normalmente afectarán a más de un atributo de calidad.

Normalmente las tácticas pueden ser descompuestas hasta formar un árbol jerárquico, esto se hace para tratar de refinar la decisión y que afecte más positivamente que negativamente en el conjunto de la arquitectura.

3.3.1. Tácticas arquitectónicas de disponibilidad

La disponibilidad es un atributo de calidad que tiene en cuenta los fallos, caídas de una funcionalidad especificada visible al usuario, y el tiempo de recuperación a esos fallos.

Existen tres tipos de tácticas para la disponibilidad, que se corresponden con la detección de fallos, recuperación de fallos y la prevención de fallos, a continuación se muestran las más típicas:

Detección de fallos

1. **Ping / Echo:** consiste básicamente en que un componente emite un *ping* (pregunta) a otro componente y espera un *echo* (respuesta) del mismo en un tiempo predefinido, si no llega dicha respuesta estaríamos ante un caso de caída.

   Este sistema puede organizarse de diferentes maneras, desde la más clásica donde un nodo emite ping a varios hasta el diseño de una jerarquía de varios nodos para repartir las responsabilidades y tiempo de proceso.

2. **Heartbeat:** en español latido de corazón se fundamenta en la emisión de una señal por parte de un componente hacia otro componente en un periodo de tiempo constante. Si el componente receptor no recibe la señal en el tiempo esperado detecta la caída.

   En muchas ocasiones se utiliza este sistema para enviar datos, por ejemplo para hacer llegar los mismos datos varias veces asegurando su llegada.

3. **Excepciones:** es una opción basada en la inclusión en código fuente de excepciones controladas para poder reaccionar. Esta es una opción para que un mismo componente se controle a sí mismo.

Recuperación de fallos

1. **Votación:** se utiliza para sistemas con varios procesadores que trabajan de forma paralela donde uno realizar el rol de *votante*; los demás procesadores envían un dato calculado en base a un algoritmo al votante y éste, mediante otro algoritmo, detecta errores en los demás procesos o fallos.

2. **Redundancia activa:** dirigida a sistemas que trabajan con componentes redundantes, se trata de hacer que todos los procesos respondan a eventos en paralelo, se suelen utilizar las primeras respuestas y las últimas se descartan.

   Cuando se produce un fallo el tiempo antes de la última copia de seguridad es medida en milisegundos. Esta táctica es muy común en sistemas de cliente de servidor como los gestores de bases de datos dónde son necesarias respuestas rápidas antes un error.

3. **Redundancia pasiva:** se implementa en sistemas con varios componentes, uno hace el rol de principal y los otros de esclavos; el principal emite señales hacia los esclavos indicando en qué estado deberían estar estos últimos.

   Cuando se produce un fallo el componente principal se asegura, a través del último estado del sistema, de que la última copia de seguridad es lo bastante consistente para restaura a dicho estado.

4. **Repuesto:** esta táctica se basa en mantener a espaldas un sistema *de repuesto* que pueda reemplazar a partes del actual. Se debe mantener el último estado del sistema en un dispositivo persistente para que el sistema de reemplazo pueda partir de ahí.

5. **Operación sombra:** se asienta en la ejecución de un proceso o componente que ha fallado anteriormente en modo *sombra* sin que sea perceptible al usuario y asegurarse de que su comportamiento es el esperado antes de incorporarse al modo de trabajo normal.

6. **Checkpoint / rollback:** muy popular entre los sistemas operativos se trata de grabar estados coherentes del sistema y así cuando se produce un fallo puede volver a ese estado despreciando lo ocurrido posteriormente.

Prevención de fallos

1. **Ponerse en fuera de servicio:** esta táctica desactiva un componente para realizarle un bloque de pruebas con el fin de evitar fallos previsibles. Lo normal es que o bien una estrategia de arquitectura o bien el mismo sistema apoye durante el retiro temporal del componente en cuestión.

2. **Transacciones:** el sistema realiza agrupaciones de acciones, en este contexto llamadas transacciones, que se pueden deshacer de una sola vez. Se utiliza sobre todo para no dejar datos incoherentes si no se ejecutan de forma éxitos varias transacciones dependientes.

3. **Monitorización de procesos:** táctica basada en mantener un proceso independiente, el monitor, que, una vez detectada la caída de otro proceso, se encargue de crear una nueva instancia del mismo en un estado coherente.

3.3.2. Tácticas arquitectónicas de modificabilidad

Que sirva como recordatorio decir que la modificabilidad es al atributo de calidad que mide el coste de la implementación de un cambio en el software, al igual que la disponibilidad las tácticas se agrupan según la finalidad, el primero tiene como objetivo limitar lo máximo posible el número de módulos directamente afectados por un cambio, se conoce como *localización de modificaciones*. El objetivo del segundo grupo es acotar las modificaciones a los módulos localizados, *prevención del efecto dominó*. El último grupo tiene como finalidad controlar el tiempo y el coste de la implementación del cambio, *acortar tiempo implementación*.

Localización de modificaciones

1. **Mantenimiento de la coherencia semántica:** esta táctica pretende dejar un módulo diseñado tal que sus responsabilidades tengan las menores dependencias posibles con otros módulos.

2. **Anticipación a cambios esperados:** normalmente una vez en curso un proyecto surgen cambios que hacer en otros ciclos de vida posteriores, esta táctica consiste en dejar los módulos diseñados para que cada cambio sólo afecte a un módulo, esto sería el mejor caso posible.

3. **Creación de un módulo general:** simplemente es la definición de un módulo con funcionalidades comunes, de configuración, de la aplicación, por ejemplo idioma, parámetros generales, …, así cualquier cambio al respecto estará acotado a un solo módulo.

Prevención del efecto dominó

Como se ha comentado anteriormente estas tácticas sirven para paliar el efecto que tiene una modificación concreta sobre un módulo en los demás módulos del sistema, antes de exponer las tácticas conviene citar las diferentes dependencias que existen para un mejor entendimiento de las tácticas:

- Sintaxis de datos: para que el módulo B consuma los datos que produzca el módulo A los datos deben ser consistentes en tipo o formato al producirse y al consumirse.

- Sintaxis de servicio: para que B pueda compilar y ejecutarse correctamente los servicios que proporciona A e invoca B deben ser coherentes con los supuestos del módulo B.

- Semántica de datos: para que B pueda ejecutar correctamente la semántica de los datos producidos por A y consumidos por B deben ser consistentes con los supuestos por B.

- Semántica de servicio: para que B puede ejecutar correctamente la semántica de los servicios producidos por A y utilizados por B deben ser consistentes con los esperados por B.

- Secuencia de datos: para que B puede ejecutar correctamente debe recibir los datos producidos por A en una secuencia fija.

- Secuencia de control: para que B puede ejecutar correctamente A debe ejecutar correctamente dentro de unas limitaciones de tiempo indicadas.

- Identidad de una interfaz de A: para que B pueda compilar y ejecutar correctamente la identidad de la interfaz de a debe ser consistente con lo supuesto por B.

- Localización de A: dependencia en tiempo de ejecución, para que B pueda ejecutar correctamente A debe encontrarse en una ubicación concreta en tiempo de ejecución.

- Calidad de servicio / datos de A: para que B pueda ejecutar con éxito las propiedades de la calidad de los datos o servicios prestados por A debe cumplir con lo esperado por B.

- Existencia de A: para que B pueda ejecutar correctamente A debe existir.

- Comportamiento de los recursos de A: para que B pueda ejecutarse correctamente necesita que los recursos de A se comporten a lo esperado.

A partir de aquí se pasa a listar las tácticas relacionadas:

1. **Ocultación de información:** descomposición de las responsabilidades de un módulo en concreto de tal manera que se separe la parte privada de la parte pública. Con esto se pretende aislar todo lo posible las responsabilidades y que cualquier cambio que no afecte a otro módulo sólo sea este el módulo impactado.

2. **Creación y mantenimiento de interfaces de comunicación:** con esta táctica se promueve a la creación y mantenimiento de interfaces entre módulos, de esta manera cualquier cambio de lógica en la interfaz sólo afecta a dicha interfaz.

3. **Restricción de rutas de comunicación:** está táctica intenta eliminar líneas de comunicación entre módulos de tal manera que se reduzcan los módulos que consumen información de un módulo dado y que se reduzca también el número de módulos que produce datos para el mismo.

4. **Utilización de intermediario:** básicamente es la creación de un módulo que actúe de intermediario para las comunicaciones de dos módulos dados, acotando los cambios posibles a dicho módulo. Estos intermediarios pueden ser los tipos de dependencia que hemos visto anteriormente.

Acortar tiempo de implementación

Se trata de tácticas muy generales para que cambios *normales* dentro del ciclo de vida de un producto software afecten lo mínimo al tiempo de coste:

1. **Archivos de configuración para establecer los parámetros de arranque**

2. **Utilizar polimorfismo para llamar a funciones en tiempo de ejecución**

3. **Utilización de protocolos para la unión de procesos independientes en tiempo de ejecución**

Sirva decir que este será el atributo de calidad usado para el ejemplo sobre la aplicación ArchE, algunas de las tácticas vistas en este punto se aplicarán más adelante en el documento.

3.3.3. Tácticas arquitectónicas de rendimiento

Para que sea fácil recordar lo expuesto anteriormente pueda recordar el atributo de rendimiento se encarga de medir si un sistema responde a un evento en un tiempo acordado, este tiempo se conoce como tiempo de latencia.

Al llegar el evento, este tiene que ser procesado antes de generar una respuesta, esto hace que entren en escena dos colaboradores básicos en el procesamiento: el consumo de recursos (CPU, ancho de banda de red, memoria, acceso a disco, …) y el tiempo de bloqueo (tiempo que un proceso puede estar bloqueado porque está esperando a que se libere un recurso necesario para su secuencia de comandos). Con todo esto las tácticas arquitectónicas de rendimiento se dividen en tres grupos: *demanda de recursos*, *gestión de recursos*, *arbitraje de recursos*.

Demanda de recursos

1. **Aumento de la eficiencia de computación:** se trata de hacer lo más eficiente posible el proceso, mejorando su algoritmo para que consuma la mínima CPU, haciendo que en vez de acceder a disco se acceda a memoria que normalmente es más rápido también sería un ejemplo válido.

2. **Reducción de la sobrecarga de cálculo:** se trata de reducir el acceso a recursos lo máximo posible, un ejemplo es eliminar el acceso a intermediarios vistos en el anterior punto, esto sirve como ejemplo al enfrentamiento entre rendimiento y modificabilidad.

3. **Gestionar la tasa de eventos:** se intenta aumentar el tiempo de muestreo con el fin de sincronizar eventos.

4. **Control de frecuencia de muestreo:** aumentar el tiempo de muestreo para estar al tanto de la llegada de un evento llegando incluso a poner algunos en cola, con esto se reduciría el tiempo pero se podrían perder peticiones.

5. **Límite tiempo ejecución:** esta táctica se basa en poner límites al tiempo de procesamiento de la respuesta al evento, depende mucho del contexto ya que en ocasiones no tiene sentido.

6. **Limitar tamaños de cola:** al limitar el tamaño de la cola se pretende minimizar los recursos utilizadas para procesar las llegadas.

Gestión de recursos

1. **Introducir concurrencia:** sirve para reducir el tiempo de bloqueo ya que se tendría la capacidad de procesar varias peticiones a la vez. Necesita una gestión de recursos adecuada para no malgastar la capacidad del sistema.

2. **Redundancia de datos o cálculos:** se trata de crear duplicados de datos o cálculos ya realizados para así "doblar" los recursos accesibles y bajar el tiempo de CPU ahorrado por el cálculo ya guardado.

3. **Aumento de recursos:** conlleva una inversión en hardware y se refiere a aumento de procesadores y su capacidad, mayor tamaño de memoria, mayor ancho de banda en las redes de comunicación.

Arbitraje de recursos

1. **Colas *First-in/First-out*:** gestión de los recursos en base a primer proceso que lo pide primer proceso que lo consume. A priori parece la opción más equitativa pero choca con dos casuísticas, una es que haya procesos con mayor prioridad que otros y la segunda surge de la posibilidad de que exista algún proceso gaste un tiempo proporcionalmente superior a los demás por lo que puede provocar un tiempo de espera inaceptable.

2. **Prioridad fija:** trata de dar a cada petición una prioridad (típicamente alta, media, baja) según las necesidades, de esta manera las peticiones con mayor prioridad pueden procesarse antes. La contraindicación de esta táctica es la posibilidad de que peticiones con prioridad baja mueran de inanición porque siempre son "adelantadas" por peticiones de mayor prioridad.

3. **Prioridad dinámica:** similar a la anterior con la particularidad que la asignación de la prioridad es dinámica y puede depender de distintas variantes como la fecha de llegada, basada en algún archivo de configuración externo, etc…

3.3.4. Tácticas arquitectónicas de seguridad

La seguridad es el atributo de calidad que refleja la resistencia a posibles ataques por parte de un tercero y la capacidad (tiempo y recursos) que el sistema tiene para recuperarse de dichos ataques. Las posibles tácticas se dividen en grupos, al igual que los anteriores, los hay relacionados con la *resistencia contra ataques*, *detección de ataques*, *recuperación de ataques*.

Resistencia contra ataques

1. **Autenticación de usuarios:** táctica que se basa en asegurar que un usuario (sistema o persona) es quien dice ser, para implementar esta táctica se utilizan contraseñas, certificados, firmas electrónicas o identificaciones biométricas.

2. **Autorización de usuarios:** administración de los derechos de utilización del sistema para los usuarios autentificados. La finalidad de esto es restringir los privilegios críticos a un grupo limitado de usuario.

3. **Confidencialidad de datos:** mantenimiento de los datos para su acceso no autorizado, para esto se utilizan técnicas como el cifrado de datos y los enlaces de comunicación.

4. **Integridad de los datos:** táctica que sirve para mantener los datos tal y como se entregaron, para ello se manejan redundancias en el almacenamiento de dichos datos combinado con el cifrado de los mismos.

5. **Limitar la exposición:** consiste en acotar los servicios que provee el sistema lo máximo posible para no exhibir a los mismos contra un posible ataque.

6. **Limitar el acceso:** táctica asociada totalmente con las aplicaciones conocidas como **cortafuegos**, éstas tienen un funcionamiento tal que impiden el acceso a servicios o aplicaciones potencialmente vulnerables de recibir un ataque.

Detección de ataques

Para implementar la funcionalidad de detección de ataques se utilizan las conocidas como aplicaciones de *detección de intrusos*. Estos sistemas actúan comparando el tráfico continuamente de tal forma que, si detectan una anomalía en dicho tráfico avisan un posible ataque inminente.

Recuperación de ataques

Estas tácticas son similares a las tácticas de recuperación asociadas a la disponibilidad vistas en puntos anteriores, tales como la vuelta a un estado coherente del sistema guardado anteriormente. Existen otras tácticas relacionadas con la identificación del atacante, normalmente necesita un proceso que actué como una especie de auditoría de todo movimiento que haya ocurrido, de esta manera se puede detectar que aplicación produjo el ataque.

3.3.5. Tácticas arquitectónicas de capacidad de prueba

El objetivo principal del atributo de calidad de capacidad de prueba es la de asegurar una prueba será más fácil por cada ciclo de desarrollo del software que se realice, para ello se reparten las tácticas en dos tipos: *gestión de entrada y salida* y tácticas para *el control interno*.

Gestión de entrada y salida

1. **Grabación / Reproducción:** esta táctica permite el almacenamiento (*grabación*) de información de salida correcta de un componente y su uso (*reproducción*) en la prueba de otro que lo necesita.

2. **Interfaz independiente de la aplicación:** esta táctica consiste en la separación en dos componentes de la interfaz por un lado y la aplicación por otro para utilizar diferentes conjuntos de pruebas especializados en cada uno.

3. **Especialización de interfaces:** se basa en la utilización de interfaces específicas para pruebas las cuales permiten tener almacenados los valores de variables para un componente en una ejecución normal o anormal.

Supervisión interna

Este grupo de tácticas es muy variado ya que esencialmente trata de realizar un control o supervisión de lo que acontece en cada componente cuando se está ejecutando una prueba. Las tácticas incluyen desde mantener una aplicación monitor que muestre los valores de las variables del componente, mantener un registro de eventos o trazas para conocer el avance de la ejecución.

3.3.6. Tácticas arquitectónicas de usabilidad

La usabilidad se encarga de medir la facilidad para un usuario de realizar una tarea concreta, en este caso se diferencian dos tipologías de tácticas: *tiempo de ejecución*, apoyo al usuario cuando está ejecutando la herramienta, y *diseño del sistema*, apoyo al usuario durante la fase de diseño de la interfaz.

Tiempo de ejecución

1. **Iniciativa de usuario:** se fundamenta en dejar opciones al usuario para facilitar el uso del sistema, un ejemplo claro son los botones de deshacer / rehacer, que sólo actúan si el usuario quiere.

2. **Iniciativa de sistema:** se basan en que el sistema actúa de facto durante el uso cuando percibe que puede ayudar al usuario, ejemplo de esto es ya típico tutorial que salta al abrir por primera vez una aplicación y que sirve para facilitar al usuario los primeros pasos en la utilización de la aplicación.

3. **Iniciativa mixta:** como su nombre indica se refiere a acciones que engloban acciones del usuario y sistema, ejemplo de ello es la pantalla que muestran algunas aplicaciones sobre cancelar operaciones que necesitan la aprobación de usuario.

4. **Mantenimiento de un modelo de tarea:** el sistema monitoriza la tarea que está realizando el usuario y muestra mensajes de ayuda para facilitar su usuario, el famoso *clip* de Microsoft Word ilustra esta táctica.

5. **Mantenimiento de un modelo de usuario:** aquí la táctica está enfocada hacia las acciones de usuario, el ejemplo estándar son los sistemas de lectura que avanzan páginas tan rápido como el usuario lee.

Tiempo de diseño

Como se ha comentado primeramente este grupo de tácticas se basan en el diseño de la interfaz de usuario y en la implicación máxima posible de este último cuando se está en tiempo de diseño. Existen algunos patrones de interfaces gráficas que facilitan tanto al usuario y al desarrollador el entendimiento sobre las mismas:

- Patrón Modelo Vista Controlador (*MVC*)

- Modelo Presentación / Abstracción / Control

- Modelo *Seeheim.*

- Modelo *Arco* / *Slinky*.

3.3.7. Relación de Tácticas y Patrones de arquitectura

Un patrón de arquitectura es la analogía de un estilo arquitectónico en la construcción de edificios, radica en una serie de reglas combinados de una manera coherente, tiene las siguientes características:

- Un conjunto de tipos elementales.
- Una organización, topológicamente hablando, de los elementos y sus relaciones entre ellos.
- Un conjunto de restricciones semánticas.
- Un conjunto de mecanismos de interacción / relación entre componentes, un ejemplo seria la utilización de llamadas a funciones.

La relación entre un patrón arquitectónico y una táctica es que, de forma inherente, un patrón implementa, o utiliza, varias tácticas para mejorar los atributos cualitativos. Por ejemplo el patrón Modelo Vista Controlador (*MVC*) mencionado anteriormente tiene la característica principal de dividir en tres componentes el software utilizado para comunicación con el usuario:

- **Modelo**: software orientado al *core* de la aplicación, es decir, de los cálculos y operaciones necesarias para procesar la información que necesita el usuario. Envía a la *vista* los datos procesados y recibe peticiones a través del *controlador*.
- **Vista**: capa que se le presenta al usuario y donde este introduce los datos para su procesamiento y se le presenta la información.
- **Controlador**: es el intermediario entre el *modelo* y la *vista*, responde peticiones de la vista realizando peticiones al modelo.

Como se ha visto anteriormente este modelo casa completamente con tácticas orientadas a la usabilidad, y también con tácticas de modificabilidad ya que se divide el software por responsabilidades haciendo que los cambies afecten a los menos componentes posibles.

4. ArchE

4.1. Qué es ArchE?

ArchE es un sistema experto desarrollado por el SEI (*Software Engineering Institute*) con un objetivo totalmente académico y pedagógico estando destinado a los profesores que quieran mostrar conceptos de arquitecturas a sus alumnos aunque también tiene un propósito que invita a la investigación ya que propone tácticas y da la posibilidad de aplicarlas para la mejora del diseño de arquitecturas.

La última versión, la utilizada en este proyecto, es la 3.0 liberada durante el año 2008. La aplicación permite relacionar los requisitos funcionales, atributos de calidad y la propia arquitectura para asegurar el cumplimiento de dichos atributos de calidad. En esta versión sólo se evalúan dos tipos de atributos de calidad: rendimiento y modificabilidad por medio de marcos de razonamiento (*reasonings frameworks* RF) de los que se hablará detalladamente más adelante.

Mediante el uso de esta herramienta se pueden medir el grado de cumplimiento de los atributos de calidad que se han comentado durante los puntos anteriores el cual utiliza tres tipos datos de entrada que el arquitecto debe proveer a la aplicación:

- Requisitos del sistema: en la herramienta se conocen como **funciones**.
- Relaciones: las relaciones entre las distintas funciones.
- Requisitos de calidad: se introducen como **escenarios** descriptivos.

La aplicación está "montada" sobre la popular herramienta de desarrollo **Eclipse**, y su instalador, que veremos en puntos siguientes, se instala de forma transparente como un *plugin* de dicha herramienta.

Antes de mostrar cómo funciona la herramienta internamente se exponen a continuación varios conceptos clave que se repetirán en posteriores lugares del documento:

- **Responsabilidades**: Son la representación de las funciones del sistema de las que se han apuntado anteriormente.
- ***Reasoning Frameworkf* (RF)**: son los también llamados en español "marcos de razonamiento", funcionan como una especie de herramienta externa a la aplicación y se encargan de medir o evaluar el cumplimiento de un determinado atributo de calidad.

- **Escenarios**: es la manera que tiene ArchE de representar los atributos de calidad, por medio de estos objetos los cuales tienen una medida y unos estímulos de entrada y salida.

- **Tácticas**: propuestas que realizan los RF para poder mejorar el cumplimiento de los atributos de calidad, van desde establecer nuevas relaciones entre las funciones a descomponer responsabilidad para aumentar la granularidad de los componentes de la arquitectura

4.2. Como funciona ArchE

ArchE es un sistema experto que trabaja en base a reglas "if-then-else" que actúan sobre un conjunto de datos que son los que el arquitecto introduce en el sistema, a partir de los datos de entrada y las reglas la herramienta proporciona una serie de tácticas que ayudan a cumplir los atributos de calidad representados como escenarios; el sistema proporciona información de cómo las tácticas actúan sobre todos los escenarios ya que, lo que puede beneficiar al cumplimento de uno puede desfavorecer con otros escenarios. Como sistema experto se eligió **Jess**, construido sobre Java, se verá su instalación en apartados posteriores.

Los datos que el arquitecto debe introducir dentro de ArchE serán los siguientes:

- Funciones: son los requisitos del sistema expresado con un nombre y una descripción.

- Responsabilidades: se tratan de las representaciones de las funciones. Normalmente tienen una relación 1:1 pero el sistema te permite, por ejemplo, crear una responsabilidad asociada a varias funciones.

- Escenarios: es un atributo de calidad representado en seis partes:

    - Estímulo
    - Fuente del estímulo
    - Entorno asociado
    - Artefacto asociado
    - Respuesta
    - Medida de la respuesta

  No tiene porqué estar las seis características completadas. Se da la posibilidad de que un mismo atributo de calidad tenga varios escenarios asociados, lo que hace más cercano a la realidad la evaluación.

- Relaciones Escenarios-Responsabilidades: es responsabilidad del usuario, en este caso el arquitecto, de proporcionar la relación entre los escenarios de la arquitectura y las responsabilidades. Está relación es N a N ya que una misma función puede afectar a varios escenarios y un mismo escenario tener que considerar varias responsabilidades, esté último caso sería el más común.

A partir de estos datos de entradas, gracias a los RF, ArchE podrá evaluar la arquitectura en base a los escenarios propuestos, mostrando al usuario una serie de tácticas a seguir y como afectarían a todos los escenarios. Para aplicar dichas tácticas se realiza mediante preguntas (en una vista de Eclipse).

Los marcos de razonamiento aparecen en la herramienta como "herramientas externas" que funcionan por si solas y se pueden añadir/quitar de la instalación; esto es curioso ya que sigue el principio indicado durante el documento de que "nadie es experto en todos los atributos de calidad", ya que de esta manera "especializa" al evaluador de atributos de calidad por herramienta. Cabe indicar que en la instalación de ArchE se instalan los RF de evaluación del rendimiento y de la modificabilidad.

A continuación se van a detallar todos estos conceptos para entender qué función tienen dentro de ArchE y como pueden ser utilizados de la mejor manera posible:

1) MARCOS DE RAZONAMIENTO

Son los "evaluadores reales" de la arquitectura en base a los atributos de calidad representados como escenarios. Como se ha registrado anteriormente son herramientas externas que se instalan como *plugins* en la aplicación.

ArchE lleva por defecto los RF de modificabilidad y rendimiento pero proporciona una API (*Reasoning Framework Interface*) para la comunicación de cualquier RF con ArchE. Mediante dicha interfaz se puede ser capaz de que un marco de razonamiento cualquiera se adapte y funcione dentro de ArhcE para recibir los datos de la arquitectura de este y devolver los resultados de la manera que ArchE los interpreta y se los muestra al usuario.

El funcionamiento de los *plugin* de Eclipse es muy sencillo, se colocan los ficheros dentro de la subcarpeta de eclipse llamada **plugins**. Los plugins normalmente se presentan como una carpeta con varias subcarpetas y ficheros. Cuando Eclipse, o ArchE en nuestro caso, arranca carga todo el software contenido en esa carpeta para su utilización.

En el caso de los RF se buscan en la pestaña "Vistas" de Eclipse y todas tendrán un botón de "Start" y de "Stop".

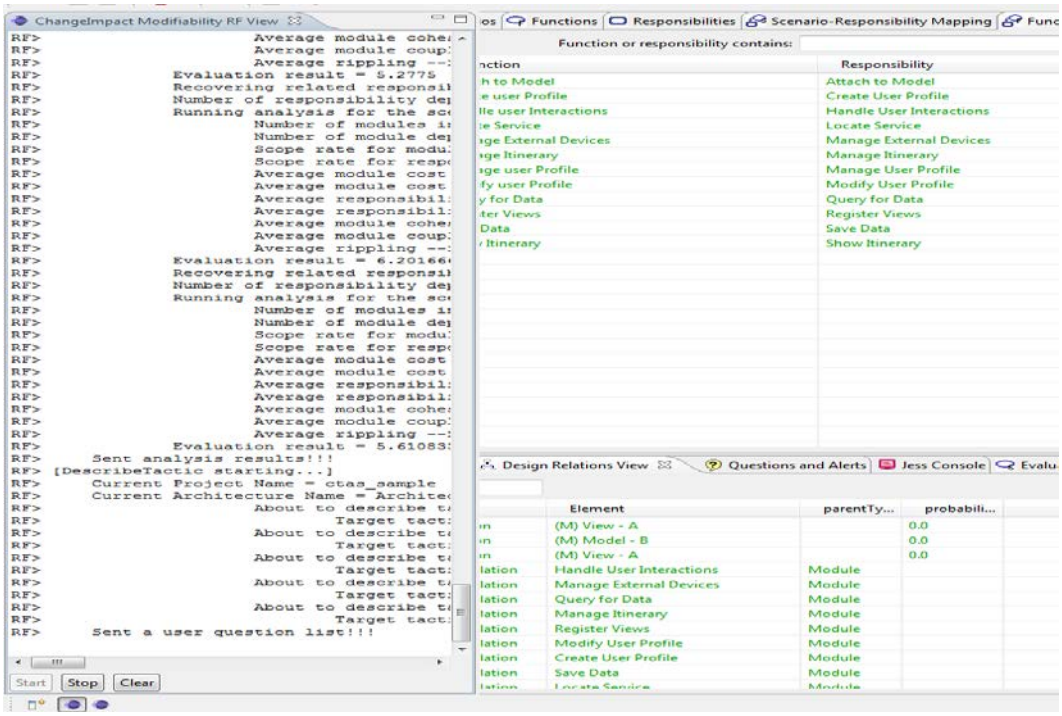


Ilustración 1: Ejemplo de RF ArchE

2) RESPONSABILIDADES

Una responsabilidad es una función que debe realizar la arquitectura. Por ello se apuntaba anteriormente que había normalmente una relación 1 a 1 entre funciones y responsabilidades en ArchE. Los RF evalúan escenarios y estos están relacionados con responsabilidades así que en el extremo final son estas últimas las que se utilizan para valorar su grado de cumplimiento.

Teóricamente las responsabilidades deberían ir asignadas a módulos, con una relación 1 a n, donde un módulo puede tener varias responsabilidades y una responsabilidad sólo puede estar asignada a un módulo. ArchE no permite la creación de módulos por parte del usuario, si lo permite a base de código fuente, por lo que realiza una asignación interna 1 a 1, para que finalmente los RF evalúen los módulos de forma transparente al usuario que piensa que lo que se evalúa son los escenarios y las responsabilidades.

Aunque internamente los RF revisen como concepto clave los módulos las responsabilidades también son claves para sacar conclusiones. Esto es porque muchas veces las tácticas propuestas por los RF tienen que ver con las responsabilidades, sobre todo aplicando la estrategia del “divide y vencerás”, descomponiendo una responsabilidad en varias para acercar a

la arquitectura al objetivo final de cumplir en el mayor grado posible con el cumplimiento de los atributos de calidad apreciados.

3) TÁCTICAS

Una táctica es una serie de acciones que propone el RF para mejorar la arquitectura en base al atributo de calidad asociado. Aparecen como preguntas en la vista de la aplicación *Question*, que proponen al usuario la división de responsabilidades en varias o asignar valores en los parámetros de responsabilidades y a las relaciones entre ellas.

4) VISTAS

En una perspectiva que ofrece la herramienta para visualizar la arquitectura de forma gráfica. Aparecen en forma de grafo las responsabilidades y las relaciones entre ellas; ofrece al arquitecto una visión esquemática de las dependencias de cada responsabilidad entregándole otro punto de vista para poder organizar mejor la arquitectura.

5) ESCENARIOS

Un escenario es un concepto de ArchE que representa a un atributo de calidad. En esencia esa una expresión, que se describe con un texto especificando una acción, que relaciona a un atributo de calidad con la arquitectura, ejemplo: *el tiempo de reacción a un error debe ser menor de 10 segundos.*

Además del texto descriptivo se deben asignar valores a seis propiedades, que son las siguientes:

- Origen del estímulo que produce el escenario.
- Estímulo que produce el escenario.
- Artefacto impactado por el estímulo
- Entorno en el que se produce el escenario.
- Respuesta del artefacto al estímulo
- Medida de la respuesta del artefacto al estímulo.

Normalmente existen varios escenarios descritos para cada atributo de calidad, lo que hará que los RF tengan que tener en cuenta más variables a la hora de proponer tácticas al usuario.

4.3. Proyecto CTAS ArchE

CTAS (*Clemson Traveler Assitant System)* es una aplicación que ofrece a sus usuarios la posibilidad de planificar sus itinerarios de ruta. Las opciones que permite son muy variadas, dándole prioridad al modo de transporte, a la duración de viaje, etc.. Esta aplicación va dirigida a varios grupos de usuarios tales como responsables gubernamentales, empresas o particulares para sus propios viajes. La aplicación se actualiza automáticamente con datos de las rutas y los medios de transporte.

Ya que los usuarios a los que va destinado son de distintos tipos cada uno tendrá unos objetivos y unas necesidades distintas que les induce a utilizar este sistema, a continuación se relatan cuáles serían:

- Usuarios Particulares: normalmente sus necesidades son que el sistema sea fácil y claro de utilizar, ya que sus viajes se suponen que son "por placer" en teoría sus necesidades se cierran aquí.

- Empresarios: estos lo que requieres es tener varias opciones de viajes y que haya la mayor información posible, sobre todo relacionada con los costes para poder elegir la más económica.

- Políticos: a este grupo de usuario lo que más le preocupa es llegar puntual a sus citas así que la aplicación debería proveer de información referente a congestiones de tráfico o posibles retrasos en otros medios de transporte, así como ofrecer alternativas si esto se produce.

Como ya se ha comentado en partes anteriores del documento al realizar una arquitectura además de los requisitos de los usuarios finales del sistema hay otros actores involucrados de los que es muy importante recabar sus intereses para construir de la mejor manera la arquitectura:

- Desarrolladores: este grupo lo que requiere es construir la arquitectura de la forma más procedural, siguiendo una metodología conocido y utilizando una tecnología familiar al equipo de trabajo y que sea de una calidad contrastada en el mercado.

- Proveedores de información: en este grupo se pueden incluir tanto los gobiernos o empresas responsables del transporte como los empresarios con hoteles o

similares. Este grupo requiere sobre todo una actualización lo más rápida posible de los datos para que el sistema esté lo más actualizada posible para sus potenciales clientes.

En los puntos posteriores se explicará el uso de la herramienta ArchE sobre el ejemplo de CTAS, será en ese punto cuando se muestren los requisitos, responsabilidades y demás entresijos de la arquitectura asociada a dicho sistema.

4.4. Instalación ArchE

En este apartado se realiza a modo de manual para la instalación de la herramienta ArchE así como la configuración, ejecución y puesta en marcha de la misma con el proyecto CTAS que servirá como arquitectura base de estudio para esta memoria.

ArchE es una aplicación que se instala por sí misma pero para su funcionamiento efectivo necesita la instalación previa de varias aplicaciones, a continuación se detallan por orden dichos paquetes de dependencias de las que algunas se han mentado en apartados anteriores:

1. **Java JDK**

   La **JDK** *(Java Develpment Kit)* es el software que provee la compañía Oracle para poder desarrollar aplicaciones con el lenguaje Java, también de la misma propiedad. Este paquete ofrece tanto como compilador, intérprete o una herramienta para la generación de documentación en base a comentarios en el código y *tags*.

   Se ha elegido la versión JDK 7 por ser la última, pero cualquier versión a partir de la 6 en adelante valdría. A continuación se muestran unas capturas de pantalla de la instalación que se realiza ejecutando el fichero del enlace anterior:

   En la página se da a elegir entre varias opciones según el sistema operativo:

Java SE Development Kit 7u25

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM v6/v7 Soft Float ABI	65.12 MB	jdk-7u25-linux-arm-sfp.tar.gz
Linux x86	80.38 MB	jdk-7u25-linux-i586.rpm
Linux x86	93.12 MB	jdk-7u25-linux-i586.tar.gz
Linux x64	81.46 MB	jdk-7u25-linux-x64.rpm
Linux x64	91.85 MB	jdk-7u25-linux-x64.tar.gz
Mac OS X x64	144.43 MB	jdk-7u25-macosx-x64.dmg
Solaris x86 (SVR4 package)	136.02 MB	jdk-7u25-solaris-i586.tar.Z
Solaris x86	92.22 MB	jdk-7u25-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	22.77 MB	jdk-7u25-solaris-x64.tar.Z
Solaris x64	15.09 MB	jdk-7u25-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	136.16 MB	jdk-7u25-solaris-sparc.tar.Z
Solaris SPARC	95.5 MB	jdk-7u25-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.05 MB	jdk-7u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.67 MB	jdk-7u25-solaris-sparcv9.tar.gz
Windows x86	89.09 MB	jdk-7u25-windows-i586.exe
Windows x64	90.66 MB	jdk-7u25-windows-x64.exe

Ilustración 2: Descarga Java

Para el ejemplo que se ha tenido en cuenta en esta memoria se elige la versión *Windows x86*. La instalación es típica de Windows donde tan sólo hay que elegir la ruta para dejar los ficheros, también instala el componente JRE *(Java Runtime Enviroment)* que es el entorno en tiempo de ejecución de Java. Es muy importante quedarse con la ruta de instalación de la JDK ya que es imprescindible para el siguiente paso.

Se obliga a crear la variable de entorno JAVA_HOME dentro de Windows y a modificar la variable PATH, se ve en las siguientes capturas de pantalla:

Primero se accede al Panel de Control de Windows:

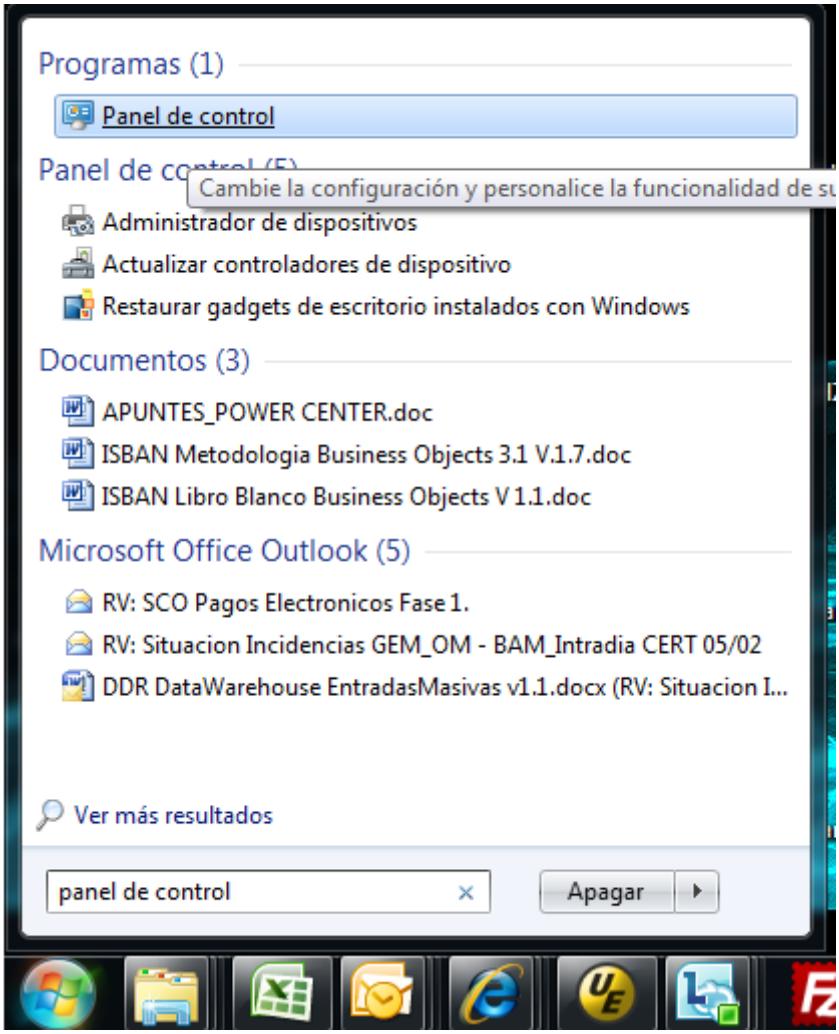


Ilustración 3: Búsqueda Panel de Control

Y una vez dentro se accede a la parte de Sistema:

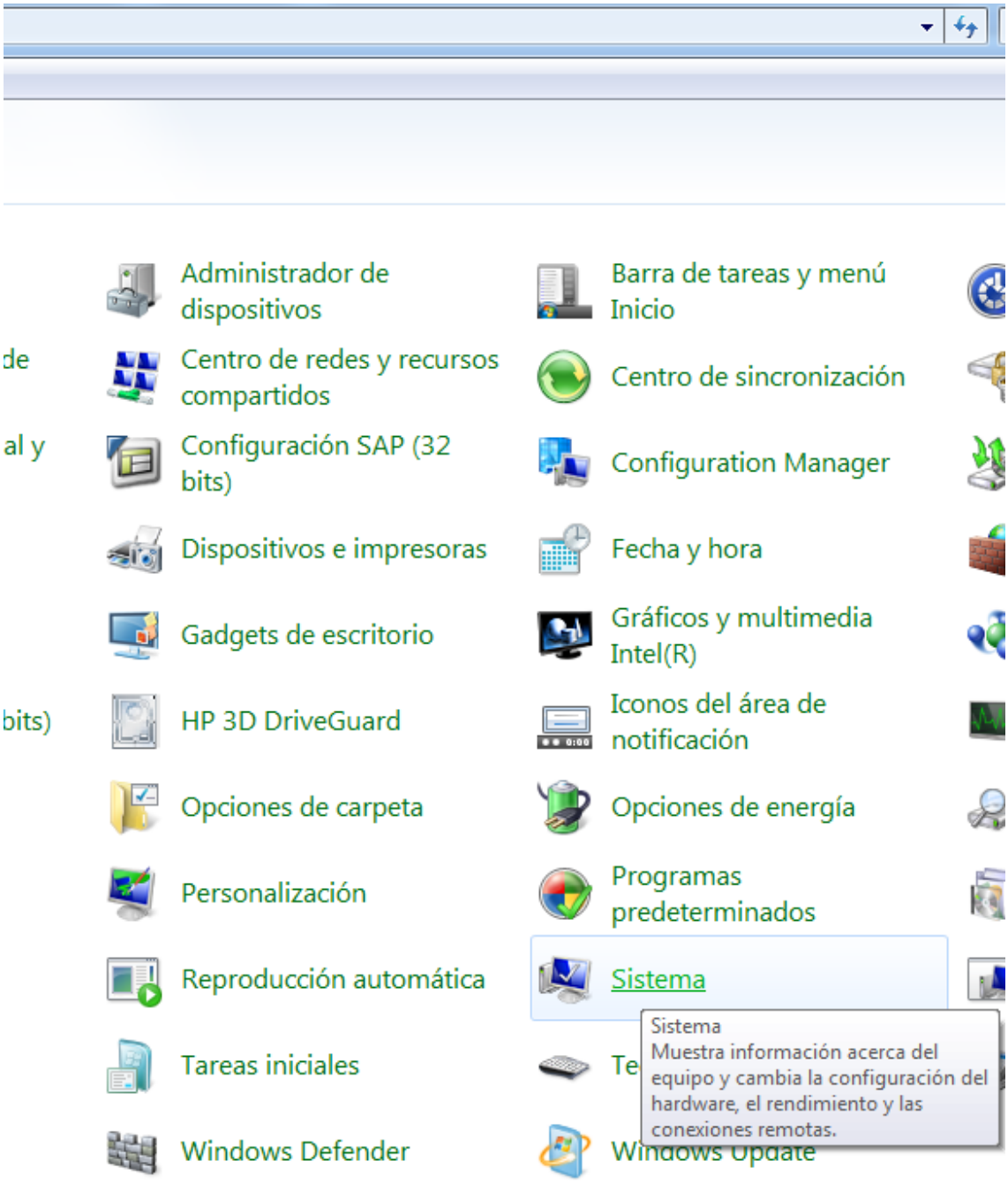


Ilustración 4: Panel de Control

Una vez dentro se accede a la parte de “Configuración avanzada del sistema”

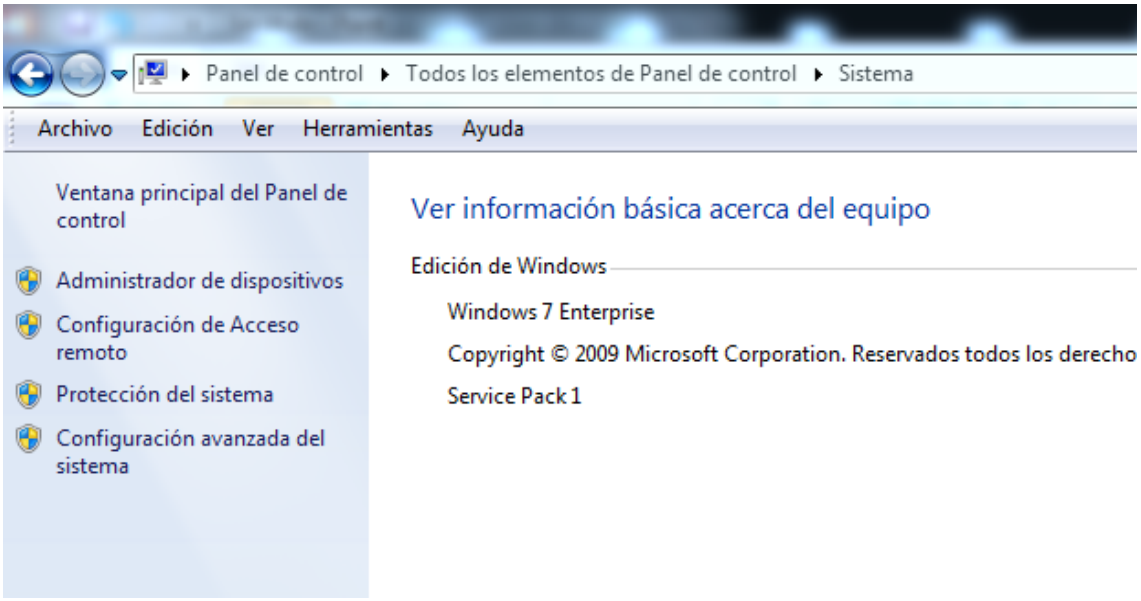


Ilustración 5: Panel de Control - Sistema

Aparecerá una ventana emergente tal como la siguiente:

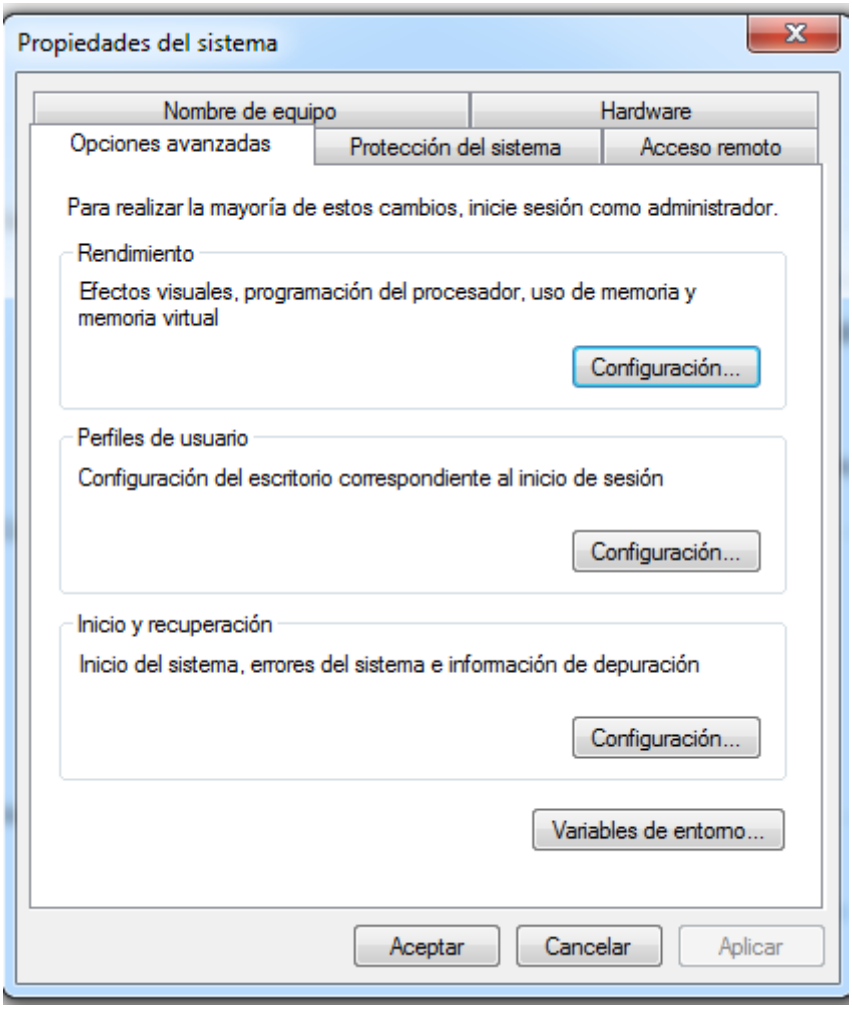


Ilustración 6: Propiedades del Sistema

Dentro de la misma se accederá a la opción “Variables de entorno”.

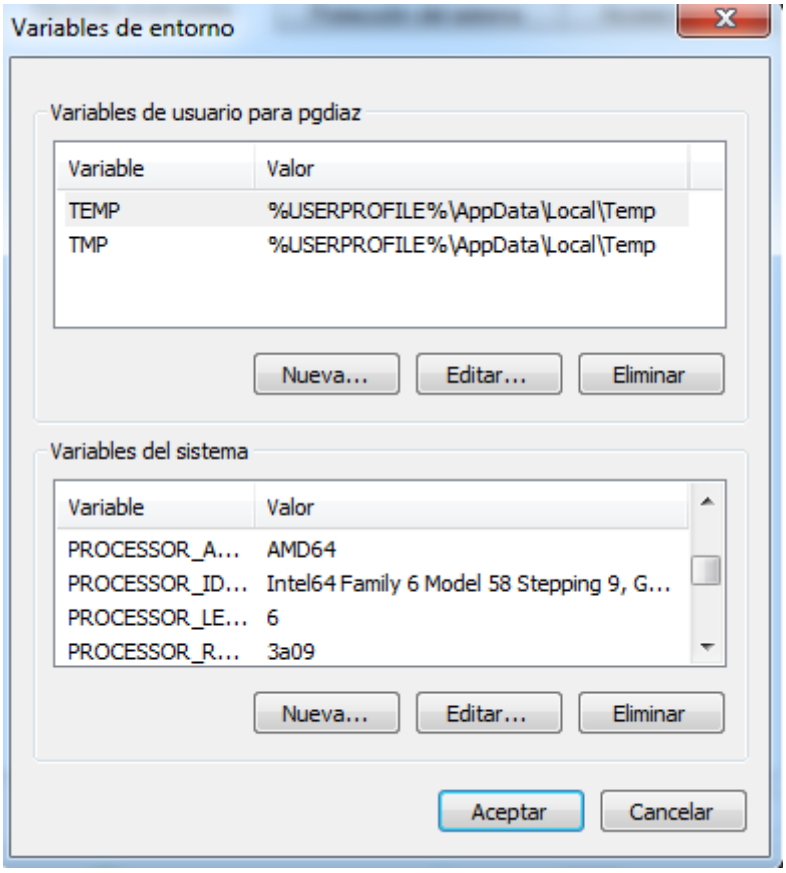


Ilustración 7: Variables de Entorno

Se selecciona la opción “Nueva” dentro del cuadro “Variables del Sistema” para crear una nueva variable de entorno dentro de la computadora.

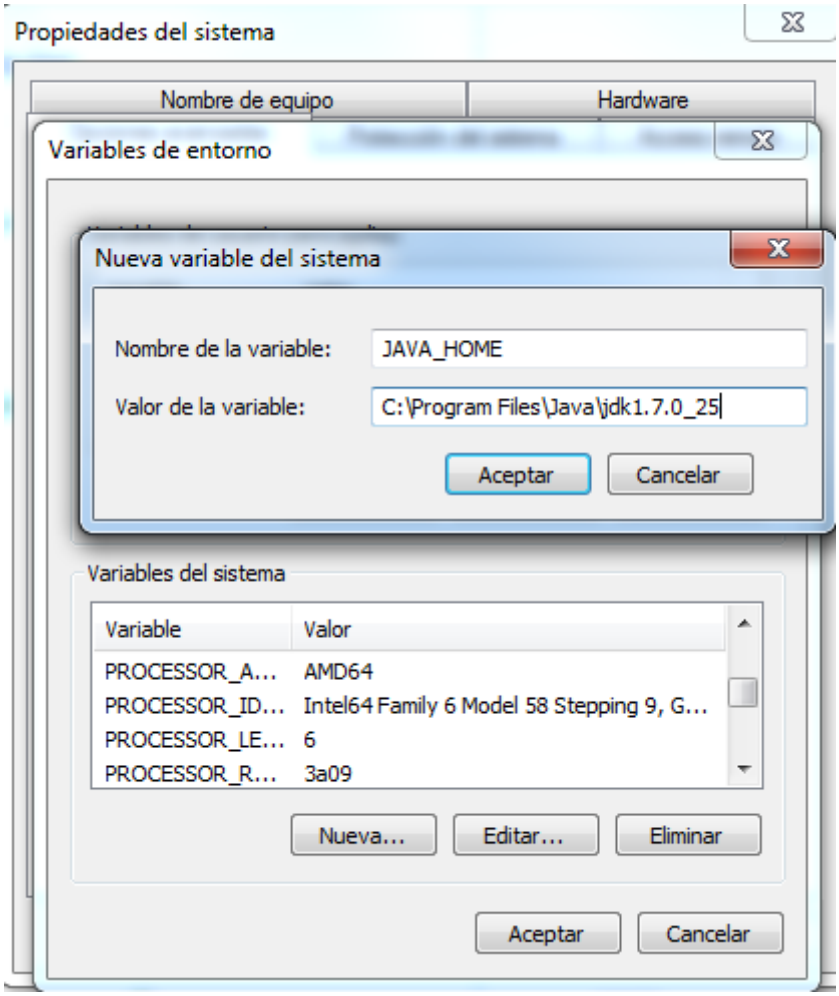


Ilustración 8: Insertar Variable de Entorno

Se le da el nombre de “JAVA_HOME” y la ruta donde se instaló la JDK, se termina la operación pulsando el botón “Aceptar”. Una vez realizado se procede a modificar la variable de entorno “Path” para que los ejecutables de la JDK se puedan ejecutar desde cualquier ruta sin hacer referencia a su situación absoluta o relativa, dentro de la misma pantalla:

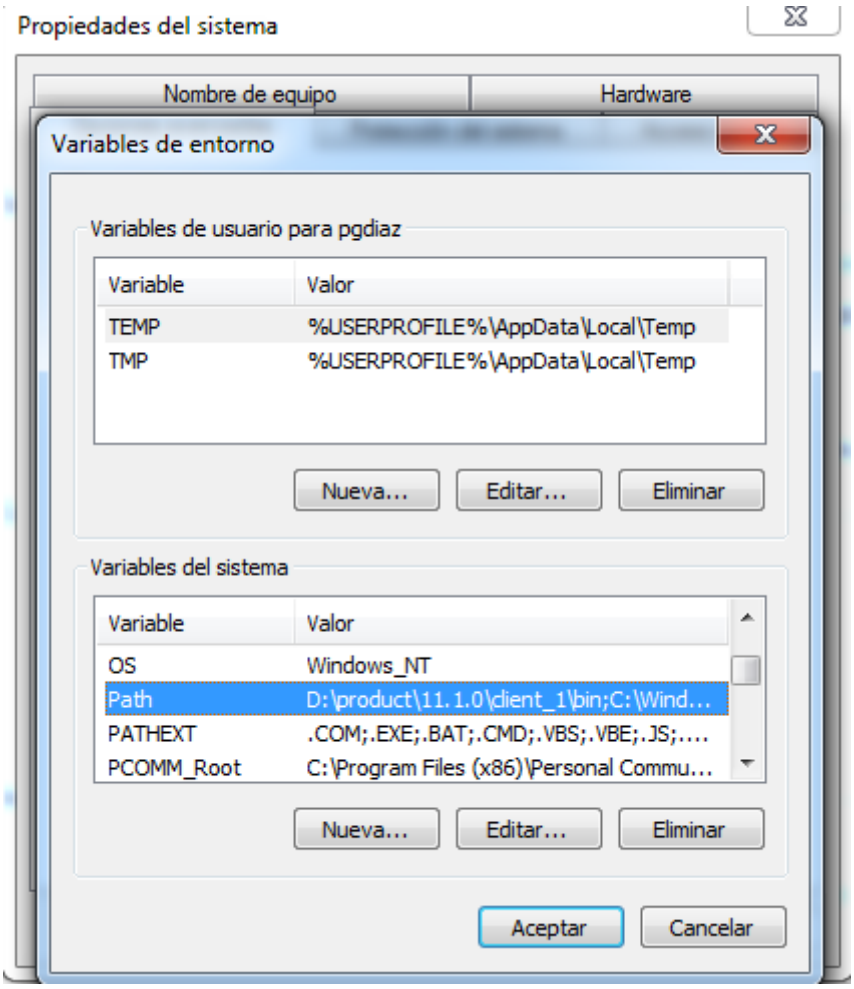


Ilustración 9: Editar Variable de Entorno 1

Se selecciona y se pulsa “Editar”.

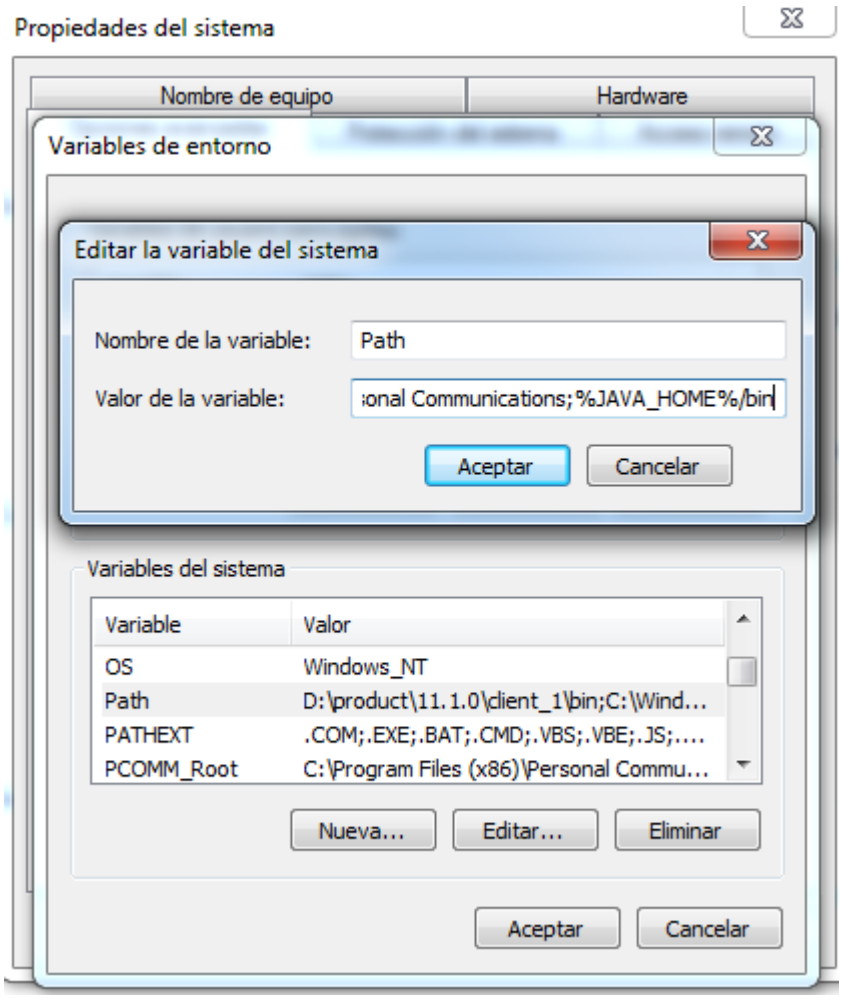


Ilustración 10: Editar Variable de Entorno 2

Se añade la cadena al final ";%JAVA_HOME%/bin", independientemente de la ruta de la JDK ya que mediante los caracteres "%" se accede al valor de la variable que se ha creado justo antes.

2. Eclipse

Eclipse es una aplicación formada por un conjunto de herramientas con licencia pública, en su mayoría de código abierto, para el desarrollo de aplicaciones de cliente enriquecido. Es la plataforma más popular para desarrollos con el IDE de Java.

Bajo esta aplicación se han desarrollado otras herramientas mediante la construcción de los llamados "plugins", que actúan como módulos que se ejecutan bajo esta plataforma.

Por recomendación se instalará la versión SDK-3.3-win32, al ser una versión que no tiene un instalador típico de Windows se puede descomprimir el contenido dentro de la carpeta que se estime oportuno.

3. MySql

Es un gestor de base de datos propiedad de la compañía *Oracle* con licencia de software libre, es multiplataforma y soporta bases de datos relacionales, multihilo y multiusuario.

Se seguirá el siguiente enlace Mysql-essential-5.067-win32.msi, se ejecutará el archivo de descarga y aparecerán las siguientes pantallas típicas de una instalación estándar de Windows:

Ilustración 11: Instalación MySql 1

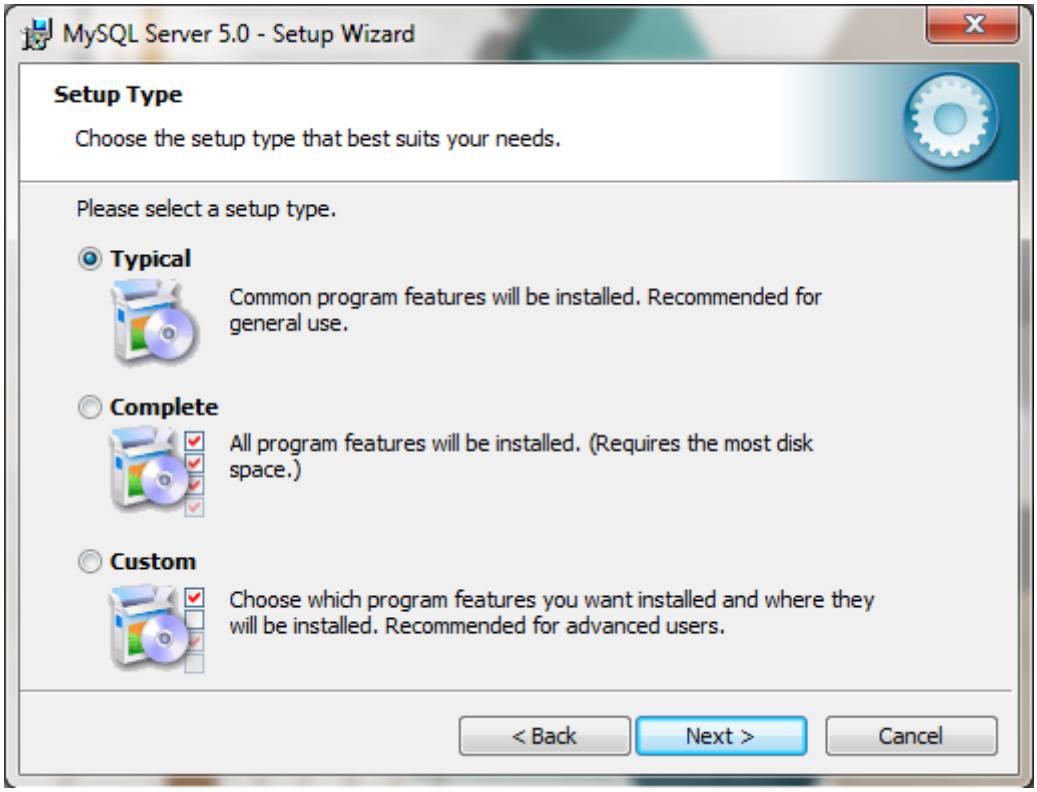


Ilustración 12: Instalación MySql 2

Se elige la instalación típica ya que no hace falta ningún paquete o configuración especial en este punto.

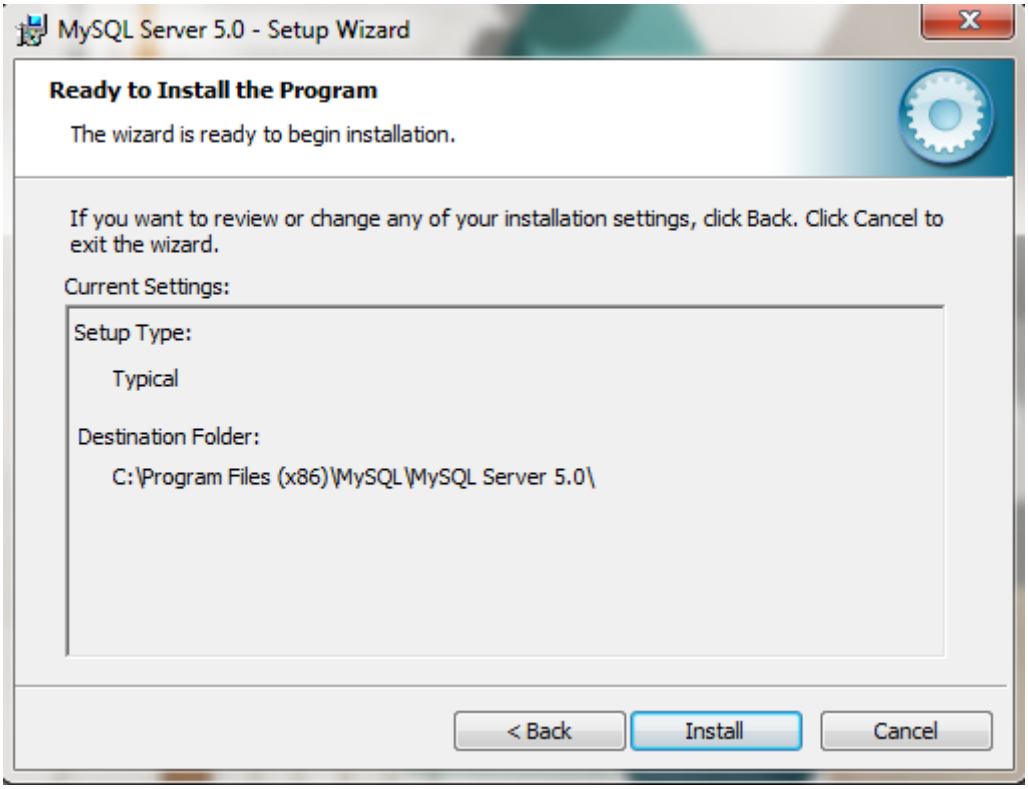


Ilustración 13: Instalación MySql 3

Se pulsa *Install*.

Y se esperará su finalización, no hará falta ninguna configuración especial ya que al instalar ArchE parametrizará la base de datos adecuadamente.

4. GEF

Es un *framework* para Eclipse que sirve para crear editores con una gran variedad de gráficos y vistas. La versión utilizada se encuentra en el siguiente enlace.

La herramienta está paquetizada como un típico plugin de eclipse, el archivo descargado se trata de un .zip que habrá que descomprimirlo en la carpeta **superior** a la que se tenga el Eclipse.

5. JESS

JESS es un lenguaje de programación basado en reglas para la plataforma Java desarrollado por Ernest Friedman-Hill de Sandia National Labs. Proporciona lo básico para la automatización de un sistema experto.

Para su instalación se tiene que descomprimir el adjunto en una carpeta cualquiera, una vez estén los archivos descomprimidos se debe crear una variable de entorno y añadirla a la variable "Path" de la misma manera que se ha descrito en el punto 1; la variable se llamará JESS_HOME y tendrá el valor **carpeta descompresión** y se incluirá en el Path esa misma variable con el sufijo **/bin**.

Además se deben copiar todos los ficheros existentes dentro de **JESS_HOME/eclipse** y copiarlos a la carpeta **carpeta_eclipse/plugins**.

6. xmlBlaster

xmlblaster es un *Middleware* orientado al intercambio de mensajes con las siguientes características más destacadas:

- Basado en MOM Java que intercambia mensajes con protocolo editor / suscriptor.
- Los mensajes son codificados por medio del lenguaje XML.
- Acepta casi cualquier formato en la transmisión, objetos Java, ficheros planos, imágenes, documentos Office, etc...
- La comunicación con el servidor se puede establecer por medio de CORBA, RMI, XmlRpc, HTTP o correo electrónico para que el cliente elija el protocolo que más le convenga.

ArchE necesita esta herramienta para el intercambio de mensajes y su instalación es independiente de los demás componentes, simplemente se debe descargar el fichero comprimido en la ruta que se deseé. Una vez realizado se obliga a crear la variable de entorno, acción ya explicada en el punto 1, XMLBLASTER_HOME con valor **carpeta descompresión/xmlBlaster**.

Para su posterior ejecución se aconseja la descarga del siguiente fichero arranca_xmlblaster.bat a la ruta que se deseé; en el punto posterior se explicará cómo utilizarlo.

7. ArchE

La instalación propia de ArchE se realiza mediante un archivo con extensión *.exe* que abre unas pantallas típicas de una instalación Windows estándar que se encuentra

comprimido dentro del siguiente archivo .zip. A continuación aparecerán las siguientes pantallas:

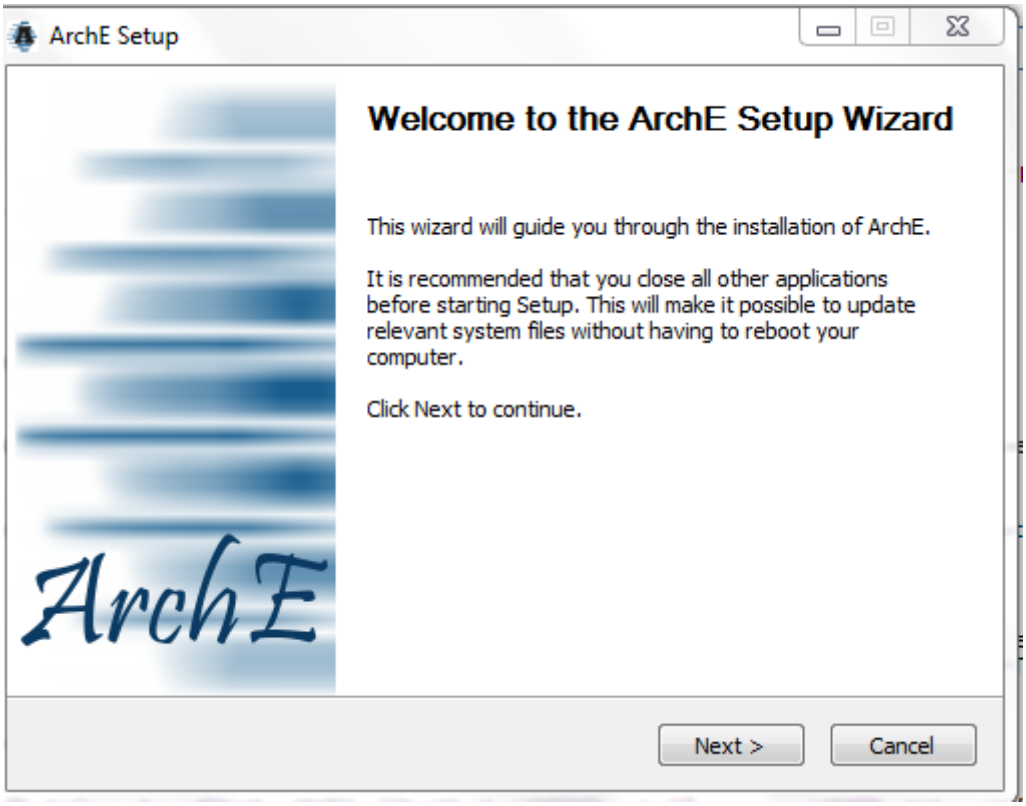


Ilustración 14: Instalación ArchE 1

Pulsando *Next* se pasa a la siguiente pantalla:

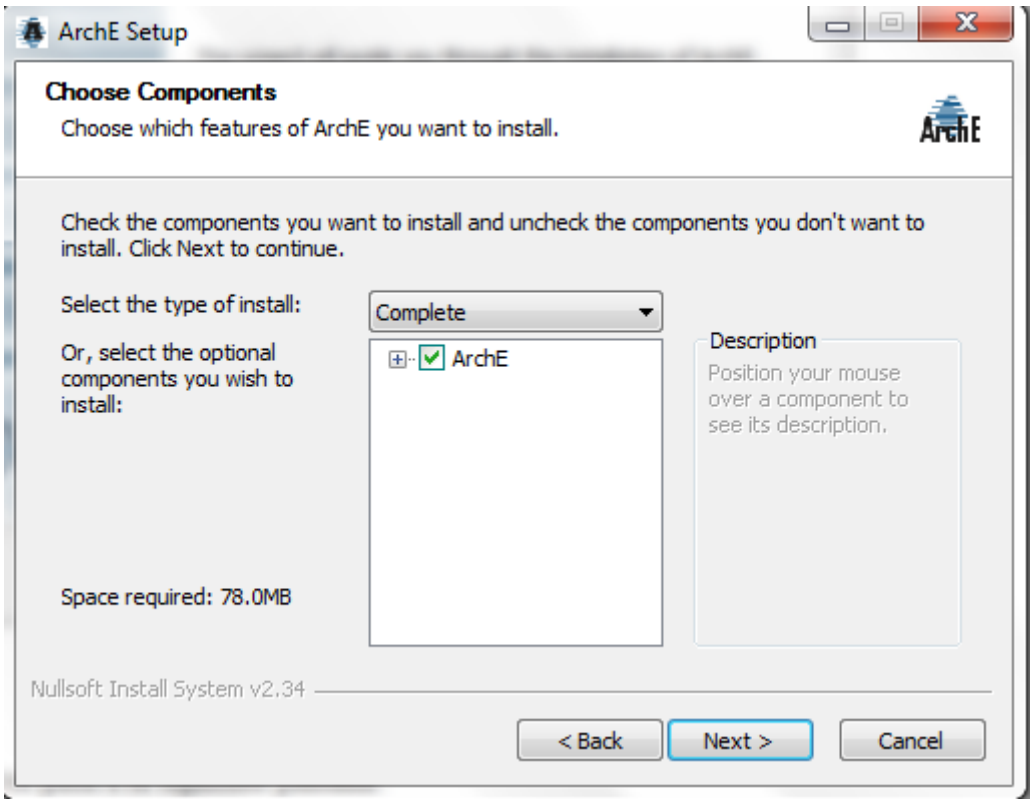


Ilustración 15: Instalación ArchE 2

Aunque exista la opción de una instalación personalizada, *Custom*, realmente el instalador obliga a una instalación completa, se selecciona *Next* y se avanza al siguiente punto:

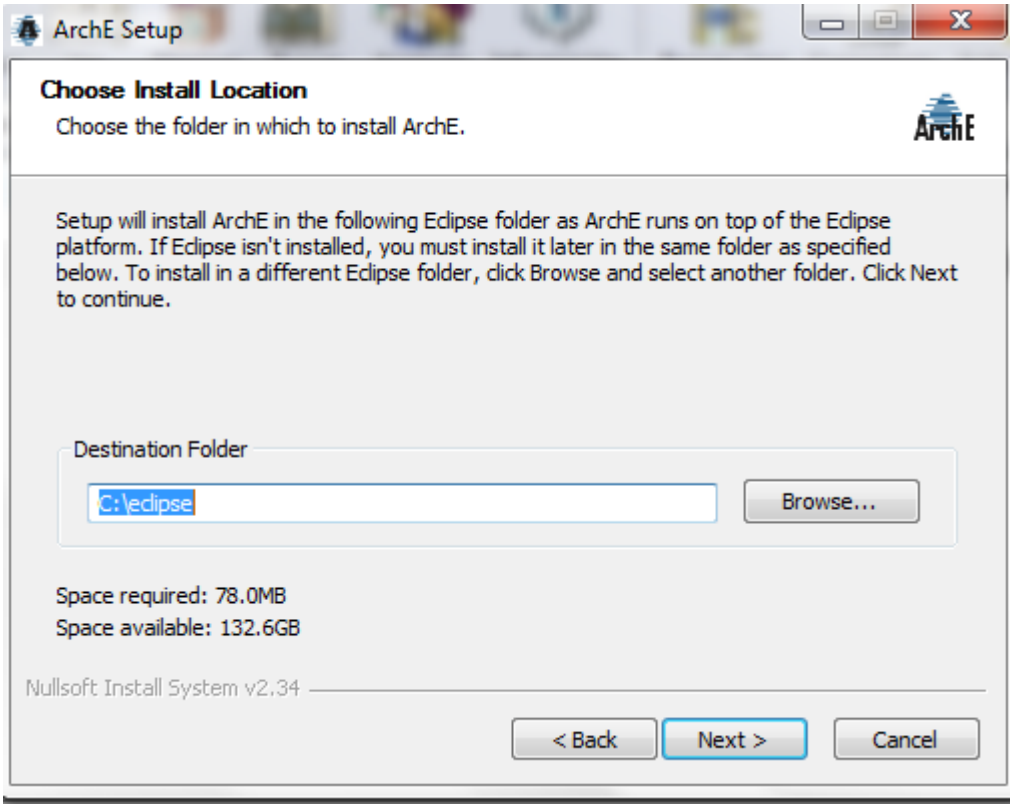


Ilustración 16: Instalación ArchE 3

Se debe seleccionar el directorio donde se hubiere realizado la instalación previa de la aplicación Eclipse para que ArchE ponga los archivos necesarios sobre esa ruta, como en las demás pantalla se debe pulsar *Next*:

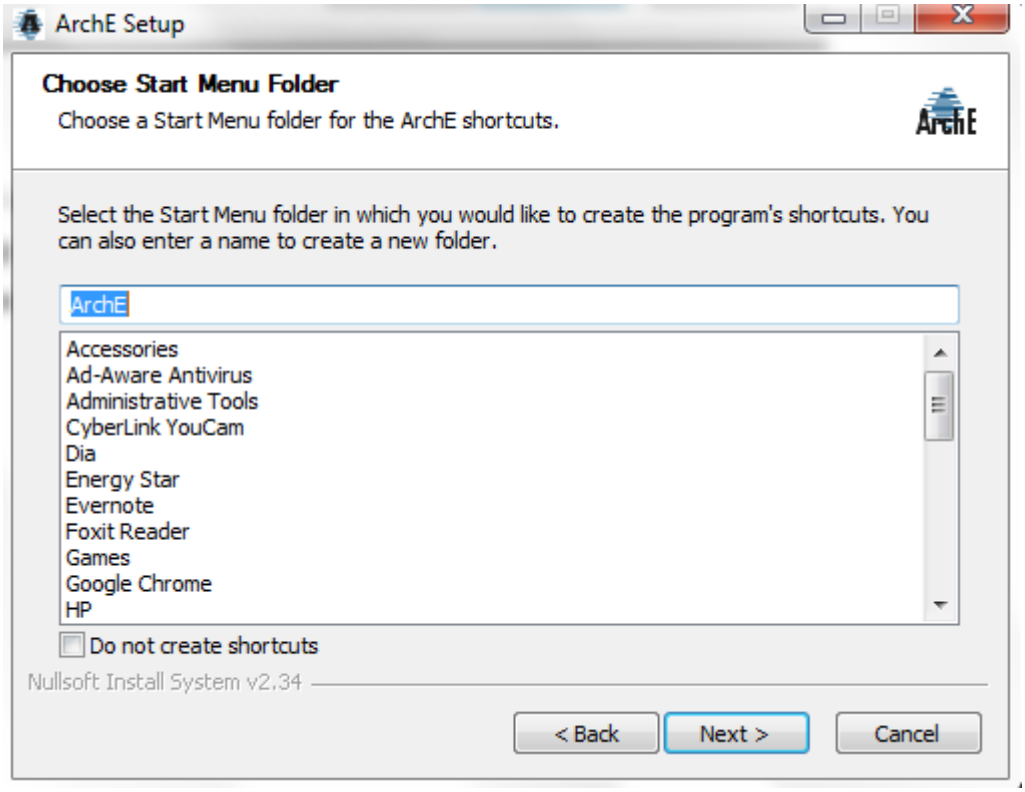


Ilustración 17: Instalación ArchE 4

Pantalla que simplemente sirve para poner el nombre en el menú Programas de Windows, está sería la siguiente pantalla de la instalación:

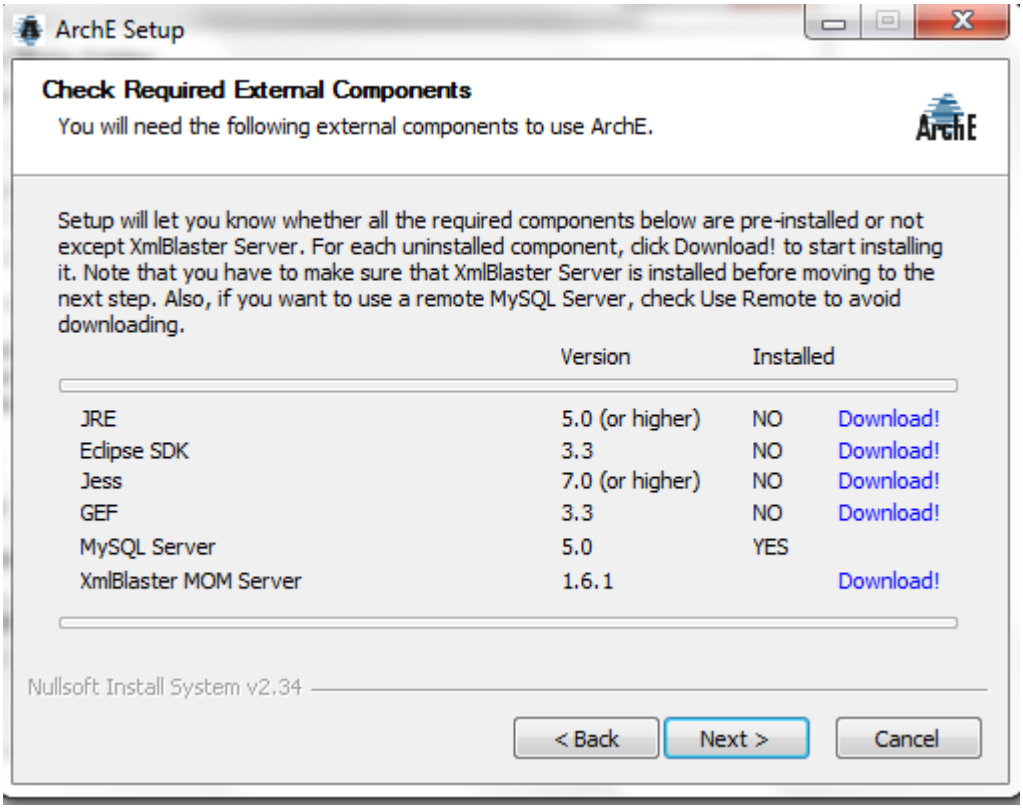


Ilustración 18: Instalación ArchE 5

Aquí el instalador comprueba que están instaladas todas las herramientas dependientes comentadas anteriormente en esta guía, tan sólo habría que instalarlas, incluso el instalador añade unos links (*Download!*) para descargar las herramientas, y seguir adelante para finalizar la instalación ArchE, se crea un acceso directo en el escritorio de esta manera:

4.5. Guía de Usuario

En este espacio se expone una especie de pautas para que el usuario pueda distinguir las diferentes opciones de las que dispone ArchE así como las bases para reconocer las vistas o perspectivas de la aplicación. Se explicará tomando como base el proyecto CTAS.

Antes de ejecutar ArchE es obligatorio tener corriendo en la máquina la aplicación *xmlBlaster*, para ello se ejecuta el archivo arranca_xmlblaster.bat mentado en el punto anterior, que hará saltar una ventana del antiguo MS-DOS ahora llamado *Símbolo del sistema.*

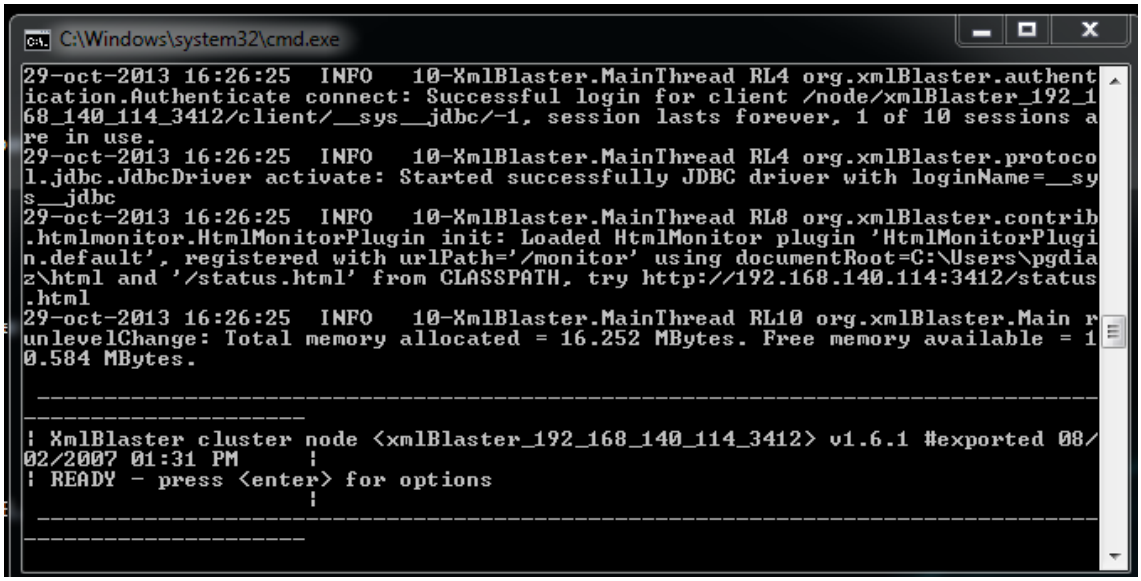


Ilustración 19: Arranque xmlblaster

Como se puede apreciar al final del pantallazo la aplicación pide al usuario una opción para su ejecución, se puede pulsar directamente ENTER para ver las opciones de las que dispone.

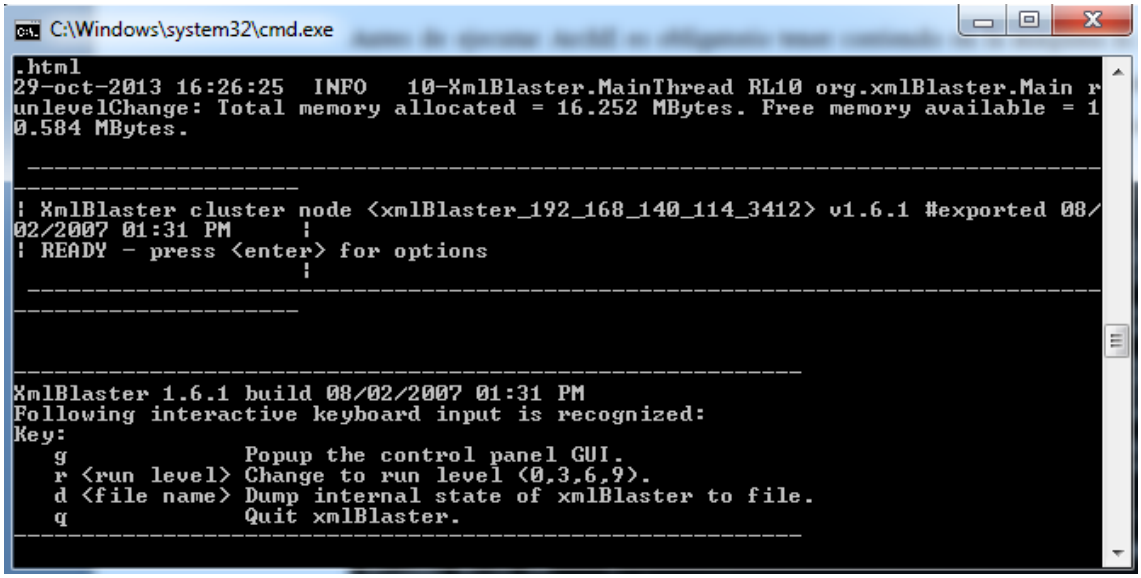


Ilustración 20: Opciones xmlBlaster

Para asegurar que está bien arrancado se activa la opción “r”, simplemente introduciendo la letra *r* y pulsando ENTER. Aparecerá un texto similar al de la próxima ilustración asegurando que la aplicación se está ejecutando correctamente.

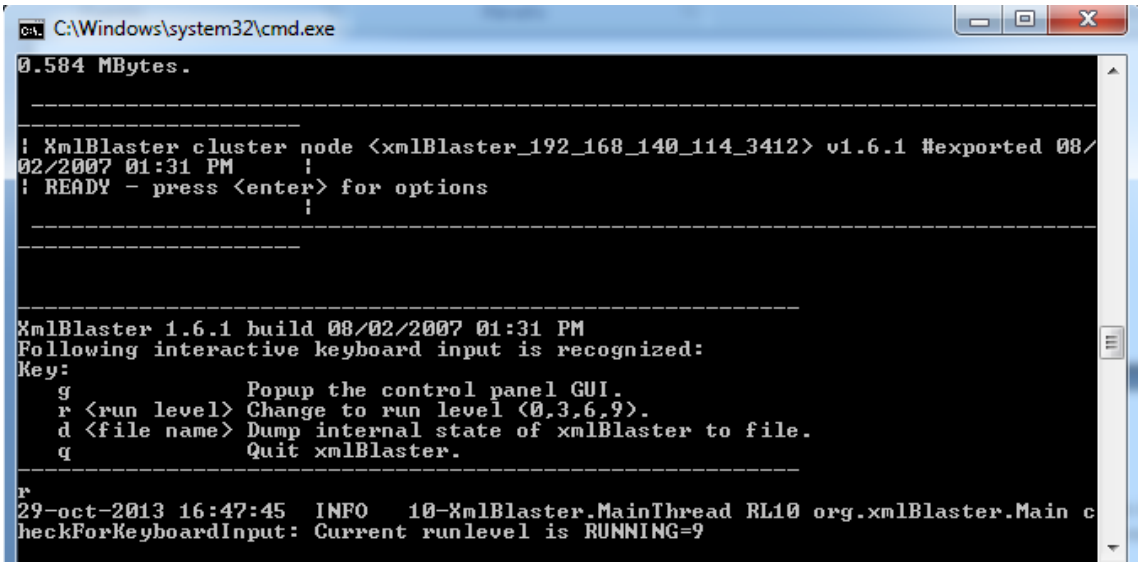


Ilustración 21: Ejecución xmlBlaster

Se abre la aplicación ArchE haciendo doble-click sobre el icono, si es la primera ejecución la aplicación pedirá una ubicación donde ubicar los ficheros correspondiente al *workspace*, es un espacio que típicamente utiliza Eclipse para guardar los ficheros de configuración.

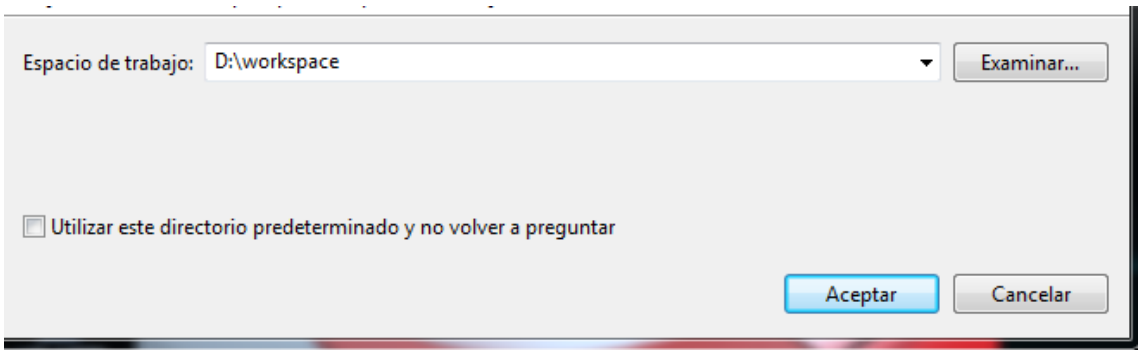


Ilustración 22: Ubicación Workspace ArchE

Seguidamente se muestra la pantalla típica de una explicación Eclipse sin ningún proyecto, llena de pantallas vacías. Para crear el primer proyecto de ArchE se eligen las siguientes opciones:

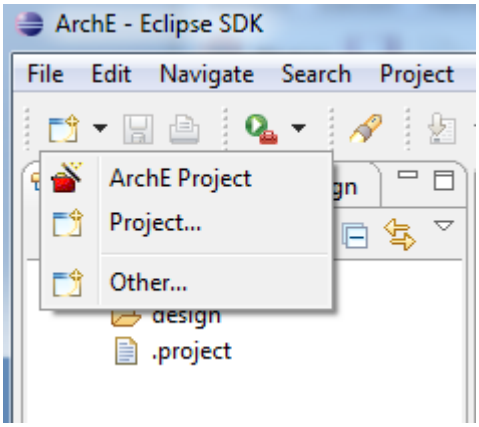


Ilustración 23: Nuevo Proyecto ArchE 1

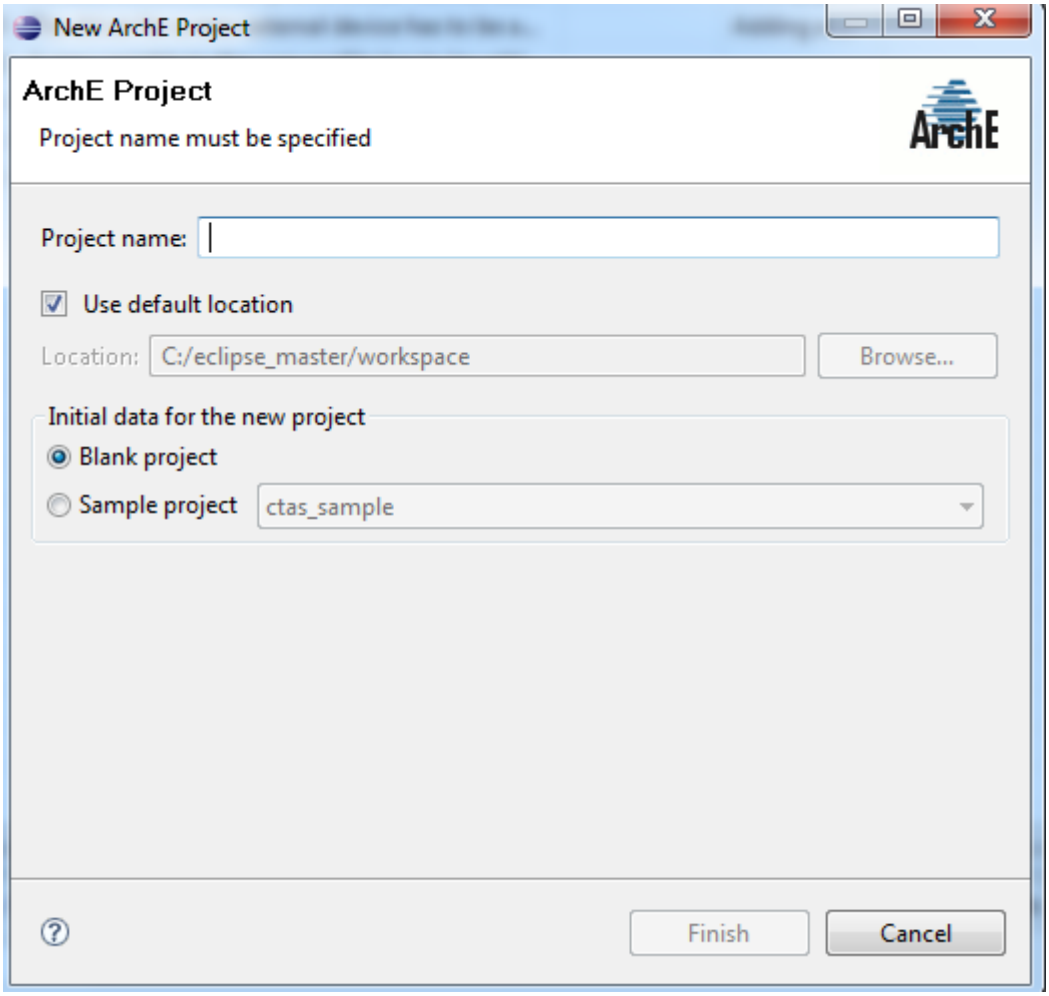


Ilustración 24: Nuevo Proyecto ArchE 2

Se elegirá para el ejemplo seguido del documento la opción de *Sample Project*, la cual creará una ruta de archivos en el workspace que se podrán visualizar con el explorador de ArchE.

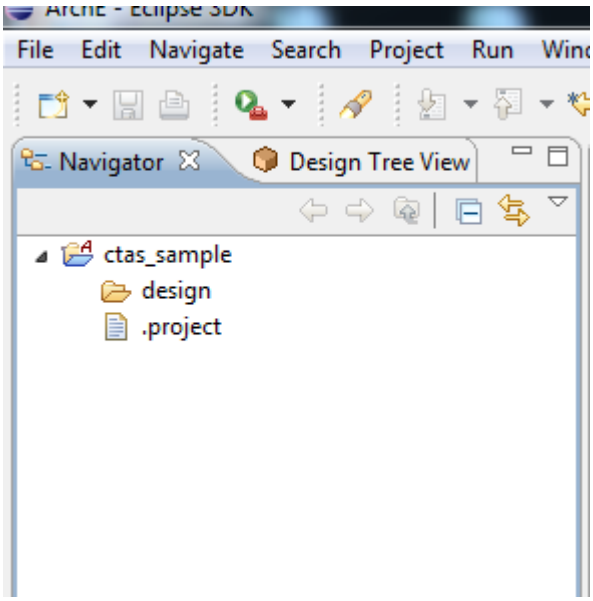


Ilustración 25: Navigator ArchE

En la vista colocada lado *Design Tree View* también se pueden apreciar los modelos del proyecto.

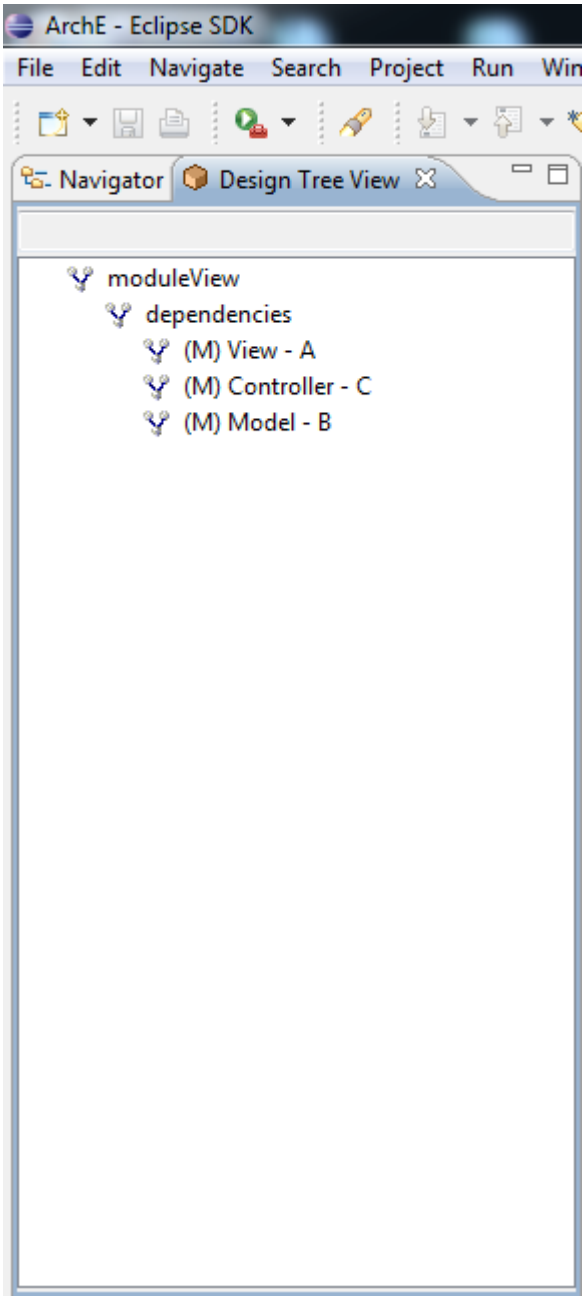


Ilustración 26: Vista de árbol ArchE

A primera vista se puede dividir la pantalla de ArchE en tres partes:

- Parte superior izquierda: vistas de ficheros y árbol de módulos mentados anteriormente.

- Parte superior central y derecha: vistas donde el usuario puede introducir datos, como escenarios, responsabilidades, relaciones…

- Parte inferior: vistas informativas para el arquitecto, aquí se mostrarán tanto resultados de los RF como las posibles tácticas a seguir.

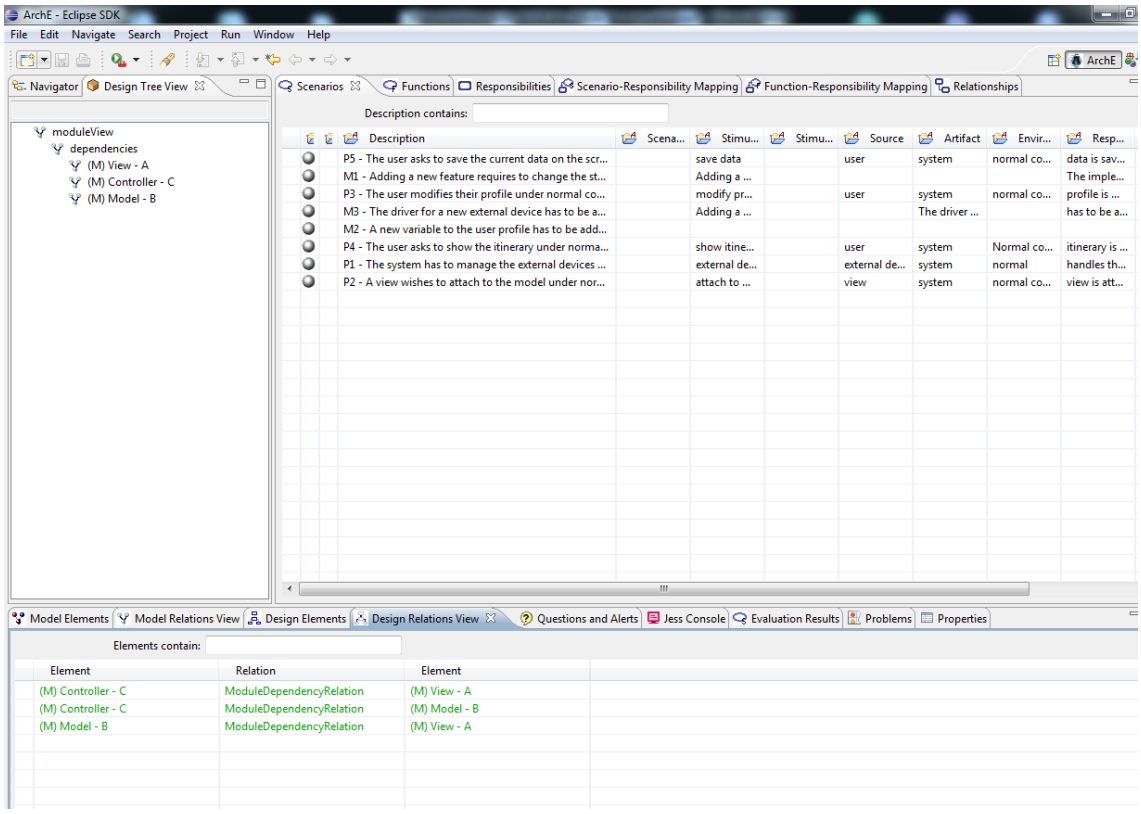


Ilustración 27: Vista General ArchE

En el caso en cuestión ya aparece rellena de datos porque se ha utilizado el proyecto de ejemplo.

A continuación se repasarán pantalla por pantalla todo lo que ofrece ArchE.

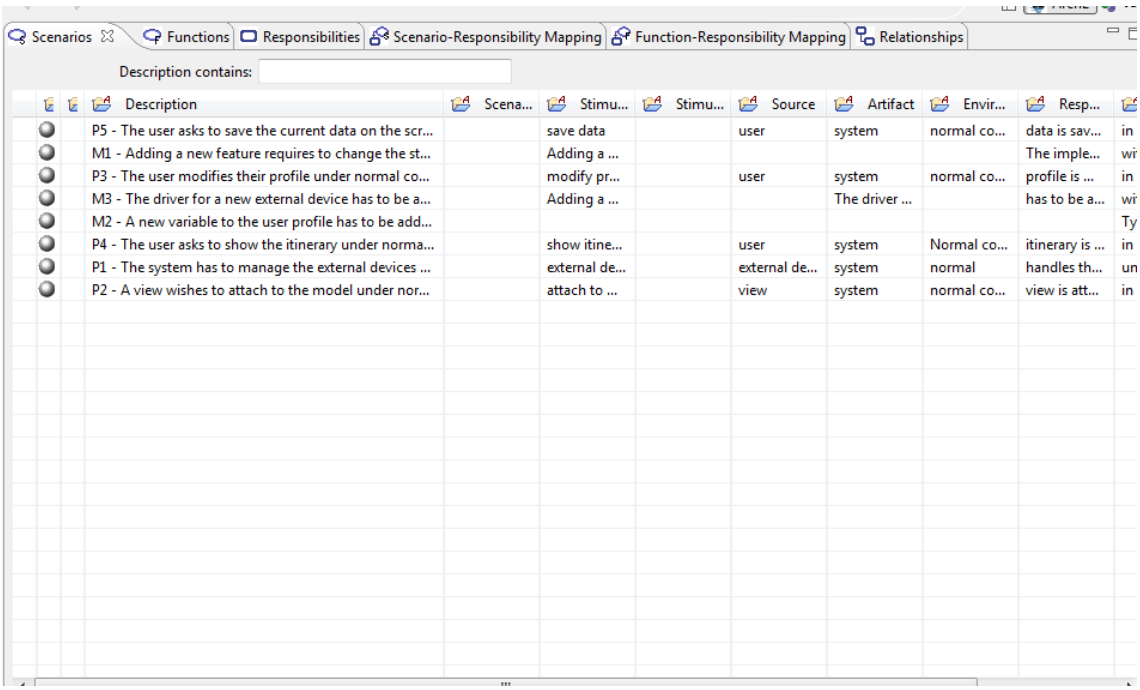


Ilustración 28: Información de escenarios

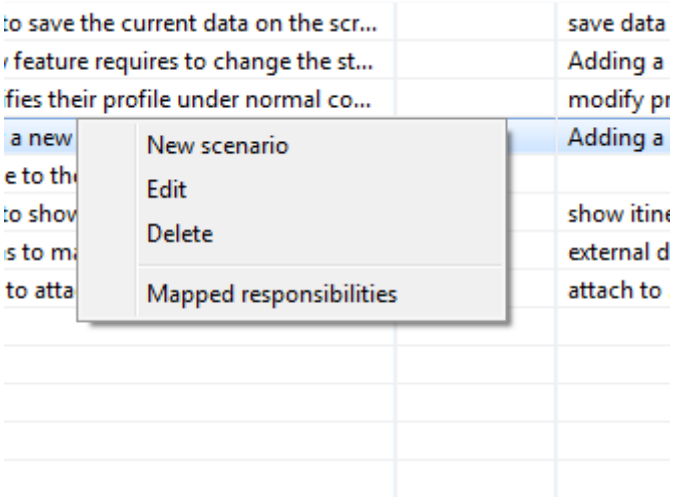


Ilustración 29: Opciones escenarios

Aparece pulsando el botón derecho del ratón encima de la tabla. Tanto en *Edit*, como en *New scenario* aparecerá una pantalla similar a la siguiente. Para todos los datos de entrada se podrá acceder de esta manera.

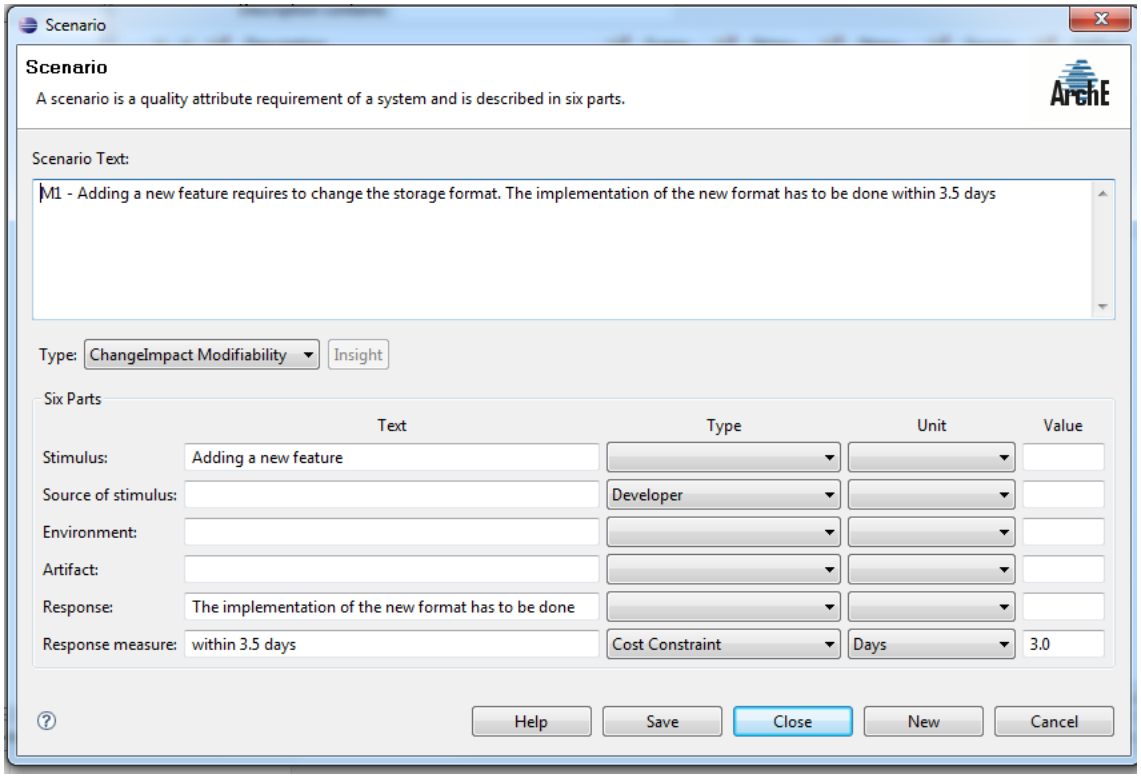


Ilustración 30: Edición escenarios

A primera vista se puede dividir Con el proyecto CTAS abierto se pueden observar todas las pantallas informativas sobre responsabilidades, relaciones entre responsabilidades, etc…

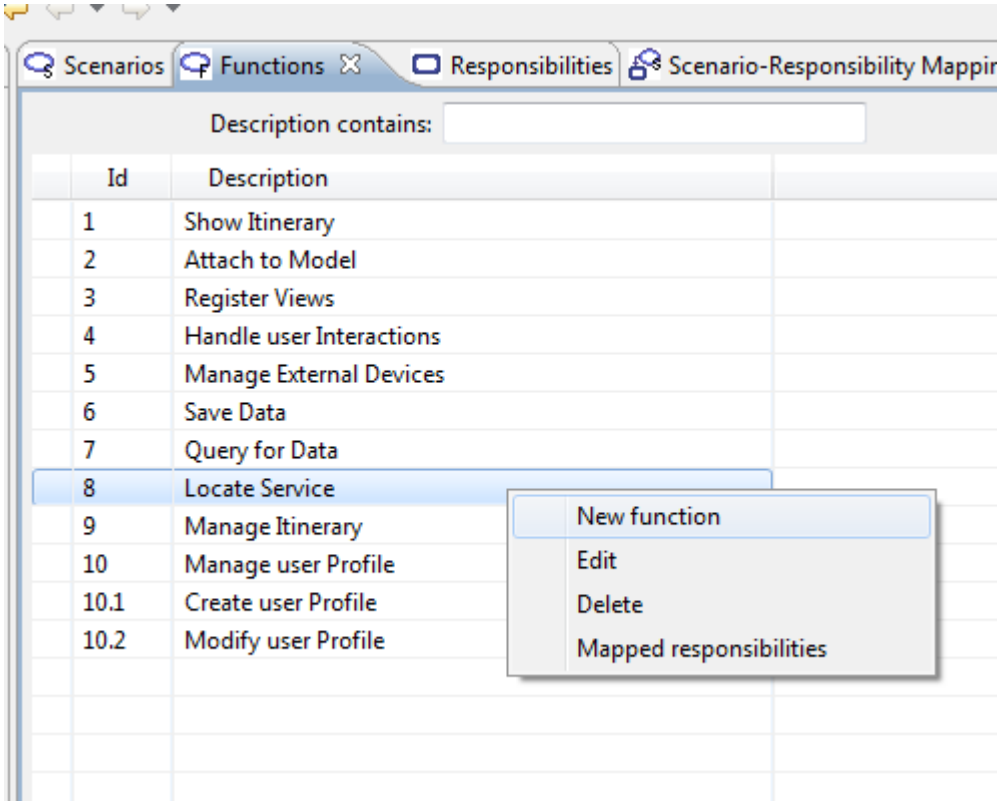


Ilustración 31: Vista Funciones

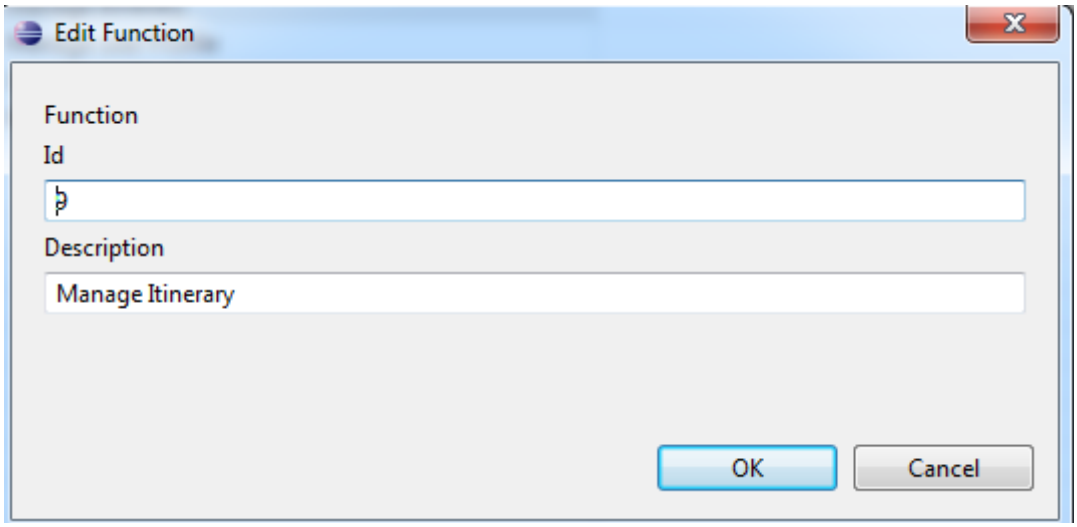


Ilustración 32: Edición Funciones

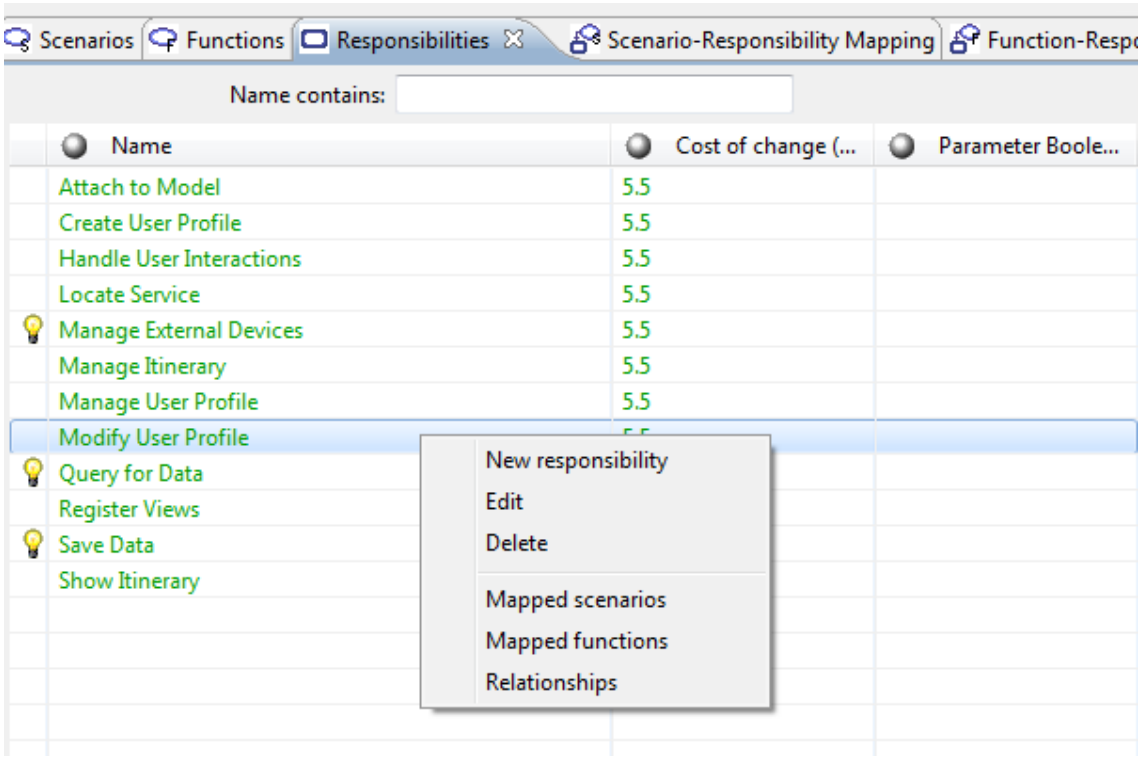


Ilustración 33: Vista Responsabilidades

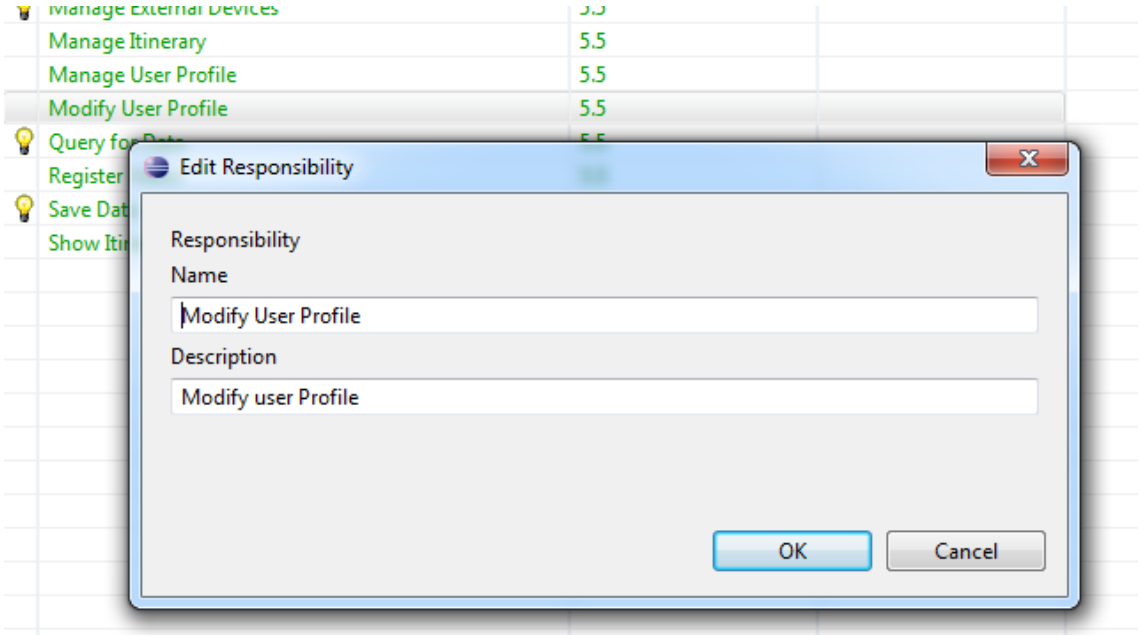


Ilustración 34: Edición Responsabilidades

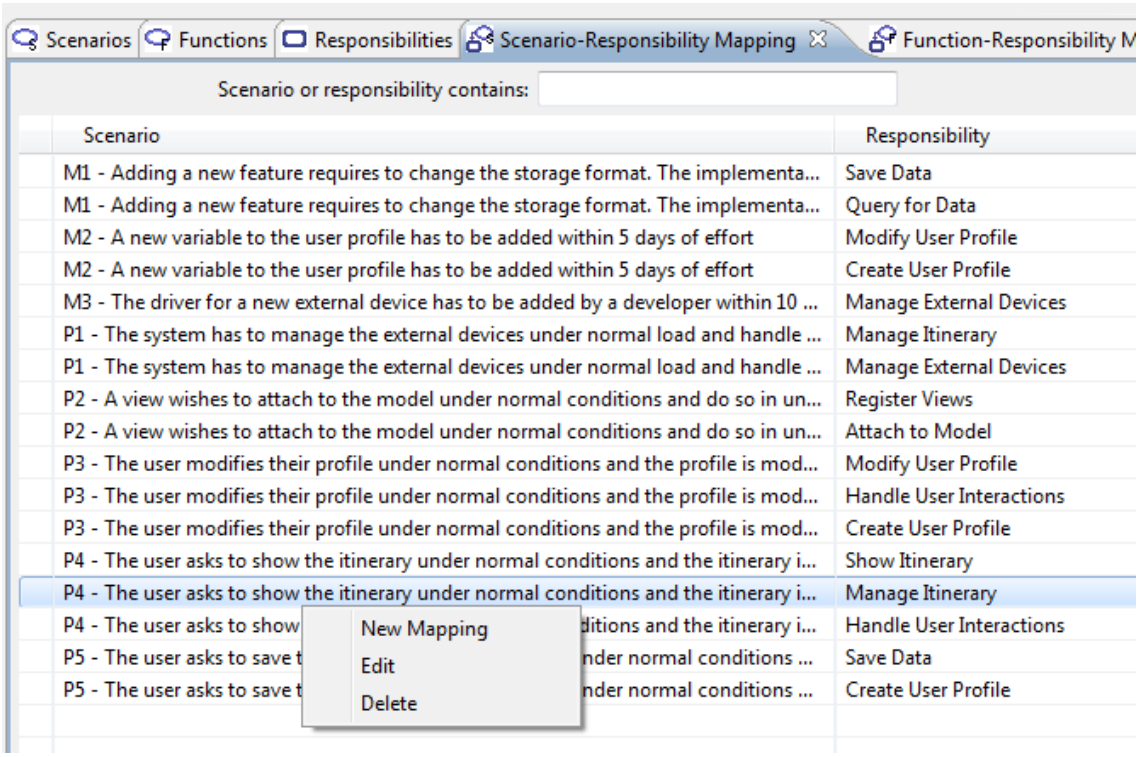


Ilustración 35: Vista Mapeo de Escenarios – Responsabilidades

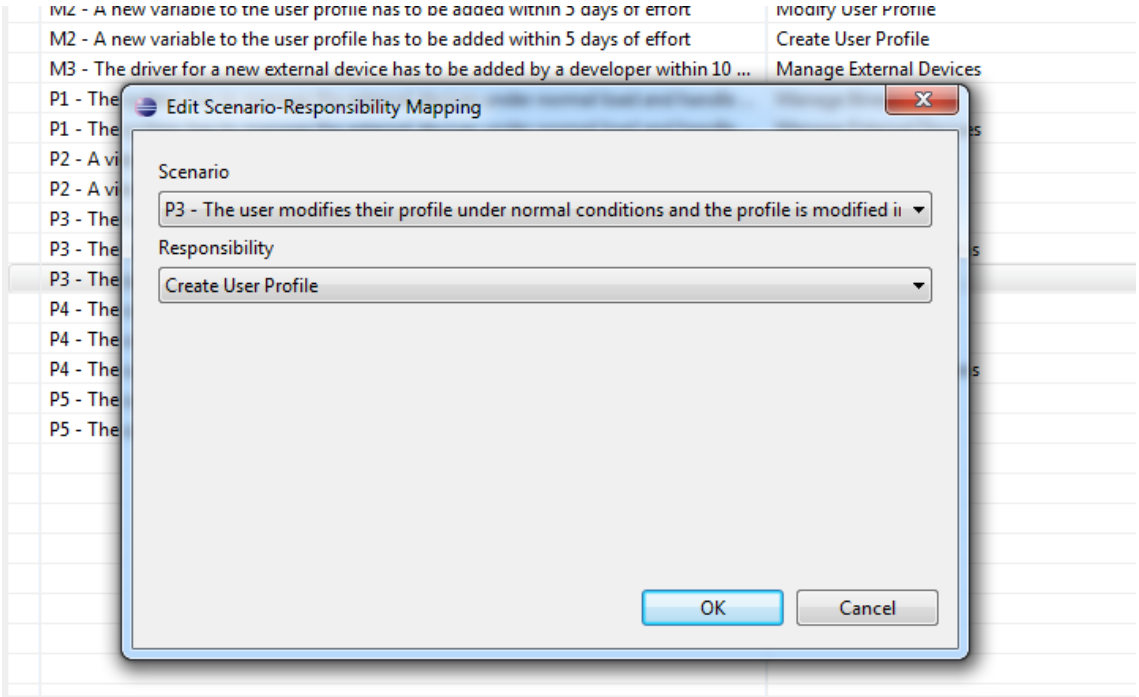


Ilustración 36: Edición Mapeo Escenarios – Responsabilidades

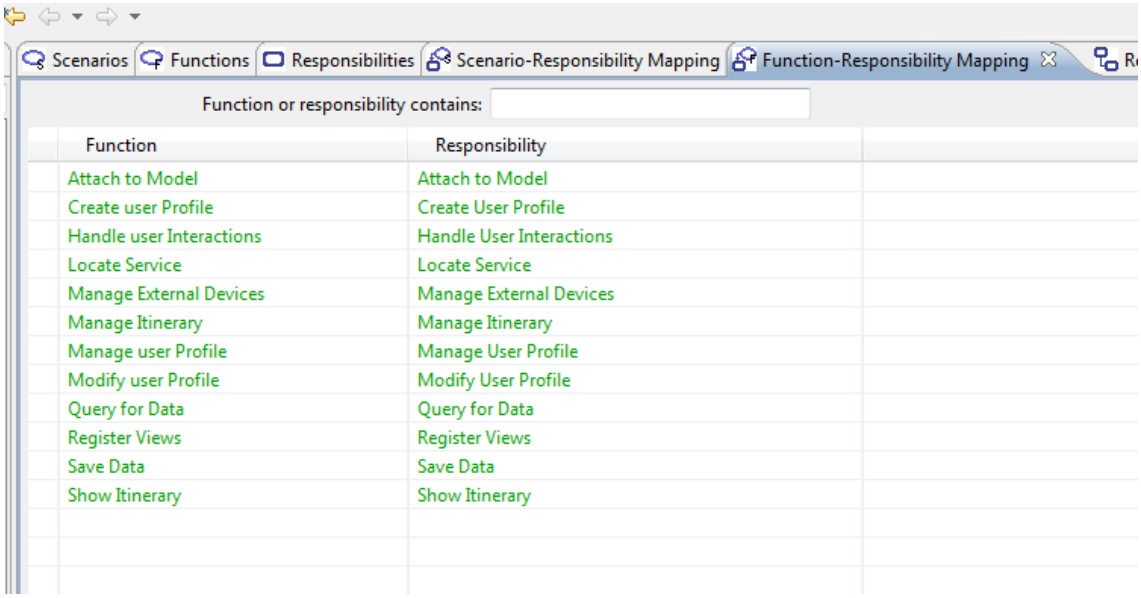


Ilustración 37: Vista Mapeo Funciones – Responsabilidades

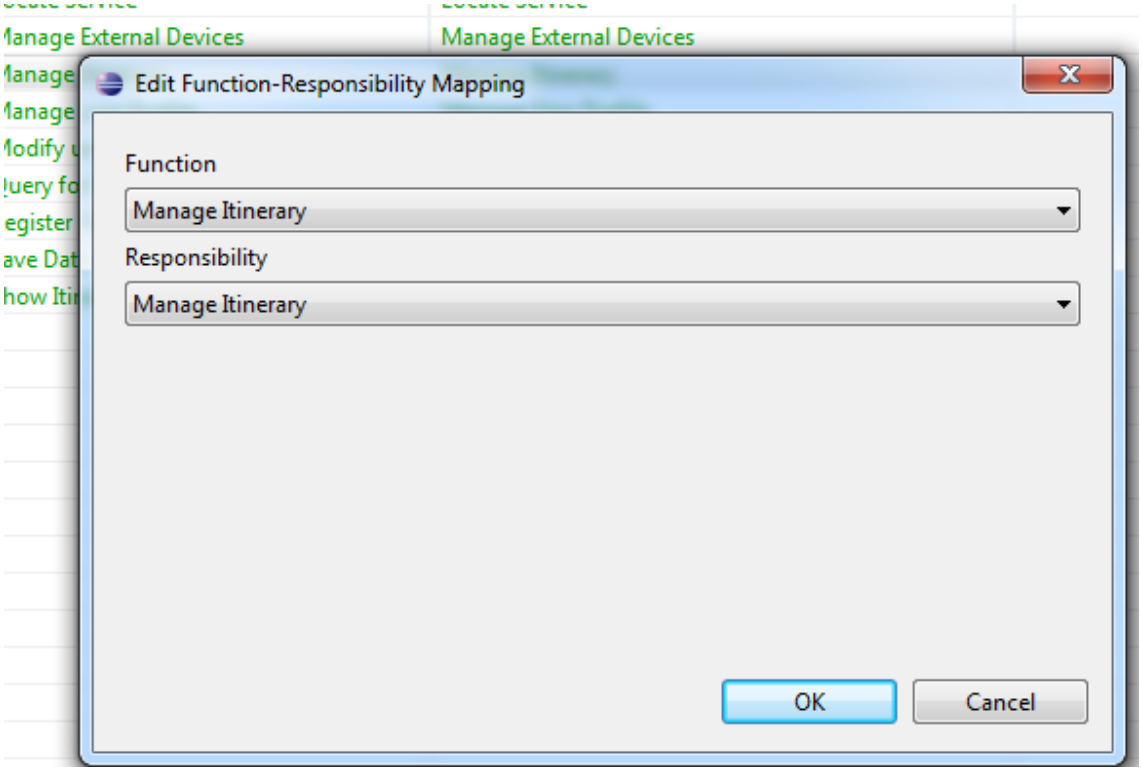


Ilustración 38: Edición Mapeo Funciones – Responsabilidades

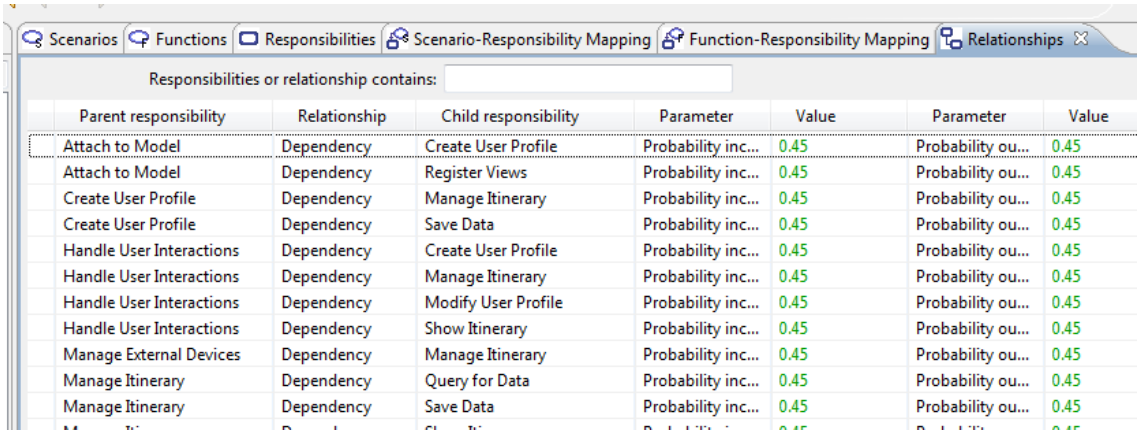


Ilustración 39: Vista Relaciones entre Responsabilidades

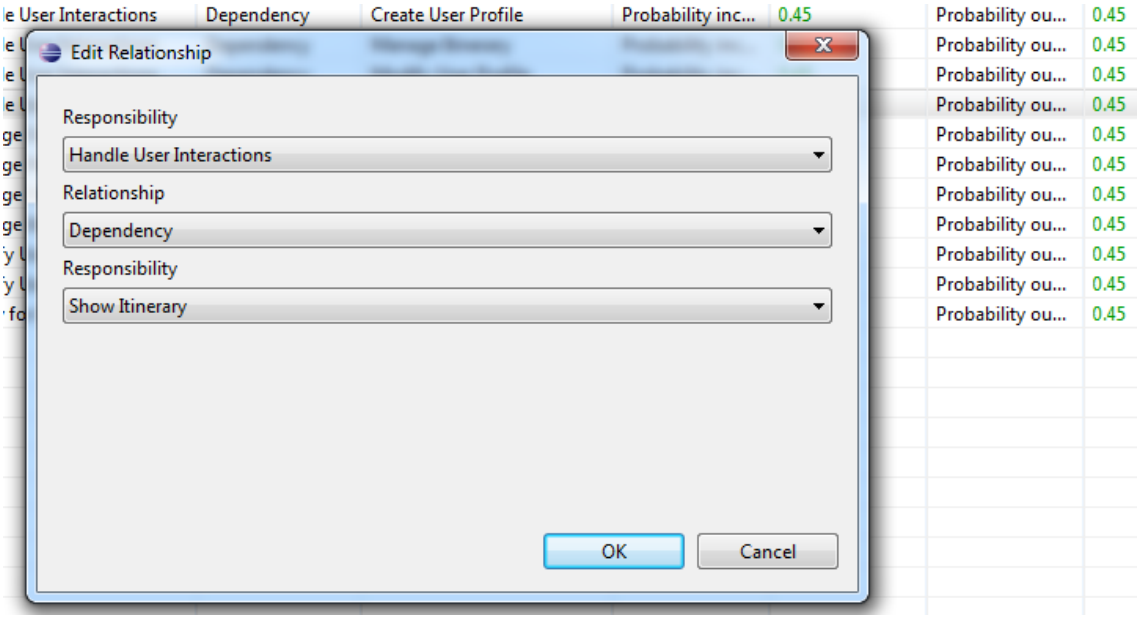


Ilustración 40: Edición Relaciones entre Responsabilidades

Hasta aquí llegarían las opciones de introducción de datos para el arquitecto, no se ha introducido una explicación de cada una por lo intuitivas que son las pantallas y porque se trata de conceptos indicados en puntos anteriores. A partir de ahora se mostrarán las pantallas informativas de la aplicación, que llevará asociada una explicación ya que algunas así la requieren. Se recuerda que, ya que se trata de una aplicación con base Eclipse, se pueden cambiar las vistas de lugar simplemente arrastrándolas al lugar deseado

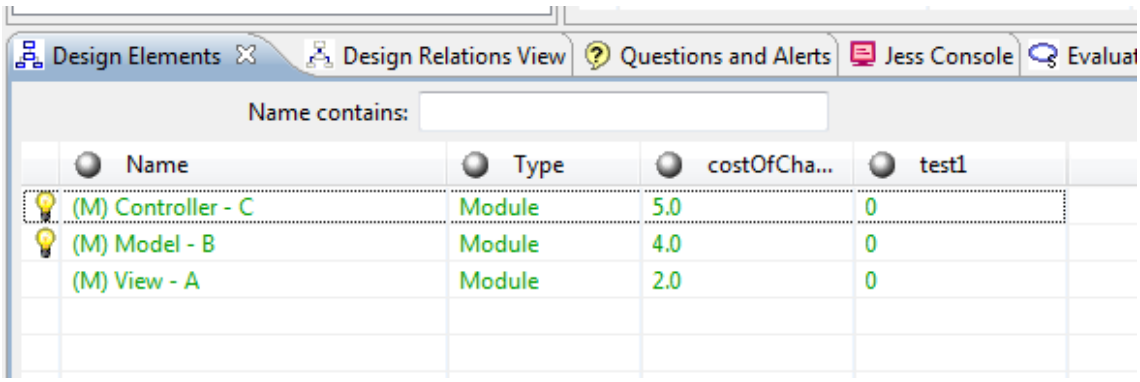


Ilustración 41: Vista Diseño de Elementos

En esta vista se muestran los elementos como modelos, es otra manera de verlos que la vista de navegación comentada anteriormente.

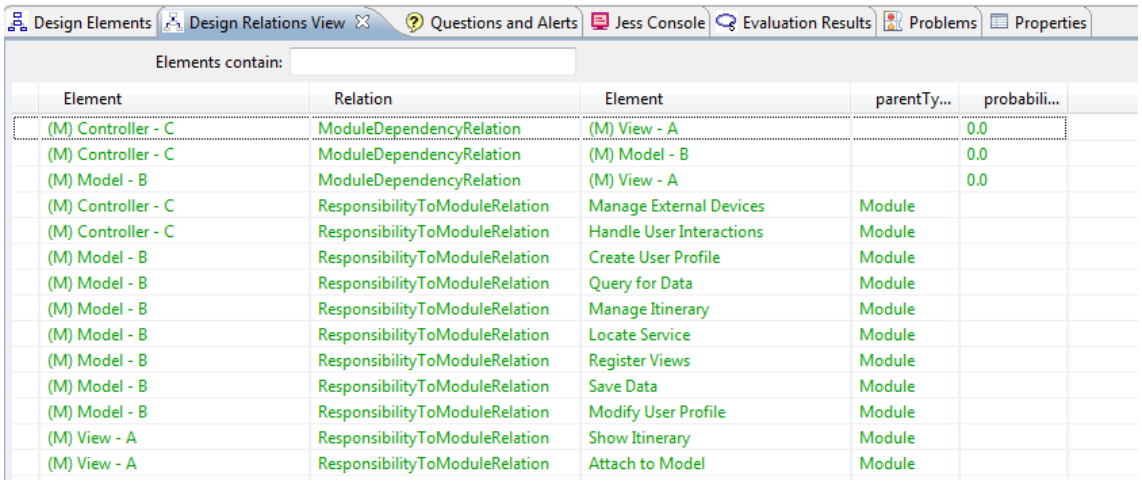


Ilustración 42: Vista Diseño Relaciones

En esta pantalla se exponen las relaciones entre módulos y elementos, como funciones o responsabilidades.

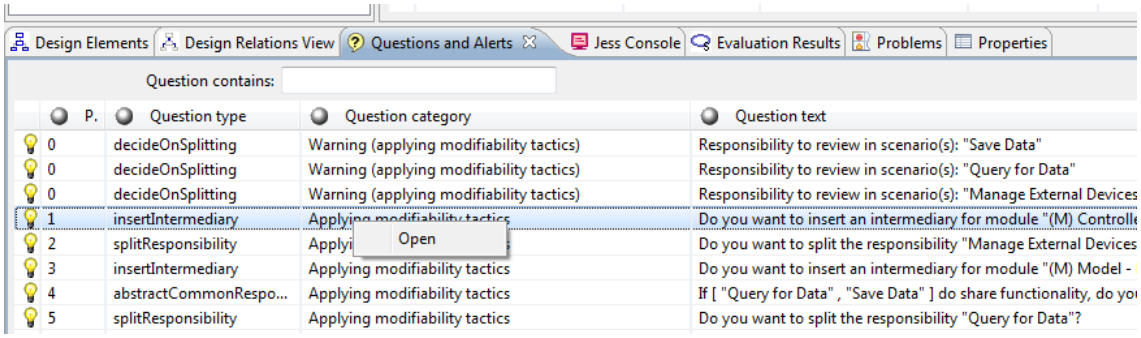


Ilustración 43: Vista Tácticas

La siguiente cuadrícula expone las tácticas propuestas después de aplicar un RF a la arquitectura en cuestión. Pulsando botón derecho encima de cada propuesta se mostrará una pantalla explicativa, donde es posible que se requiera introducir algún dato, para aplicar o no la táctica.

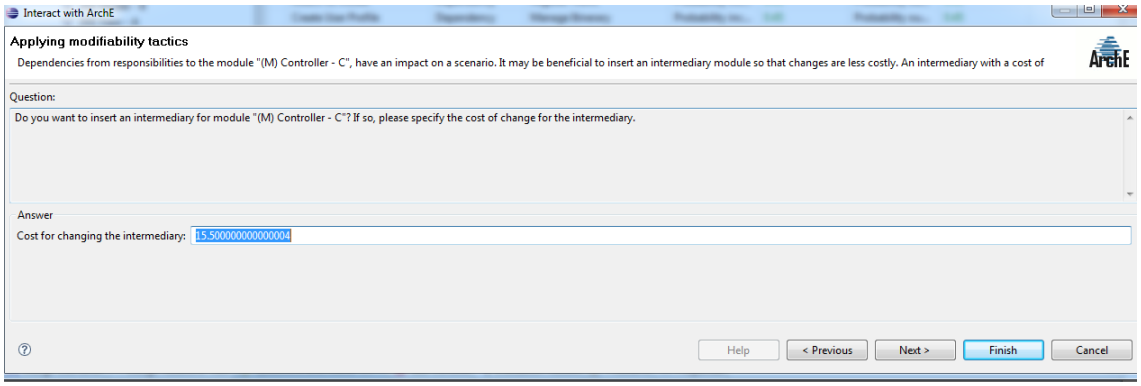


Ilustración 44: Aplicar Táctica

Como se puede apreciar aparece un texto descriptivo y, en este caso, se requiere un valor, "coste de cambiar el intermediario"; si se desea aplicarla se debe introducir un valor y pulsar "Finish", si no se debe pulsar "Cancel". Mediante los botones "Previous" y "Next" se puede avanzar por todas las tácticas propuestas.

Esta funcionalidad se ampliará en el punto siguiente con algunos ejemplos de aplicación de tácticas concretas y resultados conseguidos.

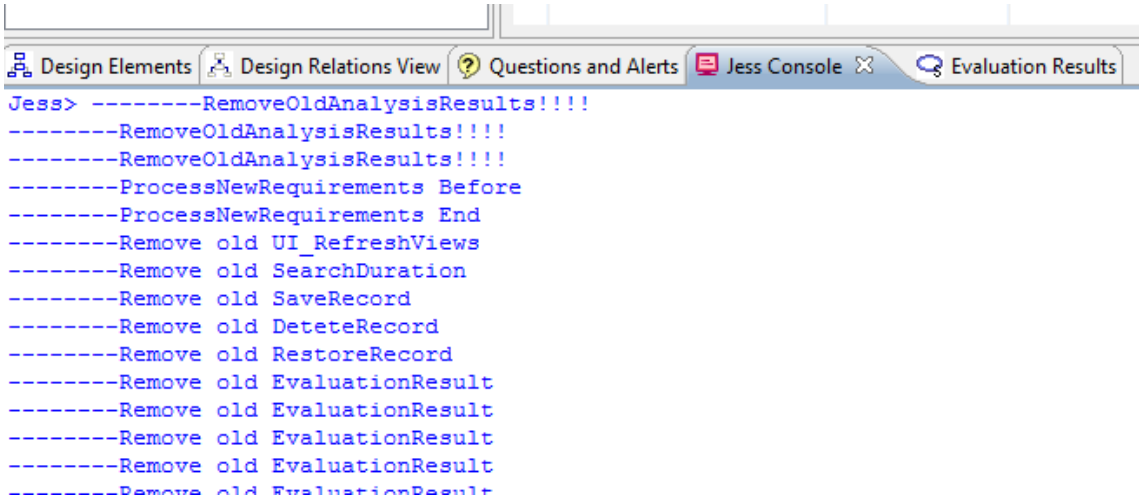


Ilustración 45: Vista Consola Jess

Se muestra el log que deja la aplicación Jess cuando se aplica alguna táctica a la arquitectura.

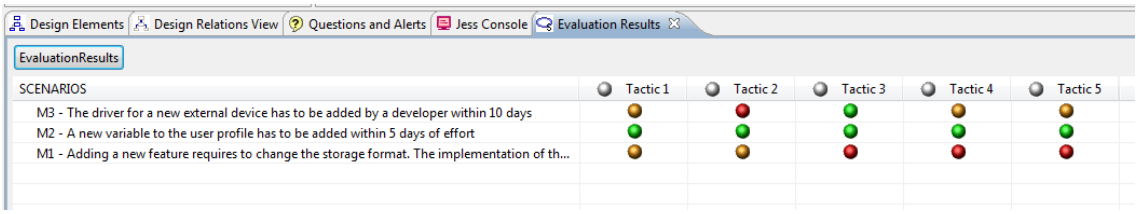


Ilustración 46: Vista Evaluación de Resultados 1

Seguramente esta sea la pantalla más interesante a nivel informativo de la herramienta, en ella se muestra el resultado esperado de aplicar las tácticas propuestas por el RF sobre cada escenario concreto de la arquitectura.

Los colores indican el grado de satisfacción de la táctica con el escenario:

- El color verde indica que la táctica satisface el escenario.
- El color naranja indica que la táctica satisface, pero no totalmente, el escenario.
- El color rojo indica que la táctica no satisface el escenario.

Si se pulsa el botón de “EvaluationResults”, la manera de exponer el resultado por la herramienta cambia de círculos a triángulos, que quieren decir lo siguiente:

- El color verde indica que la táctica va a mejorar el escenario.
- El color naranja indica que la táctica ni va a mejorar ni va a empeorar el escenario.
- El color rojo indica que la táctica va a empeorar el escenario.

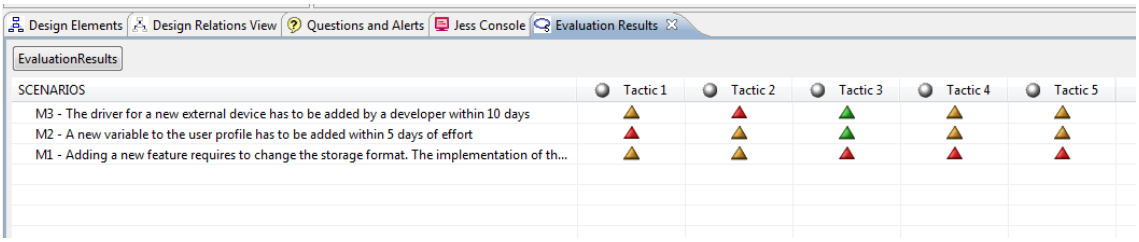


Ilustración 47: Vista Evaluación de Resultados 2

4.6. Ejemplos de utilización

En este punto se "jugará" con la herramienta para poder aplicar los conceptos resaltados durante el documento y los conocimientos adquiridos de los mismos. Para ello dentro del proyecto CTAS se han elegido los escenarios sobre los que aplique el RF de impacto de modificación o *modificabilidad* para centrarnos en un solo grupo.

Los escenarios que afectarán a las pruebas son los siguientes:

M1 – Añadir un nuevo requisito que cambie el formato de algún dato del sistema tardará como mínimo 3,5 en añadir / cambiar dicho formato.

M2 – Una nueva característica del perfil de usuario tardará en ser añadida 5 días.

M3 – Un nuevo sistema de origen de datos tarda en ser añadido al sistema por un desarrollador 10 días.

También reseñar que el proyecto consta de tres módulos llamados **A**, **B** y **C** que representan el esquema de Modelo (B), Vista (A) y Controlador (C) comentado en puntos anteriores.

Para ejecutar el RF de modificabilidad primer se debe buscar su vista asociada en la aplicación, estará típicamente en la parte posterior de la aplicación a la izquierda, se abrirá una ventana emergente donde se puede insertar un nombre para aplicar una búsqueda sobre las vistas de las que dispone ArchE.

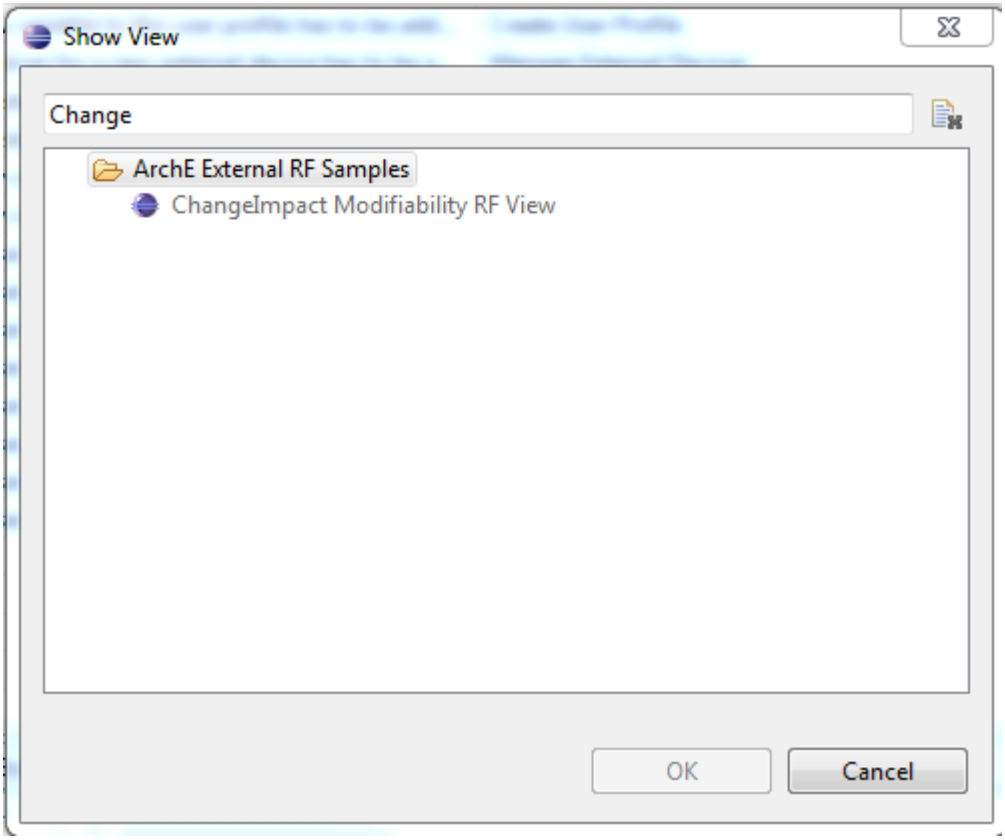


Ilustración 48: Buscar ChangeImpact Modifiability RF View

Una vez encontrado se selecciona y aparecerá en la parte posterior de ArchE junto a las perspectivas de información.

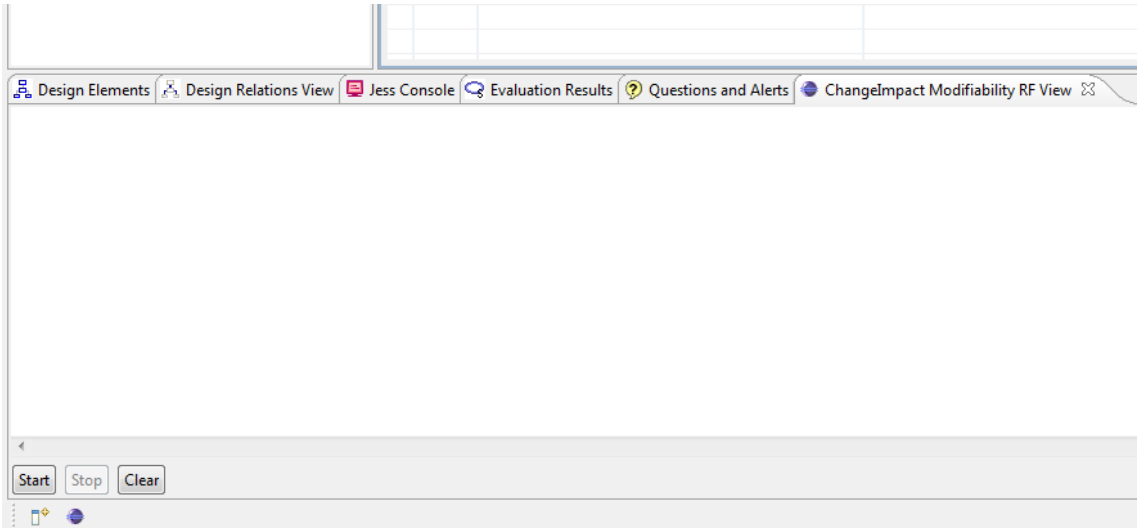


Ilustración 49: Vista RF Modificabilidad 1

La vista no es más que un log que va soltando trazas de las comprobaciones que hace el RF, se puede observar que tiene tres botones:

- *Start*: comienza con el análisis de los datos que ha introducido el arquitecto (escenarios, relaciones, responsabilidades, …)
- *Stop*: pararía con el análisis. Si este ha terminado permitiría realizar un nuevo análisis de nuevo al volverse a activar el botón *Start.*
- *Clear:* limpia la pantalla de trazas.

Si se pulsar *Start* empezará el análisis y además de llenarse el texto la vista del RF también las vistas de *Jess Console*, *Evaluation Results* y *Questions and Alerts*, vistas en el punto anterior del documento, estarán con información valiosa para el arquitecto.

Se activa el RF pulsando *Start* y ArchE sacará por la vista del RF trazas de su actividad, las perspectivas informativas tomarán datos y se podrá visualizar la información a las conclusiones y mejoras propuestas que ha llegado el RF.

En las siguientes imágenes se ven las tácticas propuestas y como afectan a los tres escenarios que se toman en cuenta:

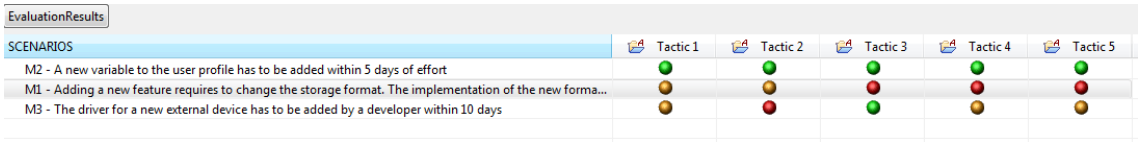


Ilustración 50: Ejemplo 1 Evaluación Resultados Círculos

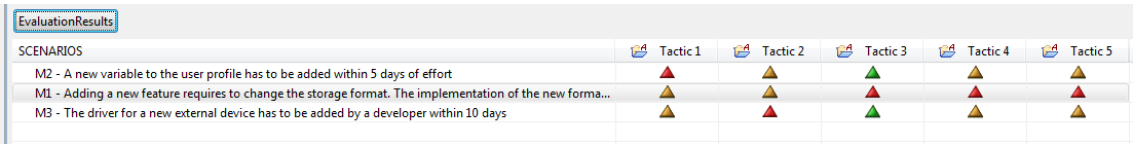


Ilustración 51: Ejemplo 1 Evaluación Resultados Triángulos

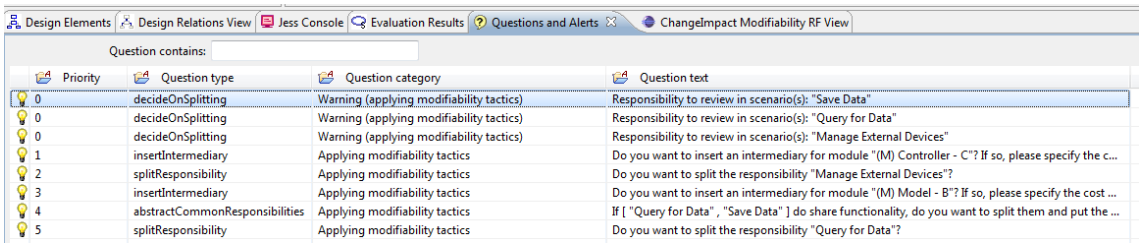


Ilustración 52: Ejemplo 1 Tácticas

La vista anterior es curiosa ya que, además de mostrarnos el detalle de las 5 tácticas propuestas y poder aplicarlas, en la primera columna viene un número indicando de que táctica se trata y las filas catalogadas con un "0" en dicha columna nos indica que hay una acción que se hará si se desencadena tal táctica.

Se elige aplicar la táctica 3 ya que va a mejorar el M2 y el M3, aunque empeora el M1. Como puede observarse las demás tácticas empeoran al menos un escenario y dejan a los demás indiferentes. Concretamente la táctica a aplicar se trata de:

Las dependencias de las responsabilidades en el Módulo B tienen un gran impacto sobre el escenario. La inserción de un módulo intermedio sería beneficiosa para bajar el coste de cambio de dicho Módulo B de 4,6 días-persona a 3,57 días-persona. Este nuevo módulo tendrá un coste de cambio de 3,9 días-persona.

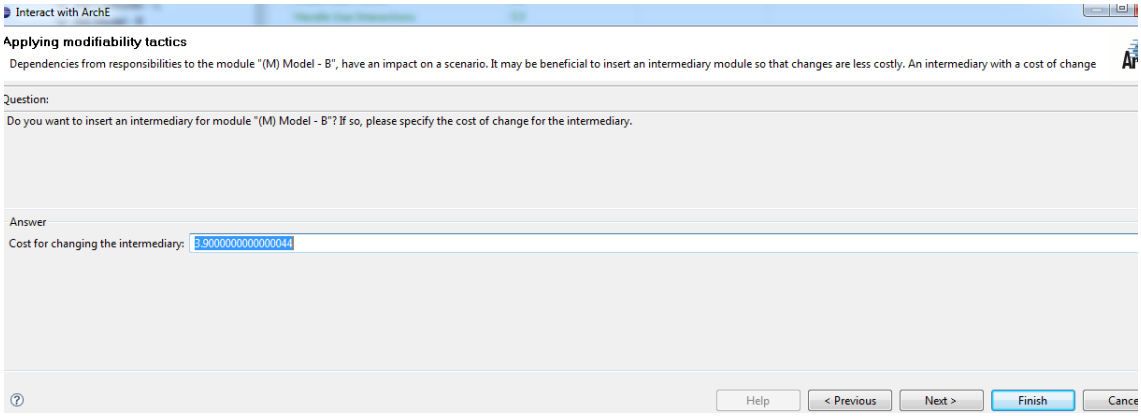


Ilustración 53: Ejemplo 1 Táctica 3

La aplicación ofrece cambiar ese *3,9* que se comenta anteriormente, dependiendo del cambio variará lo indicado anteriormente. Se elige le valor propuesto sobre todo por nuestra inexperiencia y por ser nobel con la aplicación.

Cuando aplicamos la táctica se vuelve a lanzar el RF, al terminar se pueden observar los nuevos resultados:

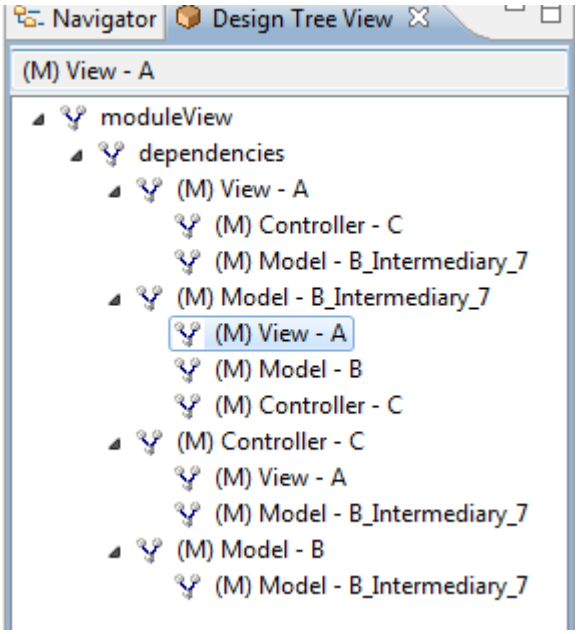


Ilustración 54: Ejemplo 1 Árbol de Módulos

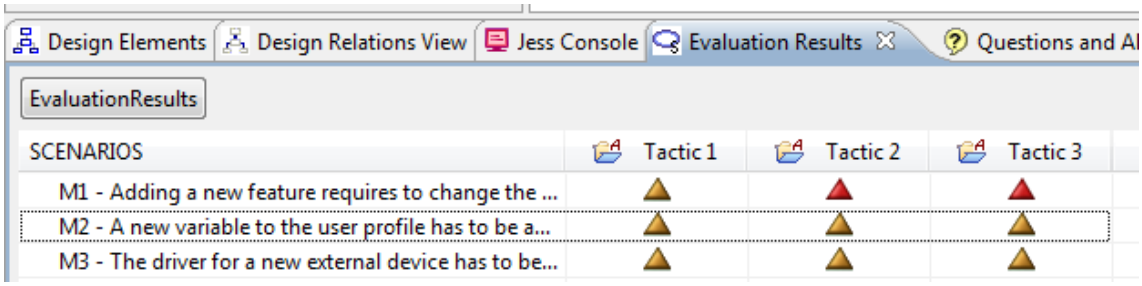


Ilustración 55: Ejemplo 1 Evaluación de Resultados Triángulos Táctica 3

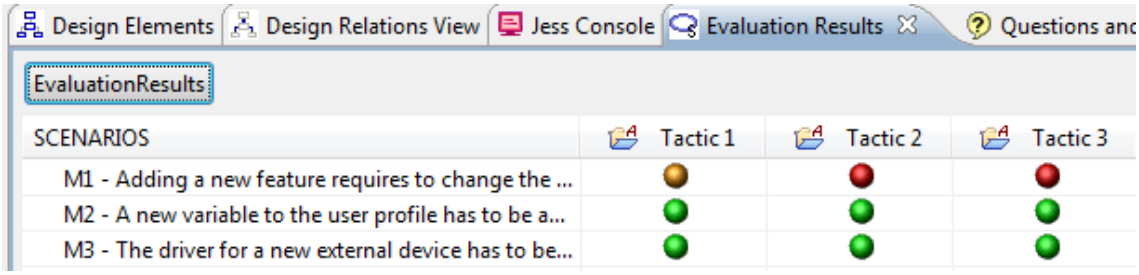


Ilustración 56: Ejemplo 1 Evaluación de Resultados Círculos Táctica 3

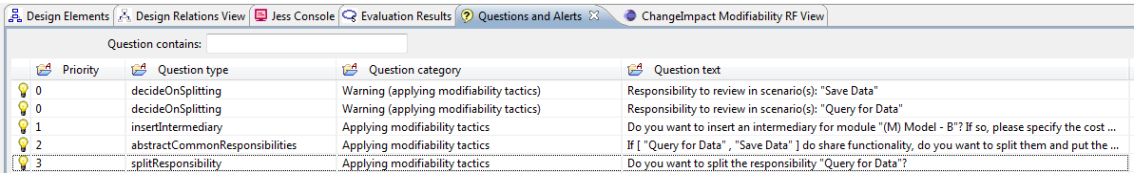


Ilustración 57: Ejemplo 1 Tácticas Táctica 3

Como puede observarse se ha creado un nuevo módulo, llamado *Model – B_Intermediary_7*, y se han propuesto tres nuevas tácticas. Para continuar con la simulación se elige la táctica 1 por ser la que mejor afecta, está táctica trataría de montar otra vez un módulo intermedio, y sus resultados serían aparentemente similares a los de la táctica anterior, nos dejaría estos resultados:

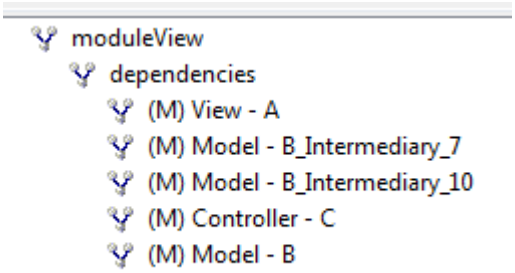


Ilustración 58: Ejemplo 1 Árbol de Módulos Táctica 1

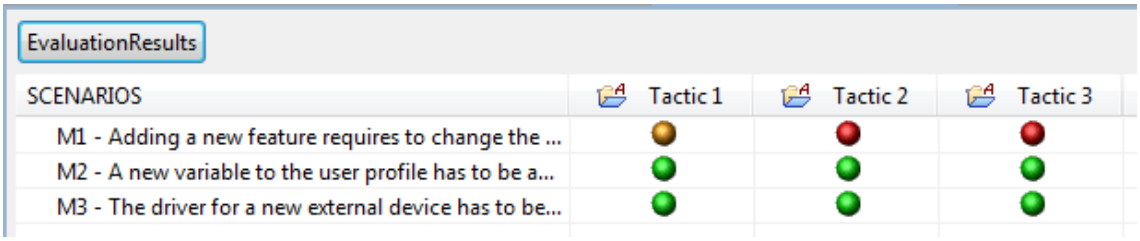


Ilustración 59: Ejemplo 1 Evaluación Resultados Táctica 1

Se puede concluir después de aplicar dicha táctica que los resultados son similares a los de la táctica 3, por ello repasamos de nuevo las tácticas que se nos han propuesto:

- Táctica 1: Incide en la creación de módulos intermedios.
- Táctica 2: Propone agrupar los aspectos comunes de las responsabilidades *Save Data* y *Quey for Data* en un módulo por separado y reducir el coste por cambios.

- Táctica 3: Formula la idea de separar la responsabilidad *Query for Data* en dos responsabilidades separando la funcionalidad para así minimizar el coste de un posible cambio.

Para este ejemplo se elegirá aplicar la táctica 2 ya que es la que mejor explica la herramienta, estos serían los resultados:

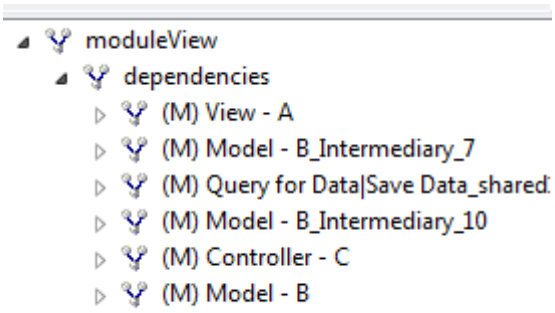


Ilustración 60: Ejemplo Árbol de Módulos Táctica 2

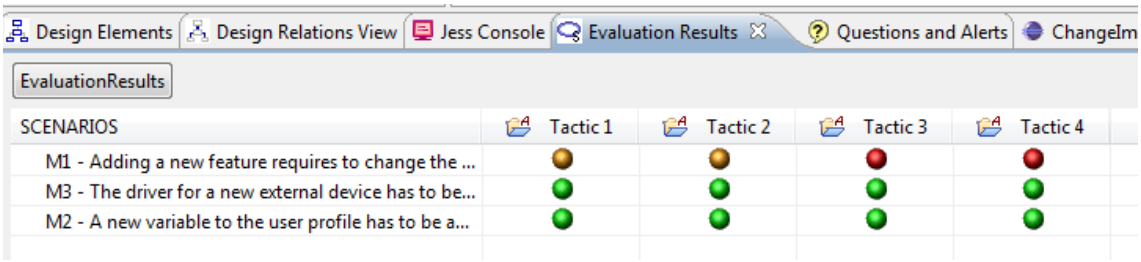


Ilustración 61: Ejemplo 1 Evaluación Resultados Círculos Táctica 2

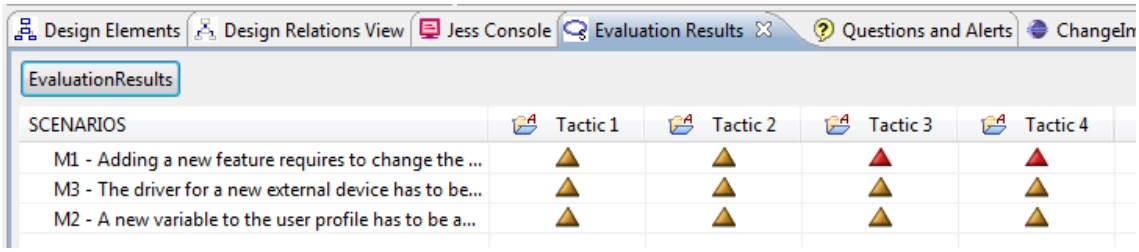


Ilustración 62: Ejemplo 1 Evaluación Resultados Triángulos Táctica 2

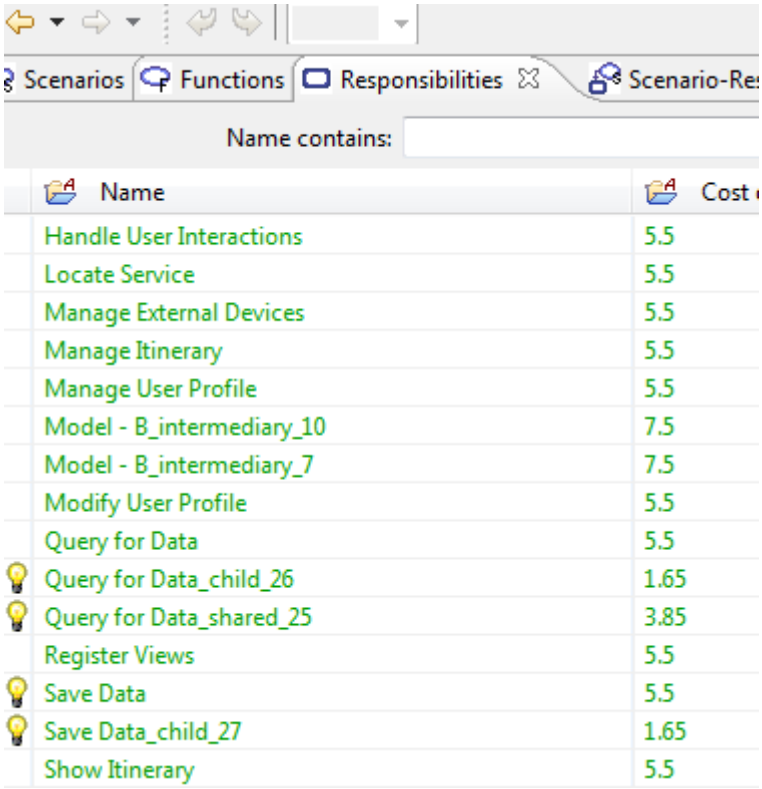


Ilustración 63: Ejemplo 1 Responsabilidades Táctica 1

Como puede observarse se han creado 3 nuevas responsabilidades, una agrupar los aspectos comunes de los dos responsabilidades agrupadas y otras 2 para los aspectos dispares, acompañado a esto se ha creado un nuevo módulo para dichos aspectos comunes. Se recuerda que se pueden cambiar los nombres de las responsabilidades pero no de los módulos, lo que permitiría un mayor entendimiento. Se proponen ahora 4 tácticas:

- Táctica 1: Ofrece asociar los escenarios a las nuevas responsabilidades que tienen los aspectos dispares.

- Táctica 2: Propone dividir la nueva responsabilidad con los aspectos comunes en dos para mejorar el coste de modificación.

- Táctica 3: Es una copia de la táctica aplicada justo antes, táctica 2 anterior, pero entre las responsabilidades nuevas *Query for Data_shared_25* y *Query for Data_child_26.*

- Táctica 4: Brinda la posibilidad de crear un módulo intermediario para el nuevo módulo que se ha creado por agrupar atributos comunes.

Con esto se da por finalizado el primer ejemplo con el que se pretendía explicar más pantallas y funcionalidades de la herramienta y además percibir el tipo de tácticas que se ofrece, como se ha visto hay dos que parece que van a aparecer en multitud de ocasiones, una es la creación de un módulo intermedio y la otra es la descomposición de responsabilidades en varias.

A continuación se especifica el Ejemplo 2, para ello se partirá de los mismos escenarios que en el ejemplo anterior:

M1 – Añadir un nuevo requisito que cambie el formato de algún dato del sistema tardará como mínimo 3,5 en añadir / cambiar dicho formato.

M2 – Una nueva característica del perfil de usuario tardará en ser añadida 5 días.

M3 – Un nuevo sistema de origen de datos tarda en ser añadido al sistema por un desarrollador 10 días.

Y de las mismas tácticas:

1. Inserción de un intermediario para el módulo C.

2. División la responsabilidad *Manage External Devices*.

3. Inserción de un intermediario para el módulo B.

4. División de las responsabilidades *Query for Data* y *Save Data* y juntar las características comunes en una responsabilidad independiente.

5. División de la responsabilidad *Query for Data*.

En vez de aplicar la táctica que impacte de mejor manera en la arquitectura se va a cambiar esa estrategia por la elección de la táctica según el “conocimiento” del arquitecto (en este caso yo mismo) sobre el sistema y las tácticas propuestas, se seguirá este tipo sea cual sea el resultado ya que el objetivo sigue siendo la exploración sobre el uso de la herramienta. Se elige la táctica **cuatro** porque se entiende que agrupando las características de las responsabilidades mencionadas se va a conseguir una mejora en la modificabilidad, este sería el resultado:

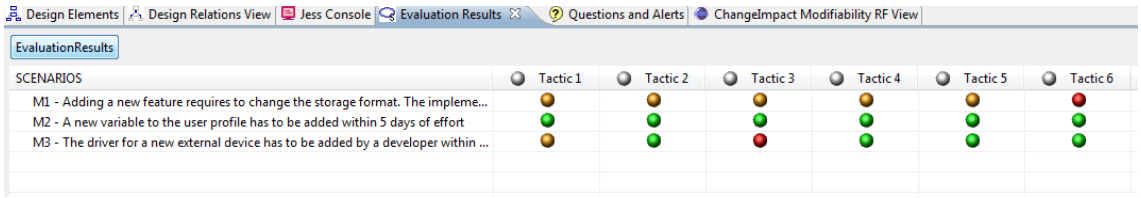


Ilustración 64: Ejemplo 2 Evaluación Resultados Círculos Táctica 4

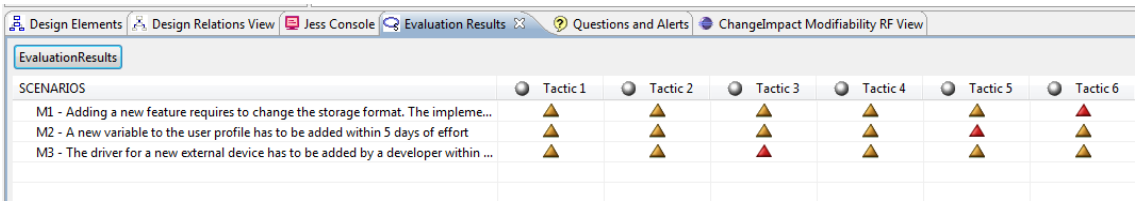


Ilustración 65: Ejemplo 2 Evaluación Resultados Triángulos Táctica 4

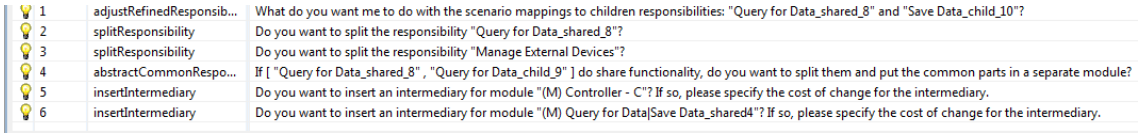


Ilustración 66: Ejemplo 2 Nuevas Tácticas Paso 1

Como se puede apreciar ArchE ofrece 6 tácticas, las cuales no tienen ningún efecto positivo (verde) sobre la arquitectura. De todas maneras se sigue escogiendo la táctica en base al conocimiento del arquitecto, en este caso se elige la táctica número **tres**: división de la responsabilidad *Manage External Devices*. Se divide la responsabilidad para poder así tener responsabilidades por tipo de dispositivo externo, este es el resultado de la aplicación de la arquitectura:

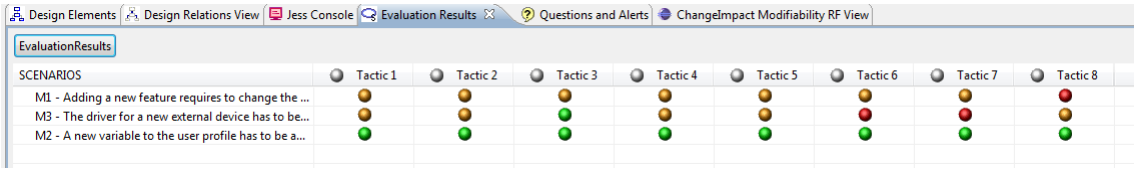


Ilustración 67: Ejemplo 2 Evaluación Resultados Círculos Táctica 3

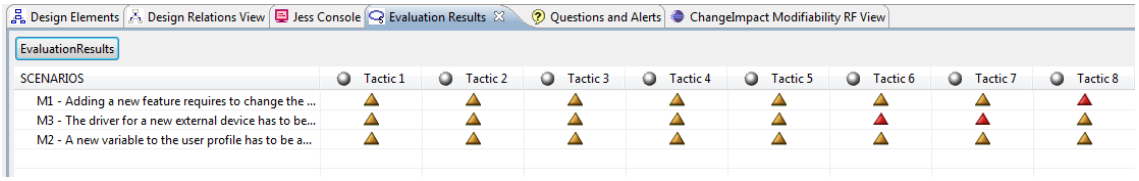


Ilustración 68: Ejemplo 2 Evaluación Resultados Triángulos Táctica 3

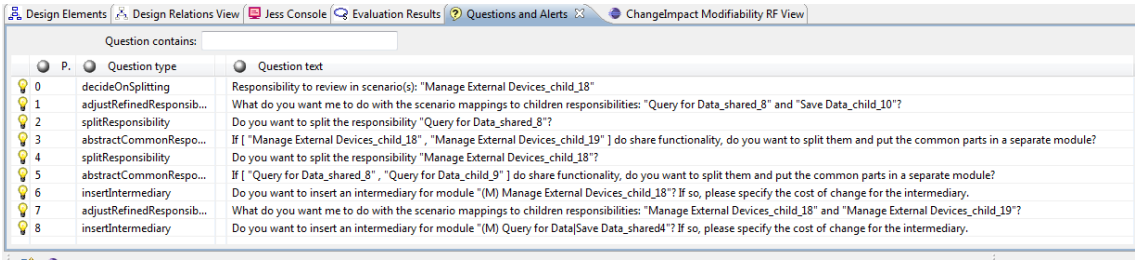


Ilustración 69: Ejemplo 2 Nuevas Tácticas Paso 2

Después de la aplicación de la táctica surgen nada más y nada menos que ocho nuevas tácticas, se valora que todas son positivas para el escenario número dos. Se elige de nuevo la táctica número **tres**: división de las responsabilidades hijas *Manage External Devices* y la fusión en una nueva responsabilidad de las características comunes, este sería el resultado:

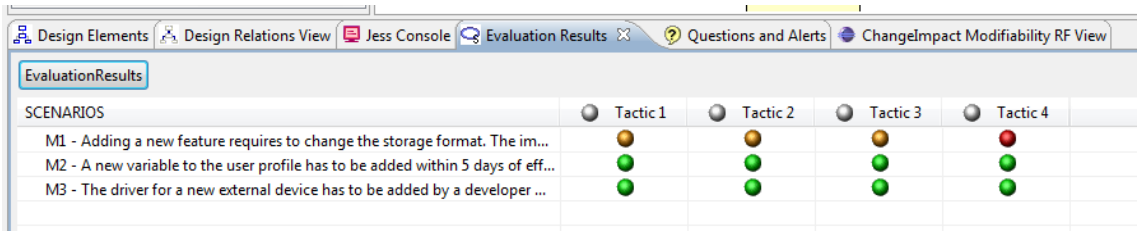


Ilustración 70: Ejemplo 2 Evaluación Resultados Círculos Táctica 3 bis

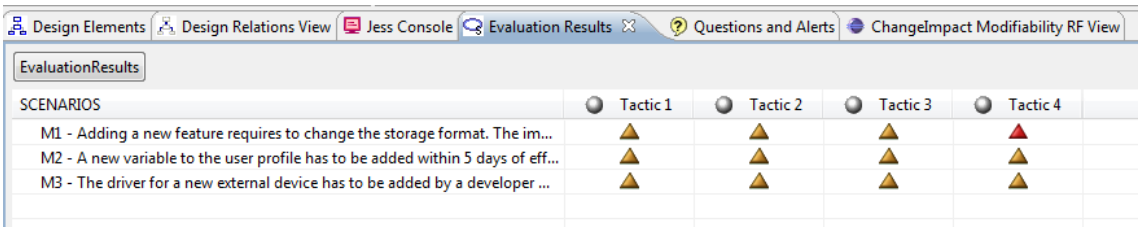
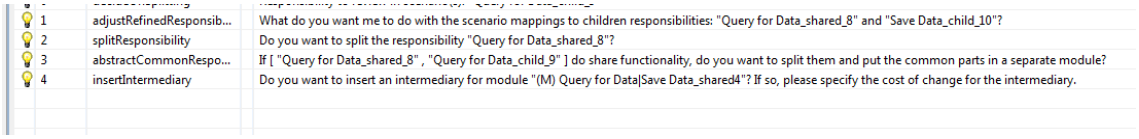


Ilustración 71: Ejemplo 2 Evaluación Resultados Triángulos Táctica 3 bis



1	adjustRefinedResponsib...	What do you want me to do with the scenario mappings to children responsibilities: "Query for Data_shared_8" and "Save Data_child_10"?
2	splitResponsibility	Do you want to split the responsibility "Query for Data_shared_8"?
3	abstractCommonRespo...	If ["Query for Data_shared_8" , "Query for Data_child_9"] do share functionality, do you want to split them and put the common parts in a separate module?
4	insertIntermediary	Do you want to insert an intermediary for module "(M) Query for Data|Save Data_shared4"? If so, please specify the cost of change for the intermediary.

Ilustración 72: Ejemplo 2 Nuevas Tácticas Paso 3

Como se puede ver el resultado resulta positivo, ya que se reducen las tácticas propuestas y, a la vez, estas son más positivas para los escenarios.

En este punto se resuelve dejar de aplicar tácticas porque se deja explicado el funcionamiento de la herramienta y como puede beneficiar al trabajo de un arquitecto; las conclusiones sobre su funcionamiento se expondrán en el siguiente punto.

4.7. Mejora sobre CTAS

Es este punto se resuelve dejar de aplicar tácticas con el fin de de explicar y profundizar sobre el manejo de la herramienta ArchE, la finalidad será introducir una mejora sobre el proyecto CTAS, a través de ArchE obviamente, en base a los conocimientos adquiridos por la investigación y realización del proyecto. La mejora tratará sobre el atributo de calidad de la *modificabilidad* y se realizará en base a la aplicación de tácticas.

Este sería el estado del proyecto al empezar en cuanto a escenarios:

- M1 – Añadir un nuevo requisito que cambie el formato de algún dato del sistema tardará como mínimo 3,5 en añadir / cambiar dicho formato.
- M2 – Una nueva característica del perfil de usuario tardará en ser añadida 5 días.
- M3 – Un nuevo sistema de origen de datos tarda en ser añadido al sistema por un desarrollador 10 días.

En base a estos tres escenarios vamos a intentar mejorarlos utilizando tres pilares: la experiencia propia en desarrollo software, conocimiento adquirido en el estudio de arquitecturas y herramienta ArchE y la investigación independiente. Por supuesto se utilizará la herramienta ArchE para realizar dicha mejora.

Se realiza una primera comprobación del proyecto con el RF de modificabilidad, los resultados serán los anteriormente dispuestos en puntos anteriores:

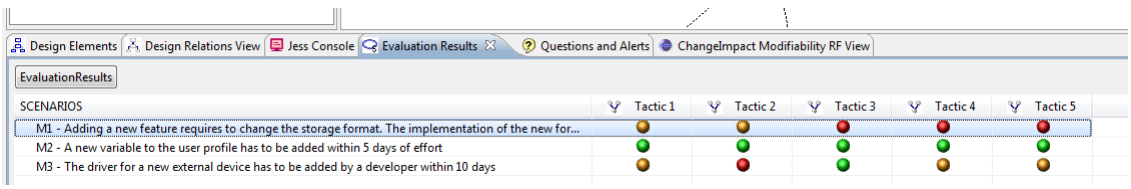


Ilustración 73: Evaluación Resultados Círculos Mejora Paso 1

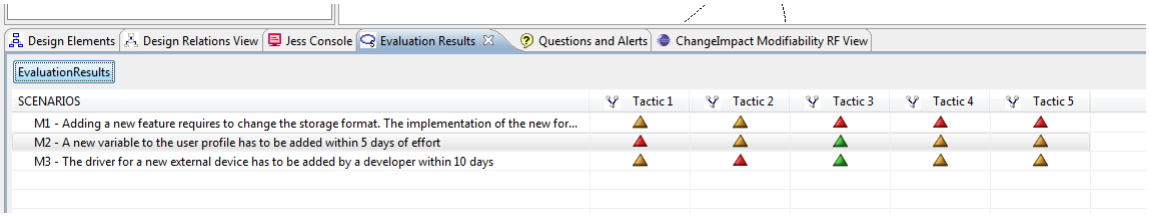


Ilustración 74: Evaluación Resultados Triángulos Mejora Paso 1

Claramente la herramienta no aconseja utilizar la táctica **número 3** que reza:

Insertar un intermediario para el Módulo B (Model) con un costo de 3,9 días.

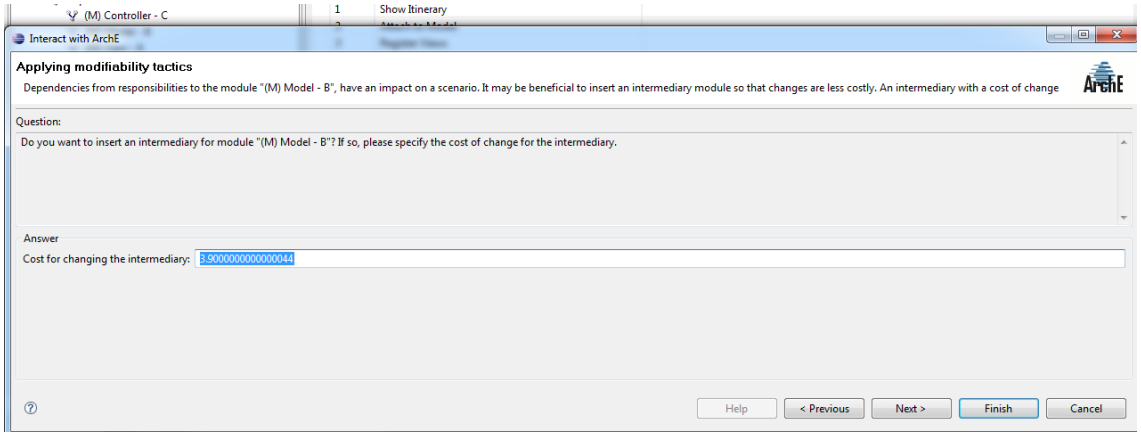


Ilustración 75: Mejora Táctica 3 Paso 1

La aplicamos y nos encontramos el siguiente resultado:

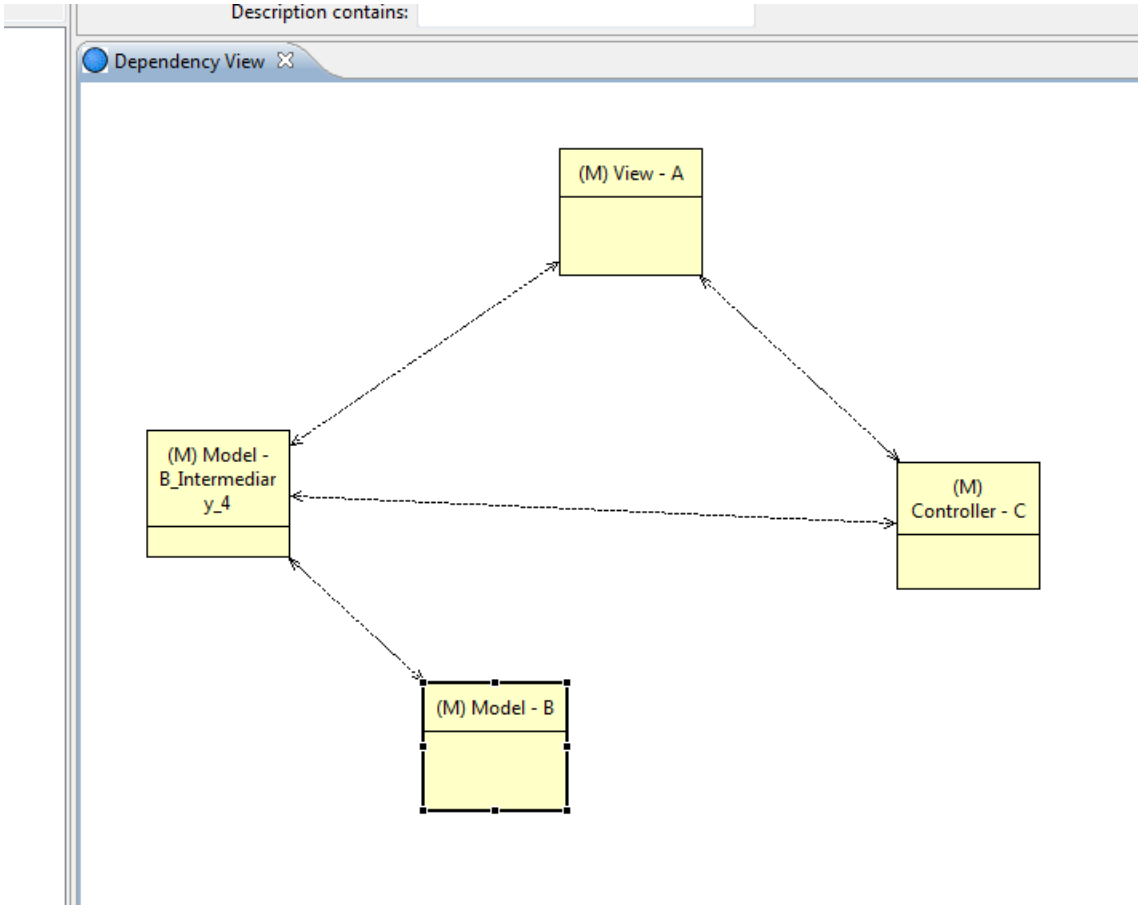


Ilustración 76: Mejora Diagrama Módulos Tras Paso 1

Como puede observarse en este sencillo diagrama el intermediario se inserta para servir de interfaz del Módulo B y los otros dos.

Como se observa en el siguiente cuadro se han conseguido mejorar los escenarios dos y tres aunque a la vez se ha empeorado el escenario 1.

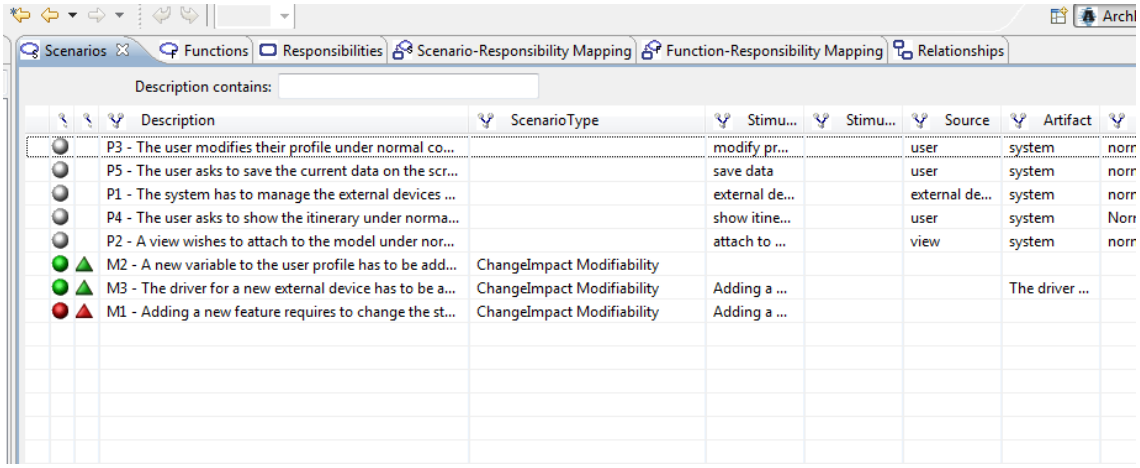


Ilustración 77: Mejora Escenarios Paso 1

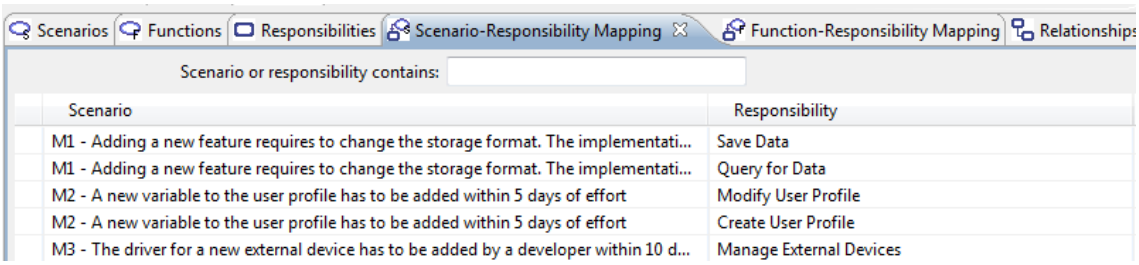


Ilustración 78: Relación Escenarios - Responsabilidades

Realizando una interpretación de cómo han quedado los escenarios, con el apoyo de la relación de escenarios/responsabilidades en la **Ilustración 78**, se puede concluir que:

- M1: al introducir un nuevo módulo lógicamente aumentaría el tiempo de introducir un nuevo requisito por tanto este escenario empeora respecto al estado anterior del sistema.

- M2: ya que este escenario está totalmente relacionada con la creación/modificación de los perfiles de usuario la táctica iba, en mayor medida, orientada a mejorar dicho escenario. Con la colocación del intermediario se consigue desacoplar el módulo B por lo que el cambio iría solamente orientado a dicho módulo y al intermediario, 2 módulos y no tres como antes.

- M3 – similar al anterior, el módulo B es el orientado al tratamiento de datos en primera instancia (*Model*), por lo tanto un nuevo origen de datos sólo afectaría a dicho módulo y, dependiendo de la casuística, al módulo intermediario.

Y a continuación se exponen las tácticas propuestas por la herramienta:

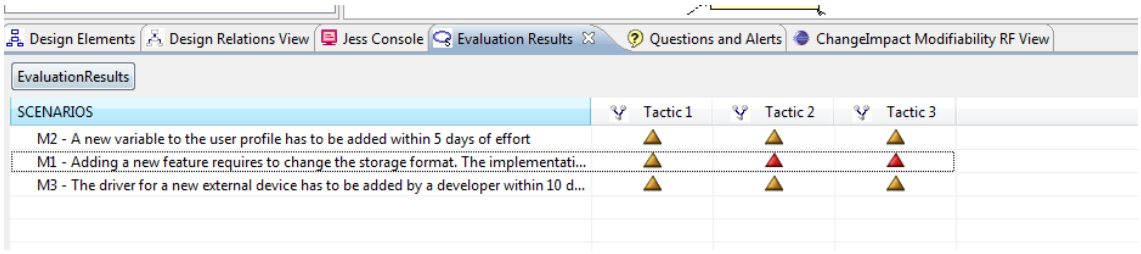


Ilustración 79: Evaluación Resultados Triángulos Mejora Paso 2

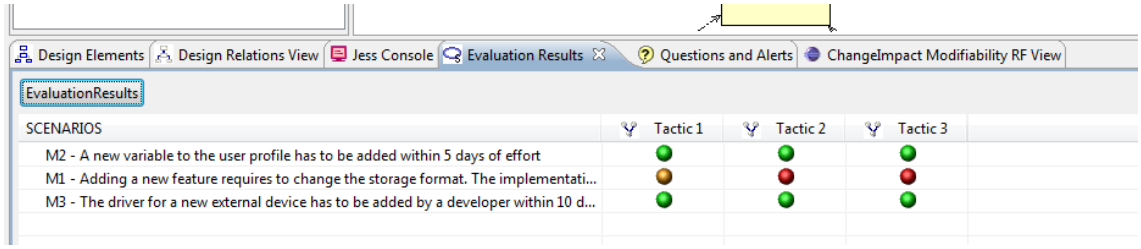


Ilustración 80: Evaluación Resultados Círculos Mejora Paso 2

- Táctica 1: *Insertar un intermediario para el Módulo B (Model) con un costo de 3,9 días*.
- Táctica 2: *Crear un nuevo módulo que incluya las funcionalidades comunes de las responsabilidades "Guardar Datos" y "Consultar Datos"*.
- Táctica 3: *Dividir la responsabilidad "Consultar Datos"*.

En teoría, según ArchE, la mejor elección sería aplicar la táctica número uno pero se desprecia de la elección ya que, por el conocimiento de la herramienta y con un pensamiento a futuro, se vería "falto de coherencia" introducir un nuevo módulo intermediario en el mismo lugar dónde se ha introducido en el paso anterior, además la herramienta da la sensación que se enclava en un *bucle* y propone continuamente dicha táctica.

Por lo tanto quedarían dos tácticas a elegir, a primera vista se parecen bastante: tienen el punto común de la responsabilidad "*Consultar Datos*", si se analiza un poco la información que nos da la herramienta se puede concluir:

- Sobre Táctica 2: se propone separar en un nuevo módulo las características comunes de *"Guardar Datos"* y *"Consultar Datos"*, si se piensa fríamente es muy normal que al guardar datos en una BBDD o fichero se realicen consultas para realizar comprobaciones de integridad, relaciones entre entidades, etc.., por lo que aplicar esta táctica parece un movimiento lógico.

- Sobre Táctica 3: se propone separar las funcionalidades de la responsabilidad *"Consultar Datos"* en dos, la aplicación no nos da más información al respecto.

Sobre estas pautas lo más lógico es la elección de la Táctica 2 para su aplicación sobre el proyecto CTAS. El resultado sería el siguiente:

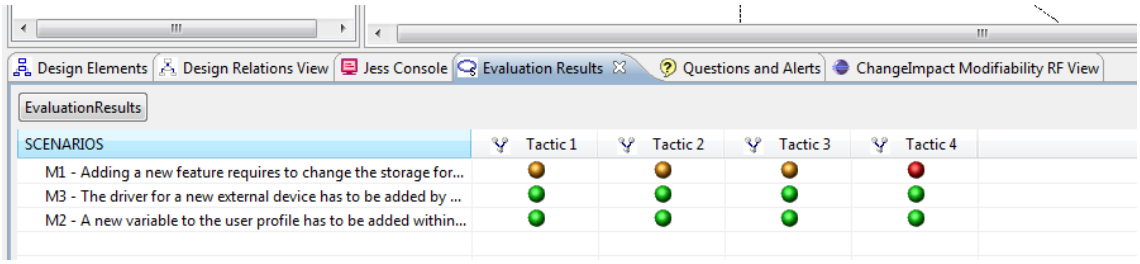


Ilustración 81: Evaluación Resultados Círculos Mejora Paso 3

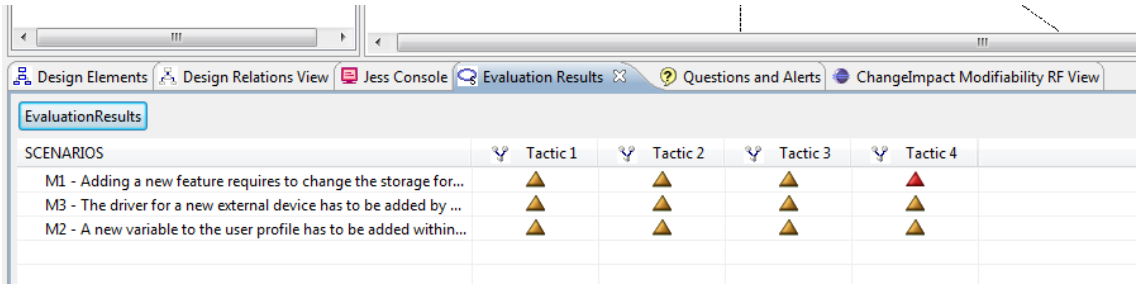


Ilustración 82: Evaluación Resultados Triángulos Mejora Paso 3

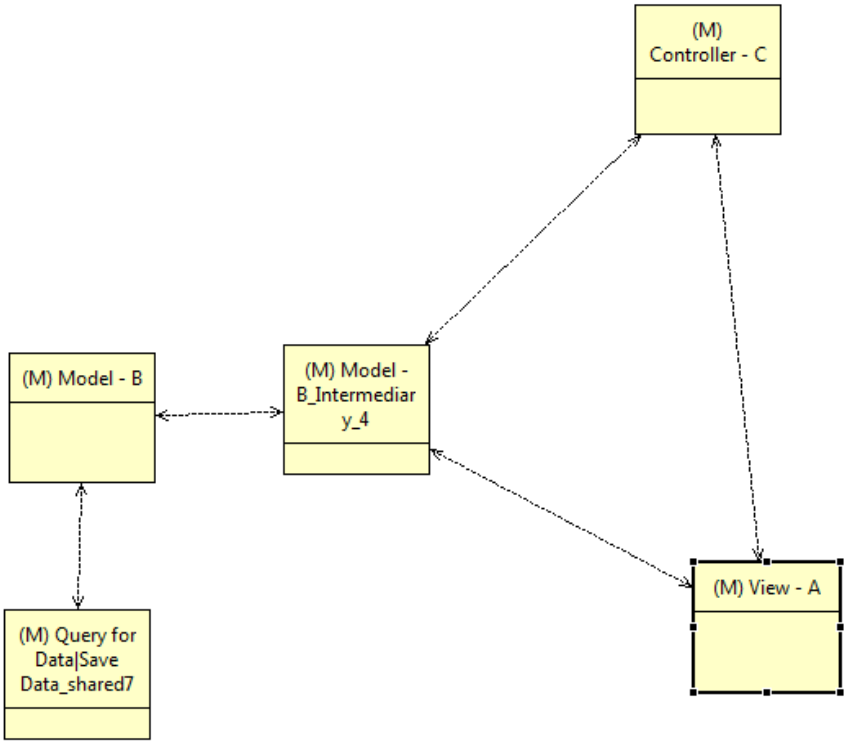


Ilustración 83: Diagrama Módulos Mejora Paso 3

		Description	Scena...	Stimu...	Stimu...	Source
		P2 - A view wishes to attach to the model under nor...		attach to ...		view
		P1 - The system has to manage the external devices ...		external de...		external de...
		P3 - The user modifies their profile under normal co...		modify pr...		user
		P5 - The user asks to save the current data on the scr...		save data		user
		P4 - The user asks to show the itinerary under norma...		show itine...		user
		M2 - A new variable to the user profile has to be add...	ChangeIm...			
		M3 - The driver for a new external device has to be a...	ChangeIm...	Adding a ...		
		M1 - Adding a new feature requires to change the st...	ChangeIm...	Adding a ...		

Ilustración 84: Escenarios Mejora Paso 3

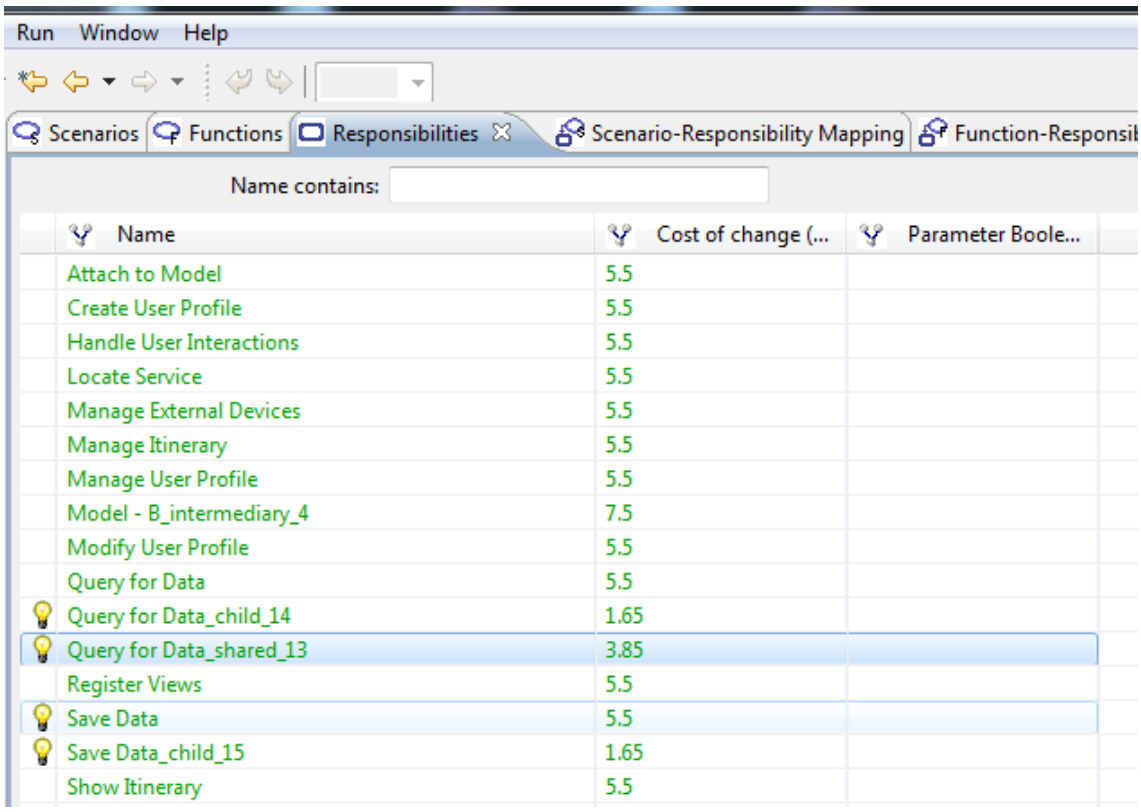


Ilustración 85: Responsabilidades Mejora Paso 3

Como puede observarse se añaden nuevas responsabilidades y un nuevo módulo. Se sigue mejorando el comportamiento de los escenarios M2 y M3 pero así no se consigue optimizar el comportamiento del escenario M1.

En cuanto a las tácticas que la herramienta propone son las siguientes:

- Táctica 1: *Modificar las nuevas responsabilidades (en cuanto a asignación al escenario 2)*

- Táctica 2: *Dividir la responsabilidad hija de la antigua "Consultar Datos"*, ahora llamada "*Query for Data_shared_13"*.

- Táctica 3: *Crear un nuevo módulo que incluya las funcionalidades comunes de las responsabilidades "Query for Data_Shared_13" y "Query for Data_chid_14"*.

- Táctica 4: *Insertar un intermediario para el Módulo B (Model) con un costo de 3,9 días*.

Como ha podido apreciarse en las ilustraciones anteriores ninguna de estas tácticas contribuye a mejorar el escenario M1 (*Añadir un nuevo requisito que cambie el formato de algún dato del sistema tardará como mínimo 3,5 en añadir / cambiar dicho formato*), lo cual parece lógico si se lee detenidamente cada táctica propuesta, todas introducen nuevos componentes o entidades con lo que aumenta la lista de piezas a tocar si se quisiera introducir un requisito nuevo.

Por tanto se va a introducir una modificación dentro de la arquitectura de forma "manual" en base a los conocimientos adquiridos y la experiencia del autor.

Como puede observarse, después de las tácticas aplicadas, las responsabilidades asociadas al escenario M1 han aumentado (resaltadas en verde están las nuevas):

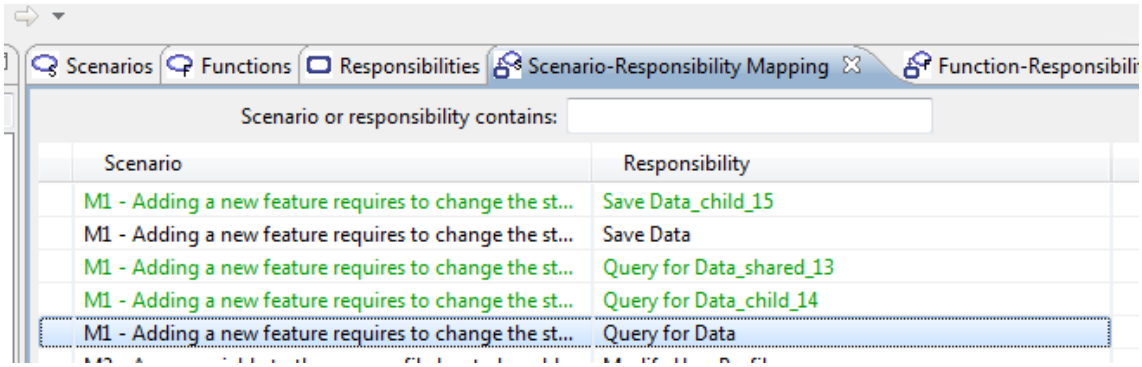


Ilustración 86: Responsabilidades Escenario M1

Para este escenario, cuantas más responsabilidades existan será peor para él, ya que más lugares habrá que estudiar y modificar para introducir un nuevo cambio.

Para acortar el tiempo de implementación se propone seguir estas tácticas:

1. **Archivos de configuración para establecer los parámetros de arranque**
2. **Utilizar polimorfismo para llamar a funciones en tiempo de ejecución**
3. **Utilización de protocolos para la unión de procesos independientes en tiempo de ejecución**

Para este escenario en concreto utilizar archivos de configuración no aportará ninguna mejora en concreto, por lo que las más claras serían la segunda y la tercera.

Si se ve de una manera abstracta la opción 2 y la 3 son similares ya que tratan de introducir una especie de mecanismo de mediación para que la conversión de datos y los procesos sea transparente en la comunicación.

Para este ejemplo parece más claro aplicar la tercera opción, ya que CTAS es una aplicación que utilizarán varios proveedores que tendrán sus propios componentes y sistemas, de esta manera se introduce un método para hacer la comunicación igual entre procesos.

Pero como se representa esta táctica en la aplicación ArchE?, la mejor manera sería quitar el enlace entre la táctica M1 y las responsabilidades hijas de *Query of Data* y *Save Data*, de esta forma se simboliza dentro de la aplicación que la comunicación entre las responsabilidades padre e hijas utilizan un protocolo que hará transparente la comunicación aunque cambien los datos.

Después de todo el proceso expuesto en este punto se realiza una mejora dentro del sistema CTAS que afecta a todos los escenarios afectados a la cualidad de *modificabilidad*.

5. Conclusiones y líneas de trabajo futuras

Si se repasan los objetivos descritos en el punto segundo de este documento se puede observar que se han cumplido todos ellos, sobre todo el principal que trataba sobre la comprensión de los requisitos de calidad o atributos de calidad, qué son, reconocerlos y medir su impacto así como poder aplicarlos al desarrollo de la arquitectura. A continuación se exponen las conclusiones a modo de *highlights* por medio de una lista:

- Se ha comprendido la importancia de los requisitos de calidad a la hora de diseñar y desarrollar una arquitectura software.

- Se ha realizado un ensayo sobre los textos propuesto, entendiendo su contenido e insertando las opiniones personales que han surgido durante la lectura y el análisis de los mismos.

- Se ha entendido el uso e instalación de la herramienta ArchE mostrando en este texto una guía tanto de instalación como de uso.

- Se han logrado aplicar los conocimientos adquiridos en los textos propuestos sobre el proyecto CTAS con la herramienta ArchE, aplicando mejoras sobre la arquitectura para la mejora de la misma.

- Se ha constatado que ArchE tiene funciones diferentes a lo que estamos acostumbrados como una aplicación estándar de Windows, como guarda los cambios, importación/exportación de proyectos…

- Se ha probado la herramienta ArchE donde se han encontrado resultados dispares, a mi entender al aplicar tácticas parece como si la herramienta se metiera en un “bucle” con algún tipo de tácticas, como la de insertar módulos intermedios.

Los conocimientos adquiridos por el autor en este proyecto, tanto por el estudio de nuevos conceptos y métodos que de aplicación real, suponen una experiencia de valor incalculable.

A continuación se enumeran una serie de posibles líneas de trabajo futuro sobre la herramienta ArchE:

- Mejora de las funciones comunes, como permitir guardar o deshacer/rehacer los cambios.

- Permitir cambiar los nombres de los elementos creados a partir de la aplicación de tácticas, los nombres que la aplicación introduce son muy comunes y esto daría un punto muy importante de libertad al arquitecto para hacer más entendible el paso a paso de evolución de la mejora de su arquitectura.

- Dar más importancia al tratamiento de módulos, es un concepto importante de la herramienta que "parece" mientras se utiliza la misma que está ahí a nivel informativo. Se podría hacer más accesible simplemente permitiendo cambiar los nombres aunque se puede ser más amplio en este aspecto.

- Introducir las funciones de importación/exportación de proyectos daría más versatilidad a la herramienta para poder trasladar tan sólo un fichero, por ejemplo del formato más estándar ahora mismo que es el xml, y no el sistema de carpetas del workspace para trabajar en el proyecto en otro equipo.

- Ahora mismo la aplicación tan sólo mide los atributos de rendimiento y modificabilidad, sería bueno que la herramienta ampliara este espectro hacia más atributos.

Como una línea futura más global sería bueno que desde el SEI se amplíe la documentación relacionada con ArchE ya que, desde mi punto de vista, es escasa y la localización dentro de la página de lugar a malentendidos; sobre todo la documentación en lo referente a como conectar un marco de razonamiento con la herramienta, ya que de esta manera sería más fácil que desarrolladores externos se "animen" a participar construyendo nuevos RF.

Como comentario que se sale de "conclusiones y líneas de trabajo futuro" se opina que ArchE es una herramienta que puede serle muy útil a un arquitecto pero que será este, en base a su experiencia y prioridades, el que le dará forma la arquitectura. Por ejemplo se ha visto que la herramienta siempre propone nuevas tácticas y será el arquitecto quien tenga que poner punto y final en algún momento.

6. Bibliografía y Referencias

Len Bass, Paul Clements, Rick Kazman (2003). *Software in Practice, Second Edition.* Addison Wesley.

David Garlan, Mary Shaw (1994). *An introduction to software architecture.* [CMU/SEI-94-TR-21, ESC-TR-94-21]. *Software Engineering Institute.*

Paul Clements, Len Bass. (2010). *Relating Business Goals to Architecturally Significant Requirements for Software Systems.* [CMU/SEI-2013-TN-018] *Software Engineering Institute*

Felix Bachmann, Len Bass, Mark Klein (2002). *Illuminating the Fundamental Contributors to Software Architecture Quality.* [CMU/SEI-2002-TR-025] *Software Engineering Institute.*

Felix Bachmann, Len Bass, Mark Klein (2003). *Preliminary Design of ArchE: A Software Architecture Design Assistant.* [CMU/SEI-2003-TR-021] *Software Engineering Institute.*

Felix Bachmann, Len Bass, Mark Klein (2003). *Deriving Architectural Tactis: A Step Toward Methodical Architectural Design.* [CMU/SEI-2003-TR-004] *Software Engineering Institute.*

Rick Kazman, Mark Klein, Paul Clements (2000). *ATAM: Method for Architecture Evaluation.* [CMU/SEI] *Software Engineering Institute.*

Mario Barbacci, Mark Klein, Charles Weinstock (1996). *Principles for Evaluating the Quality Attributes of Software Architecture.* [CMU/SEI-96-TR-036] *Software Engineering Institute.*

Rick Kazman, Len Bass, Gregory Abowd, Mike Webb (1994): *SAAM: A Method for Analyzing the Properties of Software Architectures.* [CMU/SEI] *Software Engineering Institute.*

John McGregor, Felix Bachmann, Len Bass, Philip Blanco, Mark Klein (2007): *Using ArchE in the Classroom: One Experience.* [CMU/SEI-2007-TN-001] *Software Engineering Institute.*

Len Bass, Mark Klein, Gabriel Moreno (2001). *Applicability of General Scenarios to the Architecture Tradeoff Analysis Method.* [CMU/SEI-2001-TR-014] *Software Engineering Institute.*

Len Bass, James Ivers, Mark Klein, Paulo Merson (2005). *Reasoning Frameworks.* [CMU/SEI-2005-TR-007]

Len Bass, Paul Clements, Rick Kazman, Mark Klein (2008). *Models for Evaluating and Improving Architecture Competence*. [CMU/SEI-2008-TR-006] *Software Engineering Institute.*

Franck Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. (1996). *Pattern-oriented software architecture – A system of patterns.* John Wiley & Sons.

Paul Clements, Rick Kazman, Mark Klein (2003). *Evaluating Software Architectures: Methods and Case Studies.* Addison-Wesley.

***ISO 9126**: Software product evaluation + Quality characteristics and guidelines for their use (1991).*